COMPSs Manual

Workflows and Distributed Computing Group

Last updated : November, 2022

Table of contents

List of figures

List of tables

COMP Superscalar (COMPSs) is a **task-based programming model** which aims to ease the development of applications **for distributed infrastructures**, such as large High-Performance clusters (HPC), clouds and container managed clusters. COMPSs provides a **programming interface for** the development of the **applications and a runtime system that exploits the inherent parallelism** of applications **at execution time**.

To improve programming productivity, the **COMPSs programming model** has following **characteristics**:

- **Agnostic of the actual computing infrastructure:** COMPSs offers a model that abstracts the application from the underlying distributed infrastructure. Hence, COMPSs programs do not include any detail that could tie them to a particular platform, like deployment or resource management. This makes applications portable between infrastructures with diverse characteristics.
- **Single memory and storage space:** the memory and file system space is also abtracted in COMPSs, giving the illusion that a single memory space and single file system is available. The runtime takes care of all the necessary data transfers.
- **Standard programming languages:** COMPSs is based on the popular programming language Java, but also offers language bindings for Python (PyCOMPSs) and C/C++ applications. This makes it easier to learn the model since programmers can reuse most of their previous knowledge.
- **No APIs:** In the case of COMPSs applications in Java, the model does not require to use any special API call, pragma or construct in the application; everything is pure standard Java syntax and libraries. With regard the Python and C/C++ bindings, a small set of API calls should be used on the COMPSs applications.

This manual is divided in 12 sections:

Chapter 1

What is COMPSs?

COMP Superscalar (COMPSs) is a **task-based programming model** which aims to ease the development of applications **for distributed infrastructures**, such as large High-Performance clusters (HPC), clouds and container managed clusters. COMPSs provides a **programming interface for** the development of the **applications and a runtime system that exploits the inherent parallelism** of applications **at execution time**.

To improve programming productivity, the **COMPSs programming model** has following **characteristics**:

- **Sequential programming:** COMPSs programmers do not need to deal with the typical duties of parallelization and distribution, such as thread creation and synchronization, data distribution, messaging or fault tolerance. Instead, the model is based on sequential programming, which makes it appealing to users that either lack parallel programming expertise or are looking for better programmability.
- **Agnostic of the actual computing infrastructure:** COMPSs offers a model that abstracts the application from the underlying distributed infrastructure. Hence, COMPSs programs do not include any detail that could tie them to a particular platform, like deployment or resource management. This makes applications portable between infrastructures with diverse characteristics.
- **Single memory and storage space:** the memory and file system space is also abtracted in COMPSs, giving the illusion that a single memory space and single file system is available. The runtime takes care of all the necessary data transfers.
- **Standard programming languages:** COMPSs is based on the popular programming language Java, but also offers language bindings for Python (PyCOMPSs) and C/C++ applications. This makes it easier to learn the model since programmers can reuse most of their previous knowledge.
- **No APIs:** In the case of COMPSs applications in Java, the model does not require to use any special API call, pragma or construct in the application; everything is pure standard Java syntax and libraries. With regard the Python and C/C++ bindings, a small set of API calls should be used on the COMPSs applications.

PyCOMPSs/COMPSs can be seen as a **programming environment for the development of complex workflows**. For example, in the case of PyCOMPSs, while the task-orchestration code needs to be written in Python, it supports different types of tasks, such as Python methods, external binaries, multi-threaded (internally parallelised with alternative programming models such as OpenMP or pthreads), or multi-node (MPI applications). Thanks to the use of Python as programming language, PyCOMPSs naturally integrates well with data analytics and machine learning libraries, most of them offering a Python interface. PyCOMPSs also supports reading/writing streamed data.

At a lower level, the COMPSs runtime manages the execution of the workflow components implemented with the PyCOMPSs programming model. At runtime, it generates a **task-dependency graph** by analysing the existing data dependencies between the tasks defined in the Python code. The task-graph **encodes the existing parallelism of the workflow**, which is then scheduled and executed by the COMPSs runtime in the computing resources.

The COMPSs runtime is also able to **react to tasks failures and to exceptions** in order to adapt the behaviour accordingly. These functionalities, offer the possibility of designing a **new category of workflows with very dynamic behaviour**, that can change their configuration at execution time upon the occurrence of given events.

1.1 More information:

- Project website: http://compss.bsc.es
- Project repostory: https://github.com/bsc-wdc/compss

Chapter 2

Quickstart

2.1 Install COMPSs

- Choose the installation method:

Pip

Build from sources

Supercomputer

Docker

Local to the user

Systemwide

Requirements:

- Ensure that the required system *Dependencies* are installed.
- Check that your `JAVA_HOME` environment variable points to the Java JDK folder, that the `GRADLE_HOME` environment variable points to the GRADLE folder, and the `gradle` binary is in the `PATH` environment variable.
- Enable SSH passwordless to localhost. See *Configure SSH passwordless*.

COMPSs will be installed within the `$HOME/.local/` folder (or alternatively within the active virtual environment).

```
$ pip install pycompss -v
```

Important: Please, update the environment after installing COMPSs:

```
$ source ~/.bashrc  # or alternatively reboot the machine
```

If installed within a virtual environment, deactivate and activate it to ensure that the environment is propperly updated.

> **Warning:** If using Ubuntu 18.04 or higher, you will need to comment some lines of your `.bashrc` and do a complete logout. Please, check the *Post installation* Section for detailed instructions.

See *Installation and Administration* section for more information

Requirements:

- Ensure that the required system *Dependencies* are installed.

- Check that your `JAVA_HOME` environment variable points to the Java JDK folder, that the `GRADLE_HOME` environment variable points to the GRADLE folder, and the `gradle` binary is in the `PATH` environment variable.
- Enable SSH passwordless to localhost. See *Configure SSH passwordless*.

COMPSs will be installed within the `/usr/lib64/pythonX.Y/site-packages/pycompss/` folder.

```
$ sudo -E pip install pycompss -v
```

Important: Please, update the environment after installing COMPSs:

```
$ source /etc/profile.d/compss.sh   # or alternatively reboot the machine
```

Warning: If using Ubuntu 18.04 or higher, you will need to comment some lines of your `.bashrc` and do a complete logout. Please, check the *Post installation* Section for detailed instructions.

See *Installation and Administration* section for more information

Local to the user

Systemwide

Requirements:

- Ensure that the required system *Dependencies* are installed.
- Check that your `JAVA_HOME` environment variable points to the Java JDK folder, that the `GRADLE_HOME` environment variable points to the GRADLE folder, and the `gradle` binary is in the `PATH` environment variable.
- Enable SSH passwordless to localhost. See *Configure SSH passwordless*.

COMPSs will be installed within the `$HOME/COMPSs/` folder.

```
$ git clone https://github.com/bsc-wdc/compss.git
$ cd compss
$ ./submodules_get.sh
$ cd builders/
$ export INSTALL_DIR=$HOME/COMPSs/
$ ./buildlocal ${INSTALL_DIR}
```

The different installation options can be found in the command help.

```
$ ./buildlocal -h
```

Please, check the *Post installation* Section.

See *Installation and Administration* section for more information

Requirements:

- Ensure that the required system *Dependencies* are installed.
- Check that your `JAVA_HOME` environment variable points to the Java JDK folder, that the `GRADLE_HOME` environment variable points to the GRADLE folder, and the `gradle` binary is in the `PATH` environment variable.
- Enable SSH passwordless to localhost. See *Configure SSH passwordless*.

COMPSs will be installed within the `/opt/COMPSs/` folder.

```
$ git clone https://github.com/bsc-wdc/compss.git
$ cd compss
```

```
$ ./submodules_get.sh
$ cd builders/
$ export INSTALL_DIR=/opt/COMPSs/
$ sudo -E ./buildlocal ${INSTALL_DIR}
```

The different installation options can be found in the command help.

```
$ ./buildlocal -h
```

Please, check the *Post installation* Section.

See *Installation and Administration* section for more information

Please, check the *Supercomputers* section.

COMPSs can be used within Docker using the PyCOMPSs CLI.

Requirements (Optional):

- docker >= 17.12.0-ce
- Python 3
- pip
- docker package for Python

Since the PyCOMPSs CLI package is available in Pypi (pycompss-cli), it can be easly installed with `pip` as follows:

```
$ python3 -m pip install pycompss-cli
```

A complete guide about the PyCOMPSs CLI installation and usage can be found in the *PyCOMPSs CLI* Section.

Tip: Please, check the PyCOMPSs CLI *Installation* Section for the further information with regard to the requirements installation and troubleshooting.

Warning: For macOS distributions, only installations **local to the user** are supported (both with pip and building from sources). This is due to the System Integrity Protection (SIP) implemented in the newest versions of macOS, that does not allow modifications in the `/System` directory, even when having root permissions in the machine.

2.2 Write your first app

Choose your flavour:

Java

Python

C/C++

Application Overview

A COMPSs application is composed of three parts:

- **Main application code:** the code that is executed sequentially and contains the calls to the user-selected methods that will be executed by the COMPSs runtime as asynchronous parallel tasks.
- **Remote methods code:** the implementation of the tasks.
- **Task definition interface:** It is a Java annotated interface which declares the methods to be run as remote tasks along with metadata information needed by the runtime to properly schedule the tasks.

The main application file name has to be the same of the main class and starts with capital letter, in this case it is **Simple.java**. The Java annotated interface filename is *application name + Itf.java*, in this case it is **SimpleItf.java**. And the code that implements the remote tasks is defined in the *application name + Impl.java* file, in this case it is **SimpleImpl.java**.

All code examples are in the `/home/compss/tutorial_apps/java/` folder of the development environment.

Main application code

In COMPSs, the user's application code is kept unchanged, no API calls need to be included in the main application code in order to run the selected tasks on the nodes.

The COMPSs runtime is in charge of replacing the invocations to the user-selected methods with the creation of remote tasks also taking care of the access to files where required. Let's consider the Simple application example that takes an integer as input parameter and increases it by one unit.

The main application code of Simple application is shown in the following code block. It is executed sequentially until the call to the **increment()** method. COMPSs, as mentioned above, replaces the call to this method with the generation of a remote task that will be executed on an available node.

Code 1: Simple in Java (Simple.java)

```java
package simple;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import simple.SimpleImpl;

public class Simple {

  public static void main(String[] args) {
    String counterName = "counter";
    int initialValue = args[0];

    //--------------------------------------------------------------//
    // Creation of the file which will contain the counter variable //
    //--------------------------------------------------------------//
    try {
        FileOutputStream fos = new FileOutputStream(counterName);
        fos.write(initialValue);
        System.out.println("Initial counter value is " + initialValue);
        fos.close();
    }catch(IOException ioe) {
        ioe.printStackTrace();
    }

    //---------------------------------------------//
    //           Execution of the program          //
```

(continues on next page)

```java
    //---------------------------------------------//
    SimpleImpl.increment(counterName);

    //---------------------------------------------//
    //    Reading from an object stored in a File  //
    //---------------------------------------------//
    try {
        FileInputStream fis = new FileInputStream(counterName);
        System.out.println("Final counter value is " + fis.read());
        fis.close();
    }catch(IOException ioe) {
        ioe.printStackTrace();
    }
  }
}
```

Remote methods code

The following code contains the implementation of the remote method of the *Simple* application that will be executed remotely by COMPSs.

Code 2: Simple Implementation (SimpleImpl.java)

```java
package simple;

import  java.io.FileInputStream;
import  java.io.FileOutputStream;
import  java.io.IOException;
import  java.io.FileNotFoundException;

public class SimpleImpl {
  public static void increment(String counterFile) {
    try{
      FileInputStream fis = new FileInputStream(counterFile);
      int count = fis.read();
      fis.close();
      FileOutputStream fos = new FileOutputStream(counterFile);
      fos.write(++count);
      fos.close();
    }catch(FileNotFoundException fnfe){
      fnfe.printStackTrace();
    }catch(IOException ioe){
      ioe.printStackTrace();
    }
  }
}
```

Task definition interface

This Java interface is used to declare the methods to be executed remotely along with Java annotations that specify the necessary metadata about the tasks. The metadata can be of three different types:

1. For each parameter of a method, the data type (currently *File* type, primitive types and the *String* type are supported) and its directions (IN, OUT, INOUT, COMMUTATIVE or CONCURRENT).
2. The Java class that contains the code of the method.
3. The constraints that a given resource must fulfill to execute the method, such as the number of processors or main memory size.

The task description interface of the Simple app example is shown in the following figure. It includes the description of the *Increment()* method metadata. The method interface contains a single input parameter, a string containing a path to the file counterFile. In this example there are constraints on the minimum number of processors and minimum memory size needed to run the method.

Code 3: Interface of the Simple application (SimpleItf.java)

```java
package simple;

import  es.bsc.compss.types.annotations.Constraints;
import  es.bsc.compss.types.annotations.task.Method;
import  es.bsc.compss.types.annotations.Parameter;
import  es.bsc.compss.types.annotations.parameter.Direction;
import  es.bsc.compss.types.annotations.parameter.Type;

public interface SimpleItf {

  @Constraints(computingUnits = "1", memorySize = "0.3")
  @Method(declaringClass = "simple.SimpleImpl")
  void increment(
      @Parameter(type = Type.FILE, direction = Direction.INOUT)
      String file
  );

}
```

Application compilation

A COMPSs Java application needs to be packaged in a *jar* file containing the class files of the main code, of the methods implementations and of the *Itf* annotation. This jar package can be generated using the commands available in the Java SDK or creating your application as a Apache Maven project.

To integrate COMPSs in the maven compile process you just need to add the *compss-api* artifact as dependency in the application project.

```xml
<dependencies>
    <dependency>
        <groupId>es.bsc.compss</groupId>
        <artifactId>compss-api</artifactId>
        <version>${compss.version}</version>
    </dependency>
</dependencies>
```

To build the jar in the maven case use the following command

```
$ mvn package
```

Next we provide a set of commands to compile the Java Simple application (detailed at *Java Sample applications*).

```
$ cd tutorial_apps/java/simple/src/main/java/simple/
$~/tutorial_apps/java/simple/src/main/java/simple$ javac *.java
$~/tutorial_apps/java/simple/src/main/java/simple$ cd ..
$~/tutorial_apps/java/simple/src/main/java$ jar cf simple.jar simple/
$~/tutorial_apps/java/simple/src/main/java$ mv ./simple.jar ../../../jar/
```

In order to properly compile the code, the CLASSPATH variable has to contain the path of the *compss-engine.jar* package. The default COMPSs installation automatically add this package to the CLASSPATH; please check that your environment variable CLASSPATH contains the *compss-engine.jar* location by running the following command:

```
$ echo $CLASSPATH | grep compss-engine
```

If the result of the previous command is empty it means that you are missing the *compss-engine.jar* package in your classpath. We recommend to automatically load the variable by editing the *.bashrc* file:

```
$ echo "# COMPSs variables for Java compilation" >> ~/.bashrc
$ echo "export CLASSPATH=$CLASSPATH:/opt/COMPSs/Runtime/compss-engine.jar" >> ~/.bashrc
```

Application execution

A Java COMPSs application is executed through the *runcompss* script. An example of an invocation of the script is:

```
$ runcompss --classpath=/home/compss/tutorial_apps/java/simple/jar/simple.jar simple.Simple 1
```

A comprehensive description of the *runcompss* command is available in the *Executing COMPSs applications* section.

In addition to Java, COMPSs supports the execution of applications written in other languages by means of bindings. A binding manages the interaction of the no-Java application with the COMPSs Java runtime, providing the necessary language translation.

Let's write your first Python application parallelized with PyCOMPSs. Consider the following code:

Code 4: `increment.py`

```python
import time
from pycompss.api.api import compss_wait_on
from pycompss.api.task import task

@task(returns=1)
def increment(value):
  time.sleep(value * 2)   # mimic some computational time
  return value + 1

def main():
    values = [1, 2, 3, 4]
    start_time = time.time()
    for pos in range(len(values)):
        values[pos] = increment(values[pos])
    values = compss_wait_on(values)
    assert values == [2, 3, 4, 5]
    print(values)
    print("Elapsed time: " + str(time.time() - start_time))

if __name__=='__main__':
    main()
```

This code increments the elements of an array (`values`) by calling iteratively to the `increment` function. The increment function sleeps the number of seconds indicated by the `value` parameter to represent some computational time. On a normal python execution, each element of the array will be incremented after the other (sequentially), accumulating the computational time. PyCOMPSs is able to parallelize this loop thanks to its `@task` decorator, and synchronize the results with the `compss_wait_on` API call.

Note: If you are using the PyCOMPSs CLI (pycompss-cli), it is time to deploy the COMPSs environment within your current folder:

```
$ pycompss init
```

Please, be aware that the first time needs to download the docker image from the repository, and it may take a while.

Copy and paste the increment code it into `increment.py`.

Execution

Now let's execute `increment.py`. To this end, we will use the `runcompss` script provided by COMPSs:

```
$ runcompss -g increment.py
  [Output in next step]
```

Or alternatively, the `pycompss run` command if using the PyCOMPSs CLI (which wraps the `runcompss` command and launches it within the COMPSs' docker container):

```
$ pycompss run -g increment.py
  [Output in next step]
```

Note: The `-g` flag enables the task dependency graph generation (*used later*).

The `runcompss` command has a lot of supported options that can be checked with the `-h` flag. They can also be used within the `pycompss run` command.

Tip: It is possible to run also with the `python` command using the `pycompss` module, which accepts the same flags as `runcompss`:

```
$ python -m pycompss -g increment.py   # Parallel execution
  [Output in next step]
```

Having PyCOMPSs installed also enables to run the same code sequentially without the need of removing the PyCOMPSs syntax.

```
$ python increment.py   # Sequential execution
  [2, 3, 4, 5]
  Elapsed time: 20.0161030293
```

Output

```
$ runcompss -g increment.py
  [  INFO] Inferred PYTHON language
  [  INFO] Using default location for project file: /opt/COMPSs/Runtime/configuration/xml/
↪projects/default_project.xml
  [  INFO] Using default location for resources file: /opt/COMPSs/Runtime/configuration/xml/
↪resources/default_resources.xml
  [  INFO] Using default execution type: compss


  ----------------- Executing increment.py --------------------------

  WARNING: COMPSs Properties file is null. Setting default values
  [(433)    API]  -  Starting COMPSs Runtime v3.1
  [2, 3, 4, 5]
  Elapsed time: 11.5068922043
  [(4389)    API]  -  Execution Finished


  ------------------------------------------------------------
```

Nice! it run successfully in my 8 core laptop, we have the expected output, and PyCOMPSs has been able to run the `increment.py` application in almost half of the time required by the sequential execution. *What happened under the hood?*

COMPSs started a master and one worker (by default configured to execute up to four tasks at the same time) and executed the application (offloading the tasks execution to the worker).

Let's check the task dependency graph to see the parallelism that COMPSs has extracted and taken advantage of.

Task dependency graph

COMPSs stores the generated task dependecy graph within the `$HOME/.COMPSs/<APP_NAME>_<00-99>/monitor` directory in dot format. The generated graph is `complete_graph.dot` file, which can be displayed with any dot viewer.

Tip: COMPSs provides the `compss_gengraph` script which converts the given dot file into pdf.

```
$ cd $HOME/.COMPSs/increment.py_01/monitor
$ compss_gengraph complete_graph.dot
$ evince complete_graph.pdf   # or use any other pdf viewer you like
```

It is also available within the PyCOMPSs CLI:

```
$ cd $HOME/.COMPSs/increment.py_01/monitor
$ pycompss gengraph complete_graph.dot
$ evince complete_graph.pdf   # or use any other pdf viewer you like
```

And you should see:

COMPSs has detected that the increment of each element is independent, and consequently, that all of them can be done in parallel. In this particular application, there are four `increment` tasks, and since the worker is able to run four tasks at the same time, all of them can be executed in parallel saving precious time.
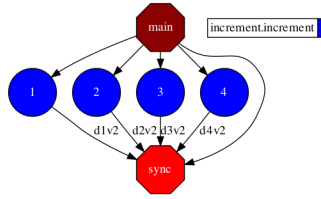
Figure 1: The dependency graph of the increment application

Check the performance

Let's run it again with the tracing flag enabled:

```
$ runcompss -t increment.py
  [  INFO] Inferred PYTHON language
  [  INFO] Using default location for project file: /opt/COMPSs//Runtime/configuration/xml/
→projects/default_project.xml
  [  INFO] Using default location for resources file: /opt/COMPSs//Runtime/configuration/xml/
→resources/default_resources.xml
  [  INFO] Using default execution type: compss

  ----------------- Executing increment.py --------------------------

  Welcome to Extrae 3.5.3

  [... Extrae prolog ...]

  WARNING: COMPSs Properties file is null. Setting default values
  [(434)    API]  -  Starting COMPSs Runtime v3.1
  [2, 3, 4, 5]
  Elapsed time: 13.1016821861

  [... Extrae eplilog ...]

  mpi2prv: Congratulations! ./trace/increment.py_compss_trace_1587562240.prv has been␣
→generated.
  [(24117)   API]  -  Execution Finished

  ------------------------------------------------------------
```

The execution has finished successfully and the trace has been generated in the `$HOME/.COMPSs/<APP_NAME>_-<00-99>/trace` directory in prv format, which can be displayed and analysed with PARAVER.

```
$ cd $HOME/.COMPSs/increment.py_02/trace
$ wxparaver increment.py_compss_trace_*.prv
```

Note: In the case of using the PyCOMPSs CLI, the trace will be generated in the `.COMPSs/<APP_NAME>_-<00-99>/trace` directory:

```
$ cd .COMPSs/increment.py_02/trace
$ wxparaver increment.py_compss_trace_*.prv
```

Once Paraver has started, lets visualize the tasks:

- Click in `File` and then in `Load Configuration`
- Look for `/PATH/TO/COMPSs/Dependencies/paraver/cfgs/compss_tasks.cfg` and click `Open`.

Note: In the case of using the PyCOMPSs CLI, the configuration files can be obtained by downloading them from the COMPSs repositoy.

And you should see:
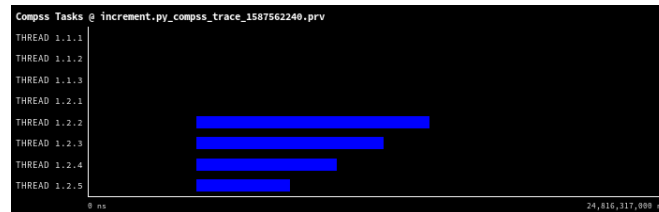
Figure 2: Trace of the increment application

The X axis represents the time, and the Y axis the deployed processes (the first three (1.1.1-1.1.3) belong to the master and the fourth belongs to the master process in the worker (1.2.1) whose events are shown with the `compss_runtime.cfg` configuration file).

The `increment` tasks are depicted in blue. We can quickly see that the four *increment* tasks have been executed in parallel (one per core), and that their lengths are different (depending on the computing time of the task represented by the `time.sleep(value * 2)` line).

Paraver is a very powerful tool for performance analysis. For more information, check the *Tracing* Section.

Note: If you are using the PyCOMPSs CLI, it is time to stop the COMPSs environment:

```
$ pycompss stop
```

Application Overview

As in Java, the application code is divided in 3 parts: the Task definition interface, the main code and task implementations. These files must have the following notation,: <app_ame>.idl, for the interface file, <app_name>.cc for the main code and <app_name>-functions.cc for task implementations. Next paragraphs provide an example of how to define this files for matrix multiplication parallelised by blocks.

Task Definition Interface

As in Java the user has to provide a task selection by means of an interface. In this case the interface file has the same name as the main application file plus the suffix "idl", i.e. Matmul.idl, where the main file is called Matmul.cc.

Code 5: Matmul.idl

```
interface Matmul
{
    // C functions
    void initMatrix(inout Matrix matrix,
                    in int mSize,
                    in int nSize,
                    in double val);
```

(continues on next page)

```
      void multiplyBlocks(inout Block block1,
                          inout Block block2,
                          inout Block block3);
};
```

The syntax of the interface file is shown in the previous code. Tasks can be declared as classic C function prototypes, this allow to keep the compatibility with standard C applications. In the example, initMatrix and multiplyBlocks are functions declared using its prototype, like in a C header file, but this code is C++ as they have objects as parameters (objects of type Matrix, or Block).

The grammar for the interface file is:

```
["static"] return-type task-name ( parameter {, parameter }* );

return-type = "void" | type

ask-name = <qualified name of the function or method>

parameter = direction type parameter-name

direction = "in" | "out" | "inout"

type = "char" | "int" | "short" | "long" | "float" | "double" | "boolean" |
       "char[<size>]" | "int[<size>]" | "short[<size>]" | "long[<size>]" |
       "float[<size>]" | "double[<size>]" | "string" | "File" | class-name

class-name = <qualified name of the class>
```

Main Program

The following code shows an example of matrix multiplication written in C++.

Code 6: Matrix multiplication

```cpp
#include "Matmul.h"
#include "Matrix.h"
#include "Block.h"
int N; //MSIZE
int M; //BSIZE
double val;
int main(int argc, char **argv)
{
      Matrix A;
      Matrix B;
      Matrix C;

      N = atoi(argv[1]);
      M = atoi(argv[2]);
      val = atof(argv[3]);

      compss_on();

      A = Matrix::init(N,M,val);

      initMatrix(&B,N,M,val);
```

```
        initMatrix(&C,N,M,0.0);

        cout << "Waiting for initialization...\n";

        compss_wait_on(B);
        compss_wait_on(C);

        cout << "Initialization ends...\n";

        C.multiply(A, B);

        compss_off();
        return 0;
}
```

The developer has to take into account the following rules:

1. A header file with the same name as the main file must be included, in this case **Matmul.h**. This header file is automatically generated by the binding and it contains other includes and type-definitions that are required.
2. A call to the **compss_on** binding function is required to turn on the COMPSs runtime.
3. As in C language, out or inout parameters should be passed by reference by means of the "**&**" operator before the parameter name.
4. Synchronization on a parameter can be done calling the **compss_wait_on** binding function. The argument of this function must be the variable or object we want to synchronize.
5. There is an **implicit synchronization** in the init method of Matrix. It is not possible to know the address of "A" before exiting the method call and due to this it is necessary to synchronize before for the copy of the returned value into "A" for it to be correct.
6. A call to the **compss_off** binding function is required to turn off the COMPSs runtime.

Functions file

The implementation of the tasks in a C or C++ program has to be provided in a functions file. Its name must be the same as the main file followed by the suffix "-functions". In our case Matmul-functions.cc.

```
#include "Matmul.h"
#include "Matrix.h"
#include "Block.h"

void initMatrix(Matrix *matrix,int mSize,int nSize,double val){
    *matrix = Matrix::init(mSize, nSize, val);
}

void multiplyBlocks(Block *block1,Block *block2,Block *block3){
    block1->multiply(*block2, *block3);
}
```

In the previous code, class methods have been encapsulated inside a function. This is useful when the class method returns an object or a value and we want to avoid the explicit synchronization when returning from the method.

Additional source files

Other source files needed by the user application must be placed under the directory "**src**". In this directory the programmer must provide a **Makefile** that compiles such source files in the proper way. When the binding compiles the whole application it will enter into the src directory and execute the Makefile.

It generates two libraries, one for the master application and another for the worker application. The directive COMPSS_MASTER or COMPSS_WORKER must be used in order to compile the source files for each type of library. Both libraries will be copied into the lib directory where the binding will look for them when generating the master and worker applications.

Application Compilation

The user command "**compss_build_app**" compiles both master and worker for a single architecture (e.g. x86-64, armhf, etc). Thus, whether you want to run your application in Intel based machine or ARM based machine, this command is the tool you need.

When the target is the native architecture, the command to execute is very simple;

```
$~/matmul_objects> compss_build_app Matmul
[ INFO ] Java libraries are searched in the directory: /usr/lib/jvm/java-1.8.0-openjdk-amd64//
↪jre/lib/amd64/server
[ INFO ] Boost libraries are searched in the directory: /usr/lib/


...


[Info] The target host is: x86_64-linux-gnu

Building application for master...
g++ -g -O3 -I. -I/Bindings/c/share/c_build/worker/files/ -c Block.cc Matrix.cc
ar rvs libmaster.a Block.o Matrix.o
ranlib libmaster.a

Building application for workers...
g++ -DCOMPSS_WORKER -g -O3 -I. -I/Bindings/c/share/c_build/worker/files/ -c Block.cc -o Block.
↪o
g++ -DCOMPSS_WORKER -g -O3 -I. -I/Bindings/c/share/c_build/worker/files/ -c Matrix.cc -o␣
↪Matrix.o
ar rvs libworker.a Block.o Matrix.o
ranlib libworker.a


...

Command successful.
```

Application Execution

The following environment variables must be defined before executing a COMPSs C/C++ application:

JAVA_HOME Java JDK installation directory (e.g. /usr/lib/jvm/java-8-openjdk/)

After compiling the application, two directories, master and worker, are generated. The master directory contains a binary called as the main file, which is the master application, in our example is called Matmul. The worker directory contains another binary called as the main file followed by the suffix "-worker", which is the worker application, in our example is called Matmul-worker.

The `runcompss` script has to be used to run the application:

```
$ runcompss /home/compss/tutorial_apps/c/matmul_objects/master/Matmul 3 4 2.0
```

The complete list of options of the runcompss command is available in Section *Executing COMPSs applications*.

Task Dependency Graph

COMPSs can generate a task dependency graph from an executed code. It is indicating by a

```
$ runcompss -g /home/compss/tutorial_apps/c/matmul_objects/master/Matmul 3 4 2.0
```

The generated task dependency graph is stored within the $HOME/.COMPSs/<APP_NAME>_<00-99>/monitor directory in dot format. The generated graph is complete_graph.dot file, which can be displayed with any dot viewer. COMPSs also provides the compss_gengraph script which converts the given dot file into pdf.

```
$ cd $HOME/.COMPSs/Matmul_02/monitor
$ compss_gengraph complete_graph.dot
$ evince complete_graph.pdf  # or use any other pdf viewer you like
```

The following figure depicts the task dependency graph for the Matmul application in its object version with 3x3 blocks matrices, each one containing a 4x4 matrix of doubles. Each block in the result matrix accumulates three block multiplications, i.e. three multiplications of 4x4 matrices of doubles.

Figure 3: Matmul Execution Graph.

The light blue circle corresponds to the initialization of matrix "A" by means of a method-task and it has an implicit synchronization inside. The dark blue circles correspond to the other two initializations by means of function-tasks; in this case the synchronizations are explicit and must be provided by the developer after the task call. Both implicit and explicit synchronizations are represented as red circles.

Each green circle is a partial matrix multiplication of a set of 3. One block from matrix "A" and the correspondent one from matrix "B". The result is written in the right block in "C" that accumulates the partial block multiplications. Each multiplication set has an explicit synchronization. All green tasks are method-tasks and they are executed in parallel.

2.3 Useful information

Choose your flavour:

Java

Python

C/C++

- Syntax detailed information -> *Java*
- Constraint definition -> *Constraints*
- Execution details -> *Executing COMPSs applications*
- Graph, tracing and monitoring facilities -> *Tools*
- Other execution environments (Supercomputers, Docker, etc.) -> *Supercomputers*
- Performance analysis -> *Tracing*
- Troubleshooting -> *Troubleshooting*
- Sample applications -> *Java Sample applications*
- Using COMPSs with persistent storage frameworks (e.g. dataClay, Hecuba) -> *Persistent Storage*

- Syntax detailed information -> *Python Binding*
- Constraint definition -> *Constraints*
- Execution details -> *Executing COMPSs applications*
- Graph, tracing and monitoring facilities -> *Tools*
- Other execution environments (Supercomputers, Docker, etc.) -> *Supercomputers*
- Performance analysis -> *Tracing*
- Troubleshooting -> *Troubleshooting*
- Sample applications -> *Python Sample applications*
- Using COMPSs with persistent storage frameworks (e.g. dataClay, Hecuba) -> *Persistent Storage*

- Syntax detailed information -> *C/C++ Binding*
- Constraint definition -> *Constraints*
- Execution details -> *Executing COMPSs applications*
- Graph, tracing and monitoring facilities -> *Tools*
- Other execution environments (Supercomputers, Docker, etc.) -> *Supercomputers*
- Performance analysis -> *Tracing*
- Troubleshooting -> *Troubleshooting*
- Sample applications -> *C/C++ Sample applications*

Chapter 3

Installation and Administration

This section is intended to walk you through the COMPSs installation.

3.1 Dependencies

Next we provide a list of dependencies for installing COMPSs package. The exact names may vary depending on the Linux distribution but this list provides a general overview of the COMPSs dependencies. For specific information about your distribution please check the *Depends* section at your package manager (apt, yum, zypper, etc.).

Table 1: COMPSs dependencies

Module	Dependencies
COMPSs Runtime	**openjdk-8-jre, graphviz, xdg-utils, openssh-server**
COMPSs Python Binding	libtool, automake, build-essential, python (>=3.6), python3-dev, python3-setuptools
COMPSs C/C++ Binding	libtool, automake, build-essential, libboost-all-dev, libxml2-dev
COMPSs Tracing	libxml2 (>= 2.5), libxml2-dev (>= 2.5), gfortran, papi

---

Tip: For macOS, we strongly recommend to use the Homebrew package manager, since it includes the majority of dependencies needed. In other package managers, such as MacPorts, quite some dependencies may be missing as packages, which will force you to have to install them from their source codes.

As an example for some distributions and versions:

Ubuntu

OpenSuse

Fedora

Debian

CentOS

20.04

18.04

16.04

Ubuntu 20.04 dependencies installation commands:

```
$ sudo apt-get install -y openjdk-8-jdk graphviz xdg-utils libtool automake build-essential␣
→python python-dev python3 python3-dev libboost-serialization-dev libboost-iostreams-dev ␣
→libxml2 libxml2-dev csh gfortran libgmp3-dev flex bison texinfo python3-pip libpapi-dev
$ sudo wget https://services.gradle.org/distributions/gradle-5.4.1-bin.zip -O /opt/gradle-5.4.
→1-bin.zip
$ sudo unzip /opt/gradle-5.4.1-bin.zip -d /opt
```

> **Attention:**   Before installing it is important to have a proper `JAVA_HOME` environment variable definition. This variable must contain a valid path to a Java JDK (as a remark, it must point to a JDK, not JRE). So, please, export this variable and include it into your `.bashrc`:
>
> ```
> $ echo 'export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/' >> ~/.bashrc
> $ export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/
> ```

Ubuntu 18.04 dependencies installation commands:

```
$ sudo apt-get install -y openjdk-8-jdk graphviz xdg-utils libtool automake build-essential␣
→python python-dev python3 python3-dev libboost-serialization-dev libboost-iostreams-dev ␣
→libxml2 libxml2-dev csh gfortran libgmp3-dev flex bison texinfo python3-pip libpapi-dev
$ sudo wget https://services.gradle.org/distributions/gradle-5.4.1-bin.zip -O /opt/gradle-5.4.
→1-bin.zip
$ sudo unzip /opt/gradle-5.4.1-bin.zip -d /opt
```

> **Attention:**   Before installing it is important to have a proper `JAVA_HOME` environment variable definition. This variable must contain a valid path to a Java JDK (as a remark, it must point to a JDK, not JRE). So, please, export this variable and include it into your `.bashrc`:
>
> ```
> $ echo 'export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/' >> ~/.bashrc
> $ export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/
> ```

Ubuntu 16.04 dependencies installation commands:

```
$ sudo apt-get install -y openjdk-8-jdk graphviz xdg-utils libtool automake build-essential␣
→libboost-serialization-dev libboost-iostreams-dev  libxml2 libxml2-dev csh gfortran python-
→pip libpapi-dev
$ sudo wget https://services.gradle.org/distributions/gradle-5.4.1-bin.zip -O /opt/gradle-5.4.
→1-bin.zip
$ sudo unzip /opt/gradle-5.4.1-bin.zip -d /opt
```

> **Attention:**   Before installing it is important to have a proper `JAVA_HOME` environment variable definition. This variable must contain a valid path to a Java JDK (as a remark, it must point to a JDK, not JRE). So, please, export this variable and include it into your `.bashrc`:
>
> ```
> $ echo 'export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/' >> ~/.bashrc
> $ export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/
> ```

Tumbleweed

Leap 15.X

42.2

OpenSuse Tumbleweed dependencies installation commands:

```
$ sudo zypper install --type pattern -y devel_basis
$ sudo zypper install -y java-1_8_0-openjdk-headless java-1_8_0-openjdk java-1_8_0-openjdk-
↪devel graphviz xdg-utils python python-devel python3 python3-devel python3-decorator␣
↪libtool automake libboost_headers1_71_0-devel libboost_serialization1_71_0 libboost_
↪iostreams1_71_0  libxml2-2 libxml2-devel tcsh gcc-fortran papi libpapi gcc-c++ libpapi papi␣
↪papi-devel gmp-devel
$ sudo wget https://services.gradle.org/distributions/gradle-5.4.1-bin.zip -O /opt/gradle-5.4.
↪1-bin.zip
$ sudo unzip /opt/gradle-5.4.1-bin.zip -d /opt
```

Attention: Before installing it is important to have a proper `JAVA_HOME` environment variable definition. This variable must contain a valid path to a Java JDK (as a remark, it must point to a JDK, not JRE). So, please, export this variable and include it into your `.bashrc`:

```
$ echo 'export JAVA_HOME=/usr/lib64/jvm/java-1.8.0-openjdk/' >> ~/.bashrc
$ export JAVA_HOME=/usr/lib64/jvm/java-1.8.0-openjdk/
```

OpenSuse Leap 15.X dependencies installation commands:

```
$ sudo zypper install --type pattern -y devel_basis
$ sudo zypper install -y java-1_8_0-openjdk-headless java-1_8_0-openjdk java-1_8_0-openjdk-
↪devel graphviz xdg-utils python python-devel python-decorator python3 python3-devel python3-
↪decorator libtool automake libboost_headers1_66_0-devel libboost_serialization1_66_0␣
↪libboost_iostreams1_66_0  libxml2-2 libxml2-devel tcsh gcc-fortran papi libpapi gcc-c++␣
↪libpapi papi papi-devel gmp-devel
$ sudo wget https://services.gradle.org/distributions/gradle-5.4.1-bin.zip -O /opt/gradle-5.4.
↪1-bin.zip
$ sudo unzip /opt/gradle-5.4.1-bin.zip -d /opt
```

Attention: Before installing it is important to have a proper `JAVA_HOME` environment variable definition. This variable must contain a valid path to a Java JDK (as a remark, it must point to a JDK, not JRE). So, please, export this variable and include it into your `.bashrc`:

```
$ echo 'export JAVA_HOME=/usr/lib64/jvm/java-1.8.0-openjdk/' >> ~/.bashrc
$ export JAVA_HOME=/usr/lib64/jvm/java-1.8.0-openjdk/
```

OpenSuse 42.2 dependencies installation commands:

```
$ sudo zypper install --type pattern -y devel_basis
$ sudo zypper install -y java-1_8_0-openjdk-headless java-1_8_0-openjdk java-1_8_0-openjdk-
↪devel graphviz xdg-utils python python-devel libpython2_7-1_0 python-decorator libtool␣
↪automake boost-devel libboost_serialization1_54_0 libboost_iostreams1_54_0 libxml2-2␣
↪libxml2-devel tcsh gcc-fortran python-pip papi libpapi gcc-c++ libpapi papi papi-devel gmp-
↪devel
$ sudo wget https://services.gradle.org/distributions/gradle-5.4.1-bin.zip -O /opt/gradle-5.4.
↪1-bin.zip
$ sudo unzip /opt/gradle-5.4.1-bin.zip -d /opt
```

Warning: OpenSuse provides Python 3.4 from its repositories, which is not supported by the COMPSs python binding. Please, update Python 3 (`python` and `python-devel`) to a higher version if you expect to install COMPSs from sources.

Alternatively, you can use a virtual environment.

> **Attention:** Before installing it is important to have a proper `JAVA_HOME` environment variable definition. This variable must contain a valid path to a Java JDK (as a remark, it must point to a JDK, not JRE). So, please, export this variable and include it into your `.bashrc`:
>
> ```
> $ echo 'export JAVA_HOME=/usr/lib64/jvm/java-1.8.0-openjdk/' >> ~/.bashrc
> $ export JAVA_HOME=/usr/lib64/jvm/java-1.8.0-openjdk/
> ```

32

25

Fedora 32 dependencies installation commands:

```
$ sudo dnf install -y java-1.8.0-openjdk java-1.8.0-openjdk-devel graphviz xdg-utils libtool␣
↪automake python27 python3 python3-devel boost-devel boost-serialization boost-iostreams␣
↪libxml2 libxml2-devel gcc gcc-c++ gcc-gfortran tcsh @development-tools bison flex texinfo␣
↪papi papi-devel gmp-devel
$ # If the libxml softlink is not created during the installation of libxml2, the COMPSs␣
↪installation may fail.
$ # In this case, the softlink has to be created manually with the following command:
$ sudo ln -s /usr/include/libxml2/libxml/ /usr/include/libxml
$ sudo wget https://services.gradle.org/distributions/gradle-5.4.1-bin.zip -O /opt/gradle-5.4.
↪1-bin.zip
$ sudo unzip /opt/gradle-5.4.1-bin.zip -d /opt
```

> **Attention:** Before installing it is important to have a proper `JAVA_HOME` environment variable definition. This variable must contain a valid path to a Java JDK (as a remark, it must point to a JDK, not JRE). So, please, export this variable and include it into your `.bashrc`:
>
> ```
> $ echo 'export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk/' >> ~/.bashrc
> $ export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk/
> ```

Fedora 25 dependencies installation commands:

```
$ sudo dnf install -y java-1.8.0-openjdk java-1.8.0-openjdk-devel graphviz xdg-utils libtool␣
↪automake python python-libs python-pip python-devel python2-decorator boost-devel boost-
↪serialization boost-iostreams libxml2 libxml2-devel gcc gcc-c++ gcc-gfortran tcsh␣
↪@development-tools redhat-rpm-config papi
$ # If the libxml softlink is not created during the installation of libxml2, the COMPSs␣
↪installation may fail.
$ # In this case, the softlink has to be created manually with the following command:
$ sudo ln -s /usr/include/libxml2/libxml/ /usr/include/libxml
$ sudo wget https://services.gradle.org/distributions/gradle-5.4.1-bin.zip -O /opt/gradle-5.4.
↪1-bin.zip
$ sudo unzip /opt/gradle-5.4.1-bin.zip -d /opt
```

> **Attention:** Before installing it is important to have a proper `JAVA_HOME` environment variable definition. This variable must contain a valid path to a Java JDK (as a remark, it must point to a JDK, not JRE). So, please, export this variable and include it into your `.bashrc`:
>
> ```
> $ echo 'export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk/' >> ~/.bashrc
> $ export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk/
> ```

8

Debian 8 dependencies installation commands:

```
$ su -
$ echo "deb http://ppa.launchpad.net/webupd8team/java/ubuntu xenial main" | tee /etc/apt/
↪sources.list.d/webupd8team-java.list
$ echo "deb-src http://ppa.launchpad.net/webupd8team/java/ubuntu xenial main" | tee -a /etc/
↪apt/sources.list.d/webupd8team-java.list
$ apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys EEA14886
$ apt-get update
$ apt-get install oracle-java8-installer
$ apt-get install graphviz xdg-utils libtool automake build-essential python python-decorator␣
↪python-pip python-dev libboost-serialization1.55.0 libboost-iostreams1.55.0 libxml2 libxml2-
↪dev libboost-dev csh gfortran papi-tools
$ wget https://services.gradle.org/distributions/gradle-5.4.1-bin.zip -O /opt/gradle-5.4.1-
↪bin.zip
$ unzip /opt/gradle-5.4.1-bin.zip -d /opt
```

Attention: Before installing it is important to have a proper `JAVA_HOME` environment variable definition. This variable must contain a valid path to a Java JDK (as a remark, it must point to a JDK, not JRE). A possible value is the following:

```
$ echo $JAVA_HOME
/usr/lib64/jvm/java-openjdk/
```

So, please, check its location, export this variable and include it into your `.bashrc` if it is not already available with the previous command.

```
$ echo 'export JAVA_HOME=/usr/lib64/jvm/java-openjdk/' >> ~/.bashrc
$ export JAVA_HOME=/usr/lib64/jvm/java-openjdk/
```

7

CentOS 7 dependencies installation commands:

```
$ sudo rpm -iUvh https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
$ sudo yum -y update
$ sudo yum install java-1.8.0-openjdk java-1.8.0-openjdk-devel graphviz xdg-utils libtool␣
↪automake python python-libs python-pip python-devel python2-decorator boost-devel boost-
↪serialization boost-iostreams libxml2 libxml2-devel gcc gcc-c++ gcc-gfortran tcsh␣
↪@development-tools redhat-rpm-config papi
$ sudo pip install decorator
```

Attention: Before installing it is important to have a proper `JAVA_HOME` environment variable definition. This variable must contain a valid path to a Java JDK (as a remark, it must point to a JDK, not JRE). A possible value is the following:

```
$ echo $JAVA_HOME
/usr/lib64/jvm/java-openjdk/
```

So, please, check its location, export this variable and include it into your `.bashrc` if it is not already available with the previous command.

```
$ echo 'export JAVA_HOME=/usr/lib64/jvm/java-openjdk/' >> ~/.bashrc
$ export JAVA_HOME=/usr/lib64/jvm/java-openjdk/
```

---

macOS Monterey

macOS Monterey dependencies installation commands:

3.1.   Dependencies   **25**

Although many packages can be installed with Homebrew, some of them will have to be installed manually from their source files. It is also important to mention that, some package names may be slightly different in Homebrew, compared to Linux distributions, thus, some previous search for equivalences may be required. Our tested installation sequence was:

```
$ brew install openjdk@8 graphviz libxslt xmlto libtool automake coreutils util-linux boost
$ sudo ln -sfn /usr/local/opt/openjdk@8/libexec/openjdk.jdk /Library/Java/JavaVirtualMachines/
↪openjdk-8.jdk
```

And xdg-utils had to be installed by hand (after installing libxslt and xmlto):

```
$ export XML_CATALOG_FILES="/usr/local/etc/xml/catalog"
$ git clone git://anongit.freedesktop.org/xdg/xdg-utils
$ cd xdg-utils
$ ./configure --prefix=/usr/local
$ make ; make install
```

Warning: Tracing is not yet available for macOS, therefore, its dependencies do not need to be installed.

Attention: Before installing it is also necessary to export the `GRADLE_HOME` environment variable and include its binaries path into the `PATH` environment variable:

```
$ echo 'export GRADLE_HOME=/opt/gradle-5.4.1' >> ~/.bashrc
$ export GRADLE_HOME=/opt/gradle-5.4.1
$ echo 'export PATH=/opt/gradle-5.4.1/bin:$PATH' >> ~/.bashrc
$ export PATH=/opt/gradle-5.4.1/bin:$PATH
```

Important: Python version 3.8 or higher is recommended since some of the Python binding features are only supported in these Python versions (e.g. worker cache)

3.1.1 Build Dependencies

To build COMPSs from sources you will also need `wget`, `git` and `maven` (maven web). To install with Pip, `pip` for the target Python version is required.

3.1.2 Optional Dependencies

For the Python binding it is recommended to have `dill` (dill project), `guppy3` (guppy3 project) and `numpy` (*numpy* project <https://pypi.org/project/numpy/>) installed:

- The `dill` package increases the variety of serializable objects by Python (for example: lambda functions)
- The `guppy3` package is needed to use the `@local` decorator.
- The `numpy` package is useful to improve the serialization/deserialization performance since its internal mechanisms are used by the Python binding.

These packages can be found in PyPI and can be installed via `pip`.

Since it is possible to execute python applications using workers spawning MPI processes instead of multiprocessing, it is necessary to have `openmpi`, `openmpi-devel` and `openmpi-libs` system packages installed and `mpi4py` with pip.

3.2 Building from sources

This section describes the steps to install COMPSs from the sources.

The first step is downloading the source code from the Git repository.

```
$ git clone https://github.com/bsc-wdc/compss.git
$ cd compss
```

Then, you need to download the embedded dependencies from the git submodules.

```
$ compss> ./submodules_get.sh
```

> **Warning:** Before running the installation script in macOS distributions, some previous definitions need to be done:
> ```
> $ alias readlink=/usr/local/bin/greadlink
> $ export LIBTOOL=`which glibtool`
> $ export LIBTOOLIZE=`which glibtoolize`
> $ export JAVA_HOME=/usr/local/cellar/openjdk@8/1.8.0+282/libexec/openjdk.jdk/Contents/Home
> ```

Finally you just need to run the installation script. You have two options:

For all users

For the current user

For installing COMPSs for all users run the following command:

```
$ compss> cd builders/
$ builders> export INSTALL_DIR=/opt/COMPSs/
$ builders> sudo -E ./buildlocal ${INSTALL_DIR}
```

> **Attention:** Root access is required.

For installing COMPSs for the current user run the following commands:

```
$ compss> cd builders/
$ builders> INSTALL_DIR=$HOME/opt/COMPSs/
$ builders> ./buildlocal ${INSTALL_DIR}
```

> **Warning:** In macOS distributions, the System Integrity Protection (SIP) does not allow to modify the /System folder even with root permissions. This means the installation building from sources can only be installed for the current user.

Tip: The `buildlocal` script allows to disable the installation of components. The options can be found in the command help:

```
$ compss> cd builders/
$ builders> ./buildlocal -h

  Usage: ./buildlocal [options] targetDir
    * Options:
```

```
    --help, -h                  Print this help message

    --opts                      Show available options

    --version, -v               Print COMPSs version

    --monitor, -m               Enable Monitor installation
    --no-monitor, -M            Disable Monitor installation
                                Default: true

    --bindings, -b              Enable bindings installation
    --no-bindings, -B           Disable bindings installation
                                Default: true

    --pycompss, -p              Enable PyCOMPSs installation
    --no-pycompss, -P           Disable PyCOMPSs installation
                                Default: true

    --tracing, -t               Enable tracing system installation
    --no-tracing, -T            Disable tracing system installation
                                Default: true

    --kafka, -k                 Enable Kafka module installation
    --no-kafka, -K              Disable Kafka module installation
                                Default: true

    --jacoco, -j                Enable Jacoco module installation
    --no-jacoco, -J             Disable Jacoco module installation
                                Default: true

    --dlb, -d                   Enable dlb module installation
    --no-dlb, -D                Disable dlb module installation
                                Default: true

    --cli, -c                   Enable Command Line Interface module installation
    --no-cli, -C                Disable Command Line Interface module installation
                                Default: true

    --nothing, -N               Disable all previous options
                                Default: unused

    --user-exec=<str>           Enables a specific user execution for maven compilation
                                When used the maven install is not cleaned.
                                Default: false

    --skip-tests                Disables MVN unit tests
                                Default:

  * Parameters:
    targetDir                   COMPSs installation directory
                                Default: /opt/COMPSs
```

Warning: Components Tracing, Kafka, Jacoco and DLB cannot be installed in macOS distributions. Therefore, at least options `-T -K -J -D` must be used when invoking `buildlocal`

3.2.1 Post installation

Once your COMPSs package has been installed remember to log out and back in again to end the installation process.

> **Caution:** Using Ubuntu version 18.04 or higher requires to comment the following lines in your `.bashrc` in order to have the appropriate environment after logging out and back again (which in these distributions it must be from the complete system (e.g. gnome) not only from the terminal, or restart the whole machine).
>
> ```
> # If not running interactively, don't do anything
> # case $- in          #
> #    *i*) ;;           # Comment these lines before logging out
> #      *) return;;     # from the whole gnome (or restart the machine).
> # esac                 #
> ```

In addition, COMPSs requires **ssh passwordless access**. If you need to set up your machine for the first time please take a look at *Additional Configuration* Section for a detailed description of the additional configuration.

3.3 Pip

3.3.1 Pre-requisites

In order to be able to install COMPSs and PyCOMPSs with Pip, the dependencies (excluding the COMPSs packages) mentioned in the *Dependencies* Section must be satisfied (*do not forget* to have proper `JAVA_HOME` and `GRADLE_HOME` environment variables pointing to the java JDK folder and Gradle home respectively, as well as the `gradle` binary in the `PATH` environment variable) and Python `pip`.

3.3.2 Installation

Depending on the machine, the installation command may vary. Some of the possible scenarios and their proper installation command are:

Install systemwide

Install in user local folder

Within a virtual environment

Install systemwide:

```
$ sudo -E pip install pycompss -v
```

> **Attention:** Root access is required.

It is recommended to restart the user session once the installation process has finished. Alternatively, the following command sets all the COMPSs environment in the current session.

```
$ source /etc/profile.d/compss.sh
```

Install in user home folder (.local):

```
$ pip install pycompss -v
```

It is recommended to restart the user session once the installation process has finished. Alternatively, the following command sets all the COMPSs environment.

```
$ source ~/.bashrc
```

Within a Python virtual environment:

```
(virtualenv) $ pip install pycompss -v
```

In this particular case, the installation includes the necessary variables in the activate script. So, restart the virtual environment in order to set all the COMPSs environment.

3.3.3 Post installation

If you need to set up your machine for the first time please take a look at *Additional Configuration* Section for a detailed description of the additional configuration.

3.4 Supercomputers

The COMPSs Framework can be installed in any Supercomputer by installing its packages as in a normal distribution. The packages are ready to be reallocated so the administrators can choose the right location for the COMPSs installation.

However, if the administrators are not willing to install COMPSs through the packaging system, we also provide a **COMPSs zipped file** containing a pre-build script to easily install COMPSs. Next subsections provide further information about this process.

3.4.1 Prerequisites

In order to successfully run the installation script some dependencies must be present on the target machine. Administrators must provide the correct installation and environment of the following software:

- Autotools
- BOOST
- Java 8 JRE

The following environment variables must be defined:

- JAVA_HOME
- BOOST_CPPFLAGS

The tracing system can be enhanced with:

- PAPI, which provides support for harware counters
- MPI, which speeds up the tracing merge (and enables it for huge traces)

3.4.2 Installation

To perform the COMPSs Framework installation please execute the following commands:

```
$ # Check out the last COMPSs release
$ wget http://compss.bsc.es/repo/sc/stable/COMPSs_<version>.tar.gz

$ # Unpackage COMPSs
$ tar -xvzf COMPSs_<version>.tar.gz

$ # Install COMPSs at your preferred target location
$ cd COMPSs
$ ./install [options] <targetDir> [<supercomputer.cfg>]

$ # Clean downloaded files
$ rm -r COMPSs
$ rm COMPSs_<version>.tar.gz
```

The installation script will install COMPSs inside the given `<targetDir>` folder and it will copy the `<supercomputer.cfg>` as default configuration. It also provides some options to skip the installation of optional features or bound the installation to an specific python version. You can see the available options with the following command.

```
$ ./install --help
```

Attention: If the `<targetDir>` folder already exists it will be **automatically erased**.

After completing the previous steps, administrators must ensure that the nodes have passwordless ssh access. If it is not the case, please contact the COMPSs team at support-compss@bsc.es.

The COMPSs package also provides a *compssenv* file that loads the required environment to allow users work more easily with COMPSs. Thus, after the installation process we recommend to source the `<targetDir>/compssenv` into the users *.bashrc*.

Once done, remember to log out and back in again to end the installation process.

3.4.3 Configuration

To maintain the portability between different environments, COMPSs has a pre-built structure of scripts to execute applications in Supercomputers. For this purpose, users must use the `enqueue_compss` script provided in the COMPSs installation and specify the supercomputer configuration with `--sc_cfg` flag.

When installing COMPSs for a supercomputer, system administrators must define a configuration file for the specific Supercomputer parameters. This document gives and overview about how to modify the configuration files in order to customize the enqueue_compss for a specific queue system and supercomputer. As overview, the easier way to proceed when creating a new configuration is to modify one of the configurations provided by COMPSs. System sdministrators can find configurations for **LSF**, **SLURM**, **PBS** and **SGE** as well as several examples for Supercomputer configurations in `<installation_dir>/Runtime/scripts/queues`. For instance, the configuration for the *MareNostrum IV* Supercomputer and the *Slurm* queue system, can be used as base file for new supercomputer and queue system cfgs. Sysadmins can modify these files by changing the flags, parameters, paths and default values that corresponds to your supercomputer. Once, the files have been modified, they must be copied to the queues folder to make them available to the users. The following paragraph describe more in detail the scripts and configuration files If you need help, contact support-compss@bsc.es.

3.4.3.1 COMPSs Queue structure overview

All the scripts and cfg files shown in Figure 4 are located in the `<installation_dir>/Runtime/scripts/` folder. `enqueue_compss` and `launch_compss` (**launch.sh in the figure**) are in the user subfolder and `submit.sh` and the `cfgs` are located in queues. There are two types of cfg files: the *queue system cfg* files, which are located in `queues/queue_systems`; and the *supercomputers.cfg* files, which are located in `queues/supercomputers`.
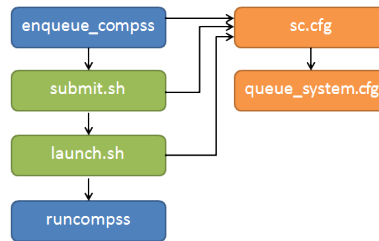
Figure 4: Structure of COMPSs queue scripts. In Blue user scripts, in Green queue scripts and in Orange system dependant scripts

3.4.3.2 Configuration Files

The cfg files contain a set of bash variables which are used by the other scripts. On the one hand, the queue system cfgs contain the variables to indicate the commands used by the system to submit and spawn processes, the commands or variables to get the allocated nodes and the directives to indicate the number of nodes, processes, etc. Below you can see an example of the most important variable definition for Slurm

```
# File: Runtime/scripts/queues/queue_systems/slurm.cfg


###############################
## SUBMISSION VARIABLES
###############################
# Variables to define the queue system directives.
# The are built as #${QUEUE_CMD} ${QARG_*}${QUEUE_SEPARATOR}value (submit.sh)
QUEUE_CMD="SBATCH"
SUBMISSION_CMD="sbatch"
SUBMISSION_PIPE="< "
SUBMISSION_HET_SEPARATOR=' : '
SUBMISSION_HET_PIPE=" "

# Variables to customize the commands know job id and allocated nodes (submit.sh)
ENV_VAR_JOB_ID="SLURM_JOB_ID"
ENV_VAR_NODE_LIST="SLURM_JOB_NODELIST"

QUEUE_SEPARATOR=""
EMPTY_WC_LIMIT=":00"

QARG_JOB_NAME="--job-name="
QARG_JOB_DEP_INLINE="false"
QARG_JOB_DEPENDENCY_OPEN="--dependency=afterany:"
QARG_JOB_DEPENDENCY_CLOSE=""

QARG_JOB_OUT="-o "
QARG_JOB_ERROR="-e "
QARG_WD="--workdir="
QARG_WALLCLOCK="-t"
```

```
QARG_NUM_NODES="-N"
QARG_NUM_PROCESSES="-n"
QNUM_PROCESSES_VALUE="\$(expr \${num_nodes} \* \${req_cpus_per_node})"
QARG_EXCLUSIVE_NODES="--exclusive"
QARG_SPAN=""

QARG_MEMORY="--mem="
QARG_QUEUE_SELECTION="-p "
QARG_NUM_SWITCHES="--gres="
QARG_GPUS_PER_NODE="--gres gpu:"
QARG_RESERVATION="--reservation="
QARG_CONSTRAINTS="--constraint="
QARG_QOS="--qos="
QARG_OVERCOMMIT="--overcommit"
QARG_CPUS_PER_TASK="-c"
QJOB_ID="%J"
QARG_PACKJOB="packjob"


################################
## LAUNCH VARIABLES
################################
# Variables to customize worker process spawn inside the job (launch_compss)
LAUNCH_CMD="srun"
LAUNCH_PARAMS="-n1 -N1 --nodelist="
LAUNCH_SEPARATOR=""
CMD_SEPARATOR=""
HOSTLIST_CMD="scontrol show hostname"
HOSTLIST_TREATMENT="| awk {' print \$1 '} | sed -e 's/\.[^\ ]*//g'"


################################
## QUEUE VARIABLES
##  - Used in interactive
##  - Substitute the %JOBID% keyword with the real job identifier dinamically
################################
QUEUE_JOB_STATUS_CMD="squeue -h -o %T --job %JOBID%"
QUEUE_JOB_RUNNING_TAG="RUNNING"
QUEUE_JOB_NODES_CMD="squeue -h -o %N --job %JOBID%"
QUEUE_JOB_CANCEL_CMD="scancel %JOBID%"
QUEUE_JOB_LIST_CMD="squeue -h -o %i"
QUEUE_JOB_NAME_CMD="squeue -h -o %j --job %JOBID%"


################################
## CONTACT VARIABLES
################################
CONTACT_CMD="ssh"
```

To adapt this script to your queue system, you just need to change the variable value to the command, argument or value required in your system. If you find that some of this variables are not available in your system, leave it empty.

On the other hand, the supercomputers cfg files contains a set of variables to indicate the queue system used by a supercomputer, paths where the shared disk is mounted, the default values that COMPSs will set in the project and resources files when they are not set by the user and flags to indicate if a functionality is available or not in a supercomputer. The following lines show examples of this variables for the *MareNostrum IV* supercomputer.

```
# File: Runtime/scripts/queues/supercomputers/mn.cfg
```

```
###############################
## STRUCTURE VARIABLES
###############################
QUEUE_SYSTEM="slurm"

###############################
## ENQUEUE_COMPSS VARIABLES
###############################
DEFAULT_EXEC_TIME=10
DEFAULT_NUM_NODES=2
DEFAULT_NUM_SWITCHES=0
MAX_NODES_SWITCH=18
MIN_NODES_REQ_SWITCH=4
DEFAULT_QUEUE=default
DEFAULT_MAX_TASKS_PER_NODE=-1
DEFAULT_CPUS_PER_NODE=48
DEFAULT_IO_EXECUTORS=0
DEFAULT_GPUS_PER_NODE=0
DEFAULT_FPGAS_PER_NODE=0
DEFAULT_WORKER_IN_MASTER_CPUS=24
DEFAULT_WORKER_IN_MASTER_MEMORY=50000
DEFAULT_MASTER_WORKING_DIR=.
DEFAULT_WORKER_WORKING_DIR=local_disk
DEFAULT_NETWORK=infiniband
DEFAULT_DEPENDENCY_JOB=None
DEFAULT_RESERVATION=disabled
DEFAULT_NODE_MEMORY=disabled
DEFAULT_JVM_MASTER=""
DEFAULT_JVM_WORKERS="-Xms16000m,-Xmx92000m,-Xmn1600m"
DEFAULT_JVM_WORKER_IN_MASTER=""
DEFAULT_QOS=default
DEFAULT_CONSTRAINTS=disabled

###############################
## Enabling/disabling passing
## requirements to queue system
###############################
DISABLE_QARG_MEMORY=true
DISABLE_QARG_CONSTRAINTS=false
DISABLE_QARG_QOS=false
DISABLE_QARG_OVERCOMMIT=true
DISABLE_QARG_CPUS_PER_TASK=false
DISABLE_QARG_NVRAM=true
HETEROGENEOUS_MULTIJOB=false

###############################
## SUBMISSION VARIABLES
###############################
MINIMUM_NUM_NODES=1
MINIMUM_CPUS_PER_NODE=1
DEFAULT_STORAGE_HOME="null"
DISABLED_STORAGE_HOME="null"

###############################
## LAUNCH VARIABLES
###############################
```

Chapter 3. Installation and Administration

```
LOCAL_DISK_PREFIX="/scratch/tmp"
REMOTE_EXECUTOR="none"  # Disable the ssh spawn at runtime
NETWORK_INFINIBAND_SUFFIX="-ib0"  # Hostname suffix to add in order to use infiniband network
NETWORK_DATA_SUFFIX="-data"  # Hostname suffix to add in order to use data network
SHARED_DISK_PREFIX="/gpfs/"
SHARED_DISK_2_PREFIX="/.statelite/tmpfs/gpfs/"
DEFAULT_NODE_MEMORY_SIZE=92
DEFAULT_NODE_STORAGE_BANDWIDTH=450
MASTER_NAME_CMD=hostname  # Command to know the mastername
ELASTICITY_BATCH=true
```

To adapt this script to your supercomputer, you just need to change the variables to commands paths or values which are set in your system. If you find that some of this values are not available in your system, leave them empty or as they are in the MareNostrum IV.

3.4.3.3 How are cfg files used in scripts?

The `submit.sh` is in charge of getting some of the arguments from `enqueue_compss`, generating the a temporal job submission script for the *queue_system* (function *create_normal_tmp_submit*) and performing the submission in the scheduler (function *submit*). The functions used in `submit.sh` are implemented in `common.sh`. If you look at the code of this script, you will see that most of the code is customized by a set of bash vars which are mainly defined in the cfg files.

For instance the submit command is customized in the following way:

```
eval ${SUBMISSION_CMD} ${SUBMISSION_PIPE}${TMP_SUBMIT_SCRIPT}
```

Where `${SUBMISSION_CMD}` and `${SUBMISSION_PIPE}` are defined in the `queue_system.cfg`. So, for the case of Slurm, at execution time it is translated to something like `sbatch < /tmp/tmp_submit_script`

The same approach is used for the queue system directives defined in the submission script or in the command to get the assigned host list.

The following lines show the examples in these cases.

```
#${QUEUE_CMD} ${QARG_JOB_NAME}${QUEUE_SEPARATOR}${job_name}
```

In the case of Slurm in MN, it generates something like `#SBATCH --job-name=COMPSs`

```
host_list=\$(${HOSTLIST_CMD} \$$${ENV_VAR_NODE_LIST}${env_var_suffix} ${HOSTLIST_TREATMENT})
```

The same approach is used in the `launch_compss` script where it is using the defined vars to customize the *project.xml* and *resources.xml* file generation and spawning the master and worker processes in the assigned resources.

At first, you should not need to modify any script. The goal of the cfg files is that sysadmins just require to modify the supercomputers cfg, and in the case that the used queue system is not in the *queue_systems*, folder it should create a new one for the new one.

If you think that some of the features of your system are not supported in the current implementation, please contact us at support-compss@bsc.es. We will discuss how it should be incorporated in the scripts.

3.4.4 Post installation

To check that COMPSs Framework has been successfully installed you may run:

```
$ # Check the COMPSs version
$ runcompss -v
COMPSs version <version>
```

For queue system executions, COMPSs provides several prebuild queue scripts than can be accessible throgh the *enqueue_compss* command. Users can check the available options by running:

```
$ enqueue_compss -h

Usage: /apps/COMPSs/2.9/Runtime/scripts/user/enqueue_compss [queue_system_options] [COMPSs_
↪options] application_name application_arguments

* Options:
  General:
    --help, -h                          Print this help message
    --heterogeneous                     Indicates submission is going to be heterogeneous
                                        Default: Disabled

  Queue system configuration:
    --sc_cfg=<name>                     SuperComputer configuration file to use. Must␣
↪exist inside queues/cfgs/

                                        Default: default


  Submission configuration:
  General submision arguments:
    --exec_time=<minutes>               Expected execution time of the application (in␣
↪minutes)
                                        Default: 10

    --job_name=<name>                   Job name
                                        Default: COMPSs
    --queue=<name>                      Queue name to submit the job. Depends on the␣
↪queue system.

                                        For example (MN3): bsc_cs | bsc_debug | debug |␣
↪interactive
                                        Default: default
    --reservation=<name>                Reservation to use when submitting the job.
                                        Default: disabled
    --constraints=<constraints>                 Constraints to pass to queue system.
                                                        Default: disabled
    --qos=<qos>                         Quality of Service to pass to the queue system.
                                        Default: default
    --cpus_per_task                     Number of cpus per task the queue system must␣
↪allocate per task.

                                        Note that this will be equal to the cpus_per_node␣
↪in a worker node and

                                        equal to the worker_in_master_cpus in a master␣
↪node respectively.
                                        Default: false
    --job_dependency=<jobID>            Postpone job execution until the job dependency␣
↪has ended.

                                        Default: None
    --storage_home=<string>             Root installation dir of the storage␣
↪implementation

                                        Default: null
    --storage_props=<string>            Absolute path of the storage properties file
```

(continues on next page)

```
                                        Mandatory if storage_home is defined
  Normal submission arguments:
    --num_nodes=<int>                   Number of nodes to use
                                        Default: 2

    --num_switches=<int>                Maximum number of different switches. Select 0␣
↪for no restrictions.

                                        Maximum nodes per switch: 18
                                        Only available for at least 4 nodes.
                                        Default: 0
    --agents=<string>                   Hierarchy of agents for the deployment. Accepted␣
↪values: plain|tree

                                        Default: tree

    --agents                            Deploys the runtime as agents instead of the␣
↪classic Master-Worker deployment.

                                        Default: disabled
  Heterogeneous submission arguments:
    --type_cfg=<file_location>          Location of the file with the descriptions of␣
↪node type requests

                                        File should follow the following format:
                                        type_X(){
                                          cpus_per_node=24
                                          node_memory=96
                                          ...
                                        }
                                        type_Y(){
                                          ...
                                        }
    --master=<master_node_type>         Node type for the master
                                        (Node type descriptions are provided in the --
↪type_cfg flag)
    --workers=type_X:nodes,type_Y:nodes Node type and number of nodes per type for the␣
↪workers

                                        (Node type descriptions are provided in the --
↪type_cfg flag)
  Launch configuration:
    --cpus_per_node=<int>               Available CPU computing units on each node
                                        Default: 48
    --gpus_per_node=<int>               Available GPU computing units on each node
                                        Default: 0
    --fpgas_per_node=<int>              Available FPGA computing units on each node
                                        Default: 0
    --io_executors=<int>                Number of IO executors on each node
                                        Default: 0
    --fpga_reprogram="<string>          Specify the full command that needs to be␣
↪executed to reprogram the FPGA with

                                        the desired bitstream. The location must be an␣
↪absolute path.

                                        Default:
    --max_tasks_per_node=<int>          Maximum number of simultaneous tasks running on a␣
↪node

                                        Default: -1
    --node_memory=<MB>                  Maximum node memory: disabled | <int> (MB)
                                        Default: disabled
    --node_storage_bandwidth=<MB>       Maximum node storage bandwidth: <int> (MB)
                                        Default: 450
```

```
    --network=<name>                      Communication network for transfers: default |␣
→ethernet | infiniband | data.

                                          Default: infiniband


    --prolog="<string>"                   Task to execute before launching COMPSs (Notice␣
→the quotes)

                                          If the task has arguments split them by ","␣
→rather than spaces.

                                          This argument can appear multiple times for more␣
→than one prolog action
                                          Default: Empty
    --epilog="<string>"                   Task to execute after executing the COMPSs␣
→application (Notice the quotes)

                                          If the task has arguments split them by ","␣
→rather than spaces.

                                          This argument can appear multiple times for more␣
→than one epilog action
                                          Default: Empty


    --master_working_dir=<path>           Working directory of the application
                                          Default: .
    --worker_working_dir=<name | path>    Worker directory. Use: local_disk | shared_disk |
→<path>

                                          Default: local_disk


    --worker_in_master_cpus=<int>         Maximum number of CPU computing units that the␣
→master node can run as worker. Cannot exceed cpus_per_node.
                                          Default: 24
    --worker_in_master_memory=<int> MB    Maximum memory in master node assigned to the␣
→worker. Cannot exceed the node_memory.

                                          Mandatory if worker_in_master_cpus is specified.
                                          Default: 50000
    --worker_port_range=<min>,<max>       Port range used by the NIO adaptor at the worker␣
→side

                                                          Default: 43001,43005
    --jvm_worker_in_master_opts="<string>" Extra options for the JVM of the COMPSs Worker in␣
→the Master Node.

                                          Each option separed by "," and without blank␣
→spaces (Notice the quotes)
                                          Default:
    --container_image=<path>              Runs the application by means of a container␣
→engine image

                                          Default: Empty
    --container_compss_path=<path>        Path where compss is installed in the container␣
→image

                                          Default: /opt/COMPSs
    --container_opts="<string>"           Options to pass to the container engine
                                          Default: empty
    --elasticity=<max_extra_nodes>        Activate elasticity specifiying the maximum extra␣
→nodes (ONLY AVAILABLE FORM SLURM CLUSTERS WITH NIO ADAPTOR)
                                          Default: 0
    --automatic_scaling=<bool>            Enable or disable the runtime automatic scaling␣
→(for elasticity)

                                          Default: true
    --jupyter_notebook=<path>,            Swap the COMPSs master initialization with␣
→jupyter notebook from the specified path.
```

```
   --jupyter_notebook                     Default: false
   --ipython                              Swap the COMPSs master initialization with␣
↪ipython.
                                          Default: empty


 Runcompss configuration:


 Tools enablers:
   --graph=<bool>, --graph, -g            Generation of the complete graph (true/false)
                                          When no value is provided it is set to true
                                          Default: false
   --tracing=<level>, --tracing, -t       Set generation of traces and/or tracing level ( [␣
↪true | basic ] | advanced | scorep | arm-map | arm-ddt | false)
                                          True and basic levels will produce the same␣
↪traces.
                                          When no value is provided it is set to 1
                                          Default: 0
   --monitoring=<int>, --monitoring, -m   Period between monitoring samples (milliseconds)
                                          When no value is provided it is set to 2000
                                          Default: 0
   --external_debugger=<int>,
   --external_debugger                    Enables external debugger connection on the␣
↪specified port (or 9999 if empty)
                                          Default: false
   --jmx_port=<int>                       Enable JVM profiling on specified port

 Runtime configuration options:
   --task_execution=<compss|storage>      Task execution under COMPSs or Storage.
                                          Default: compss
   --storage_impl=<string>                Path to an storage implementation. Shortcut to␣
↪setting pypath and classpath. See Runtime/storage in your installation folder.
   --storage_conf=<path>                  Path to the storage configuration file
                                          Default: null
   --project=<path>                       Path to the project XML file
                                          Default: /apps/COMPSs/2.9//Runtime/configuration/
↪xml/projects/default_project.xml
   --resources=<path>                     Path to the resources XML file
                                          Default: /apps/COMPSs/2.9//Runtime/configuration/
↪xml/resources/default_resources.xml
   --lang=<name>                          Language of the application (java/c/python)
                                          Default: Inferred is possible. Otherwise: java
   --summary                              Displays a task execution summary at the end of␣
↪the application execution
                                          Default: false
   --log_level=<level>, --debug, -d       Set the debug level: off | info | api | debug |␣
↪trace
                                          Warning: Off level compiles with -O2 option␣
↪disabling asserts and __debug__
                                          Default: off


 Advanced options:
   --extrae_config_file=<path>            Sets a custom extrae config file. Must be in a␣
↪shared disk between all COMPSs workers.
                                          Default: null
```

```
    --trace_label=<string>                      Add a label in the generated trace file. Only␣
→used in the case of tracing is activated.
                                                Default: None
    --comm=<ClassName>                          Class that implements the adaptor for␣
→communications

                                                Supported adaptors:
                                                        ├── es.bsc.compss.nio.master.NIOAdaptor
                                                        └── es.bsc.compss.gat.master.GATAdaptor
                                                Default: es.bsc.compss.nio.master.NIOAdaptor
    --conn=<className>                          Class that implements the runtime connector for␣
→the cloud

                                                Supported connectors:
                                                        ├── es.bsc.compss.connectors.
→DefaultSSHConnector

                                                        └── es.bsc.compss.connectors.
→DefaultNoSSHConnector
                                                Default: es.bsc.compss.connectors.
→DefaultSSHConnector
    --streaming=<type>                          Enable the streaming mode for the given type.
                                                Supported types: FILES, OBJECTS, PSCOS, ALL, NONE
                                                Default: NONE
    --streaming_master_name=<str>               Use an specific streaming master node name.
                                                Default: null
    --streaming_master_port=<int>               Use an specific port for the streaming master.
                                                Default: null
    --scheduler=<className>                      Class that implements the Scheduler for COMPSs
                                                Supported schedulers:
                                                        ├── es.bsc.compss.scheduler.
→fifodatalocation.FIFODataLoctionScheduler

                                                        ├── es.bsc.compss.scheduler.fifonew.
→FIFOScheduler

                                                        ├── es.bsc.compss.scheduler.fifodatanew.
→FIFODataScheduler

                                                        ├── es.bsc.compss.scheduler.lifonew.
→LIFOScheduler

                                                        ├── es.bsc.compss.components.impl.
→TaskScheduler

                                                        └── es.bsc.compss.scheduler.loadbalancing.
→LoadBalancingScheduler
                                                Default: es.bsc.compss.scheduler.loadbalancing.
→LoadBalancingScheduler
    --scheduler_config_file=<path>              Path to the file which contains the scheduler␣
→configuration.

                                                Default: Empty
    --library_path=<path>                       Non-standard directories to search for libraries␣
→(e.g. Java JVM library, Python library, C binding library)
                                                Default: Working Directory
    --classpath=<path>                          Path for the application classes / modules
                                                Default: Working Directory
    --appdir=<path>                             Path for the application class folder.
                                                Default: /home/group/user
    --pythonpath=<path>                         Additional folders or paths to add to the␣
→PYTHONPATH
                                                Default: /home/group/user
    --base_log_dir=<path>                       Base directory to store COMPSs log files (a .
→COMPSs/ folder will be created inside this location)
```

```
                                        Default: User home
    --specific_log_dir=<path>           Use a specific directory to store COMPSs log␣
↪files (no sandbox is created)

                                        Warning: Overwrites --base_log_dir option
                                        Default: Disabled
    --uuid=<int>                        Preset an application UUID
                                        Default: Automatic random generation
    --master_name=<string>              Hostname of the node to run the COMPSs master
                                        Default:
    --master_port=<int>                 Port to run the COMPSs master communications.
                                        Only for NIO adaptor
                                        Default: [43000,44000]
    --jvm_master_opts="<string>"        Extra options for the COMPSs Master JVM. Each␣
↪option separed by "," and without blank spaces (Notice the quotes)
                                        Default:
    --jvm_workers_opts="<string>"       Extra options for the COMPSs Workers JVMs. Each␣
↪option separed by "," and without blank spaces (Notice the quotes)
                                        Default: -Xms1024m,-Xmx1024m,-Xmn400m
    --cpu_affinity="<string>"           Sets the CPU affinity for the workers
                                        Supported options: disabled, automatic, user␣
↪defined map of the form "0-8/9,10,11/12-14,15,16"
                                        Default: automatic
    --gpu_affinity="<string>"           Sets the GPU affinity for the workers
                                        Supported options: disabled, automatic, user␣
↪defined map of the form "0-8/9,10,11/12-14,15,16"
                                        Default: automatic
    --fpga_affinity="<string>"          Sets the FPGA affinity for the workers
                                        Supported options: disabled, automatic, user␣
↪defined map of the form "0-8/9,10,11/12-14,15,16"
                                        Default: automatic
    --fpga_reprogram="<string>"         Specify the full command that needs to be␣
↪executed to reprogram the FPGA with the desired bitstream. The location must be an absolute␣
↪path.

                                        Default:
    --io_executors=<int>                IO Executors per worker
                                        Default: 0
    --task_count=<int>                  Only for C/Python Bindings. Maximum number of␣
↪different functions/methods, invoked from the application, that have been selected as tasks
                                        Default: 50
    --input_profile=<path>              Path to the file which stores the input␣
↪application profile
                                        Default: Empty
    --output_profile=<path>             Path to the file to store the application profile␣
↪at the end of the execution
                                        Default: Empty
    --PyObject_serialize=<bool>         Only for Python Binding. Enable the object␣
↪serialization to string when possible (true/false).
                                        Default: false
    --persistent_worker_c=<bool>        Only for C Binding. Enable the persistent worker␣
↪in c (true/false).
                                        Default: false
    --enable_external_adaptation=<bool> Enable external adaptation. This option will␣
↪disable the Resource Optimizer.
                                        Default: false
    --gen_coredump                      Enable master coredump generation
                                        Default: false
```

```
    --python_interpreter=<string>          Python interpreter to use (python/python2/
→python3).
                                           Default: python Version: 2
    --python_propagate_virtual_environment=<true>  Propagate the master virtual environment␣
→to the workers (true/false).
                                              Default: true
    --python_mpi_worker=<false>            Use MPI to run the python worker instead of␣
→multiprocessing. (true/false).
                                           Default: false
    --python_memory_profile                Generate a memory profile of the master.
                                           Default: false

* Application name:
    For Java applications:   Fully qualified name of the application
    For C applications:      Path to the master binary
    For Python applications: Path to the .py file containing the main program

* Application arguments:
    Command line arguments to pass to the application. Can be empty.
```

If none of the pre-build queue configurations adapts to your infrastructure (lsf, pbs, slurm, etc.) please contact the COMPSs team at support-compss@bsc.es to find out a solution.

If you are willing to test the COMPSs Framework installation you can run any of the applications available at our application repository https://github.com/bsc-wdc/apps. We suggest to run the java simple application following the steps listed inside its *README* file.

For further information about either the installation or the usage please check the *README* file inside the COMPSs package.

3.5 Additional Configuration

3.5.1 Configure SSH passwordless

By default, COMPSs uses SSH libraries for communication between nodes. Consequently, after COMPSs is installed on a set of machines, the SSH keys must be configured on those machines so that COMPSs can establish passwordless connections between them. This requires to install the OpenSSH package (if not present already) and follow these steps **on each machine**:

1. Generate an SSH key pair

```
$ ssh-keygen -t rsa
```

2. Distribute the public key to all the other machines and configure it as authorized

```
$ # For every other available machine (MACHINE):
$ scp ~/.ssh/id_rsa.pub MACHINE:./myRSA.pub
$ ssh MACHINE "cat ./myRSA.pub >> ~/.ssh/authorized_keys; rm ./myRSA.pub"
```

3. Check that passwordless SSH connections are working fine

```
$ # For every other available machine (MACHINE):
$ ssh MACHINE
```

For example, considering the cluster shown in Figure 5, users will have to execute the following commands to grant free ssh access between any pair of machines:

```
me@localhost:~$ ssh-keygen -t id_rsa
# Granting access localhost -> m1.bsc.es
me@localhost:~$ scp ~/.ssh/id_rsa.pub user_m1@m1.bsc.es:./me_localhost.pub
me@localhost:~$ ssh user_m1@m1.bsc.es "cat ./me_localhost.pub >> ~/.ssh/authorized_keys; rm ./
→me_localhost.pub"
# Granting access localhost -> m2.bsc.es
me@localhost:~$ scp ~/.ssh/id_rsa.pub user_m2@m2.bsc.es:./me_localhost.pub
me@localhost:~$ ssh user_m2@m2.bsc.es "cat ./me_localhost.pub >> ~/.ssh/authorized_keys; rm ./
→me_localhost.pub"

me@localhost:~$ ssh user_m1@m1.bsc.es
user_m1@m1.bsc.es:~> ssh-keygen -t id_rsa
user_m1@m1.bsc.es:~> exit
# Granting access m1.bsc.es -> localhost
me@localhost:~$ scp user_m1@m1.bsc.es:~/.ssh/id_rsa.pub ~/userm1_m1.pub
me@localhost:~$ cat ~/userm1_m1.pub >> ~/.ssh/authorized_keys
# Granting access m1.bsc.es -> m2.bsc.es
me@localhost:~$ scp ~/userm1_m1.pub user_m2@m2.bsc.es:~/userm1_m1.pub
me@localhost:~$ ssh user_m2@m2.bsc.es "cat ./userm1_m1.pub >> ~/.ssh/authorized_keys; rm ./
→userm1_m1.pub"
me@localhost:~$ rm ~/userm1_m1.pub

me@localhost:~$ ssh user_m2@m2.bsc.es
user_m2@m2.bsc.es:~> ssh-keygen -t id_rsa
user_m2@m2.bsc.es:~> exit
# Granting access m2.bsc.es -> localhost
me@localhost:~$ scp user_m2@m1.bsc.es:~/.ssh/id_rsa.pub ~/userm2_m2.pub
me@localhost:~$ cat ~/userm2_m2.pub >> ~/.ssh/authorized_keys
# Granting access m2.bsc.es -> m1.bsc.es
me@localhost:~$ scp ~/userm2_m2.pub user_m1@m1.bsc.es:~/userm2_m2.pub
me@localhost:~$ ssh user_m1@m1.bsc.es "cat ./userm2_m2.pub >> ~/.ssh/authorized_keys; rm ./
→userm2_m2.pub"
me@localhost:~$ rm ~/userm2_m2.pub
```
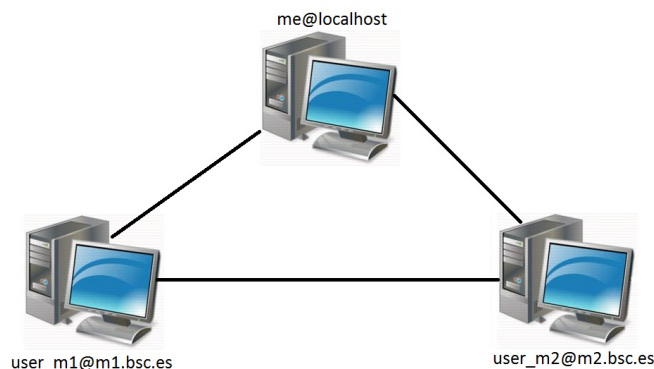

Figure 5: Cluster example

3.5.2 Configure the COMPSs Cloud Connectors

This section provides information about the additional configuration needed for some Cloud Connectors.

3.5.2.1 OCCI (Open Cloud Computing Interface) connector

In order to execute a COMPSs application using cloud resources, the rOCCI (Ruby OCCI) connector[1] has to be configured properly. The connector uses the rOCCI CLI client (upper versions from 4.2.5) which has to be installed in the node where the COMPSs main application runs. The client can be installed following the instructions detailed at http://appdb.egi.eu/store/software/rocci.cli

3.6 Configuration Files

The COMPSs runtime has two configuration files: `resources.xml` and `project.xml` . These files contain information about the execution environment and are completely independent from the application.

For each execution users can load the default configuration files or specify their custom configurations by using, respectively, the `--resources=<absolute_path_to_resources.xml>` and the `--project=<absolute_path_to_project.xml>` in the `runcompss` command. The default files are located in the `/opt/COMPSs/Runtime/configuration/xml/` path.

Next sections describe in detail the `resources.xml` and the `project.xml` files, explaining the available options.

3.6.1 Resources file

The `resources` file provides information about all the available resources that can be used for an execution. This file should normally be managed by the system administrators. Its full definition schema can be found at `/opt/COMPSs/Runtime/configuration/xml/resources/resource_schema.xsd`.

For the sake of clarity, users can also check the SVG schema located at `/opt/COMPSs/Runtime/configuration/xml/resources/resource_schema.svg`.

This file contains one entry per available resource defining its name and its capabilities. Administrators can define several resource capabilities (see example in the next listing) but we would like to underline the importance of **ComputingUnits**. This capability represents the number of available cores in the described resource and it is used to schedule the correct number of tasks. Thus, it becomes essential to define it accordingly to the number of cores in the physical resource.

```
compss@bsc:~$ cat /opt/COMPSs/Runtime/configuration/xml/resources/default_resources.xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResourcesList>
    <ComputeNode Name="localhost">
        <Processor Name="P1">
            <ComputingUnits>4</ComputingUnits>
            <Architecture>amd64</Architecture>
            <Speed>3.0</Speed>
        </Processor>
        <Processor Name="P2">
            <ComputingUnits>2</ComputingUnits>
        </Processor>
        <Adaptors>
            <Adaptor Name="es.bsc.compss.nio.master.NIOAdaptor">
                <SubmissionSystem>
                    <Interactive/>
                </SubmissionSystem>
```

(continues on next page)

---

[1] https://appdb.egi.eu/store/software/rocci.cli

```xml
                <Ports>
                    <MinPort>43001</MinPort>
                    <MaxPort>43002</MaxPort>
                </Ports>
            </Adaptor>
        </Adaptors>
        <Memory>
            <Size>16</Size>
        </Memory>
        <Storage>
            <Size>200.0</Size>
        </Storage>
        <OperatingSystem>
            <Type>Linux</Type>
            <Distribution>OpenSUSE</Distribution>
        </OperatingSystem>
        <Software>
            <Application>Java</Application>
            <Application>Python</Application>
        </Software>
    </ComputeNode>
</ResourcesList>
```

3.6.2 Project file

The project file provides information about the resources used in a specific execution. Consequently, the resources that appear in this file are a subset of the resources described in the `resources.xml` file. This file, that contains one entry per worker, is usually edited by the users and changes from execution to execution. Its full definition schema can be found at `/opt/COMPSs/Runtime/configuration/xml/projects/project_schema.xsd`.

For the sake of clarity, users can also check the SVG schema located at `/opt/COMPSs/Runtime/configuration/xml/projects/project_schema.xsd`.

We emphasize the importance of correctly defining the following entries:

installDir Indicates the path of the COMPSs installation **inside the resource** (not necessarily the same than in the local machine).

User Indicates the username used to connect via ssh to the resource. This user **must** have passwordless access to the resource (see *Configure SSH passwordless* Section). If left empty COMPSs will automatically try to access the resource with the **same username as the one that lauches the COMPSs main application**.

LimitOfTasks The maximum number of tasks that can be simultaneously scheduled to a resource. Considering that a task can use more than one core of a node, this value must be lower or equal to the number of available cores in the resource.

```
compss@bsc:~$ cat /opt/COMPSs/Runtime/configuration/xml/projects/default_project.xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Project>
    <!-- Description for Master Node -->
    <MasterNode></MasterNode>

    <!--Description for a physical node-->
    <ComputeNode Name="localhost">
        <InstallDir>/opt/COMPSs/</InstallDir>
        <WorkingDir>/tmp/Worker/</WorkingDir>
        <Application>
            <AppDir>/home/user/apps/</AppDir>
```

```
            <LibraryPath>/usr/lib/</LibraryPath>
            <Classpath>/home/user/apps/jar/example.jar</Classpath>
            <Pythonpath>/home/user/apps/</Pythonpath>
        </Application>
        <LimitOfTasks>4</LimitOfTasks>
        <Adaptors>
            <Adaptor Name="es.bsc.compss.nio.master.NIOAdaptor">
                <SubmissionSystem>
                    <Interactive/>
                </SubmissionSystem>
                <Ports>
                    <MinPort>43001</MinPort>
                    <MaxPort>43002</MaxPort>
                </Ports>
                <User>user</User>
            </Adaptor>
        </Adaptors>
    </ComputeNode>
</Project>
```

3.6.3 Configuration examples

In the next subsections we provide specific information about the services, shared disks, cluster and cloud configurations and several `project.xml` and `resources.xml` examples.

3.6.3.1 Parallel execution on one single process configuration

The most basic execution that COMPSs supports is using no remote workers and running all the tasks internally within the same process that hosts the application execution. To enable the parallel execution of the application, the user needs to set up the runtime and provide a description of the resources available on the node. For that purpose, the user describes within the `<MasterNode>` tag of the `project.xml` file the resources in the same way it describes other nodes' resources on the using the `resources.xml` file. Since there is no inter-process communication, adaptors description is not allowed. In the following example, the master will manage the execution of tasks on the MainProcessor CPU of the local node - a quad-core amd64 processor at 3.0GHz - and use up to 16 GB of RAM memory and 200 GB of storage.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Project>
    <MasterNode>
        <Processor Name="MainProcessor">
            <ComputingUnits>4</ComputingUnits>
            <Architecture>amd64</Architecture>
            <Speed>3.0</Speed>
        </Processor>
        <Memory>
            <Size>16</Size>
        </Memory>
        <Storage>
            <Size>200.0</Size>
        </Storage>
    </MasterNode>
</Project>
```

If no other nodes are available, the list of resources on the `resources.xml` file is empty as shown in the following file sample. Otherwise, the user can define other nodes besides the master node as described in the following

section, and the runtime system will orchestrate the task execution on both the local process and on the configured remote nodes.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResourcesList>
</ResourcesList>
```

3.6.3.2 Cluster and grid configuration (static resources)

In order to use external resources to execute the applications, the following steps have to be followed:

1. Install the *COMPSs Worker* package (or the full *COMPSs Framework* package) on all the new resources.
2. Set SSH passwordless access to the rest of the remote resources.
3. Create the *WorkingDir* directory in the resource (remember this path because it is needed for the `project.xml` configuration).
4. Manually deploy the application on each node.

The `resources.xml` and the `project.xml` files must be configured accordingly. Here we provide examples about configuration files for Grid and Cluster environments.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResourcesList>
    <ComputeNode Name="hostname1.domain.es">
        <Processor Name="MainProcessor">
            <ComputingUnits>4</ComputingUnits>
        </Processor>
        <Adaptors>
            <Adaptor Name="es.bsc.compss.nio.master.NIOAdaptor">
                <SubmissionSystem>
                    <Interactive/>
                </SubmissionSystem>
                <Ports>
                    <MinPort>43001</MinPort>
                    <MaxPort>43002</MaxPort>
                </Ports>
            </Adaptor>
            <Adaptor Name="es.bsc.compss.gat.master.GATAdaptor">
                <SubmissionSystem>
                    <Batch>
                        <Queue>sequential</Queue>
                    </Batch>
                    <Interactive/>
                </SubmissionSystem>
                <BrokerAdaptor>sshtrilead</BrokerAdaptor>
            </Adaptor>
        </Adaptors>
    </ComputeNode>

    <ComputeNode Name="hostname2.domain.es">
        ...
    </ComputeNode>
</ResourcesList>
```

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Project>
    <MasterNode/>
    <ComputeNode Name="hostname1.domain.es">
```

```
        <InstallDir>/opt/COMPSs/</InstallDir>
        <WorkingDir>/tmp/COMPSsWorker1/</WorkingDir>
        <User>user</User>
        <LimitOfTasks>2</LimitOfTasks>
    </ComputeNode>
    <ComputeNode Name="hostname2.domain.es">

        ...

    </ComputeNode>
</Project>
```

3.6.3.3 Shared Disks configuration example

Configuring shared disks might reduce the amount of data transfers improving the application performance. To configure a shared disk the users must:

1. Define the shared disk and its capabilities
2. Add the shared disk and its mountpoint to each worker
3. Add the shared disk and its mountpoint to the master node

Next example illustrates steps 1 and 2. The `<SharedDisk>` tag adds a new shared disk named `sharedDisk0` and the `<AttachedDisk>` tag adds the mountpoint of a named shared disk to a specific worker.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResourcesList>
    <SharedDisk Name="sharedDisk0">
        <Storage>
            <Size>100.0</Size>
            <Type>Persistent</Type>
        </Storage>
    </SharedDisk>


    <ComputeNode Name="localhost">
      ...
      <SharedDisks>
        <AttachedDisk Name="sharedDisk0">
          <MountPoint>/tmp/SharedDisk/</MountPoint>
        </AttachedDisk>
      </SharedDisks>
    </ComputeNode>
</ResourcesList>
```

On the other side, to add the shared disk to the **master node**, the users must edit the `project.xml` file. Next example shows how to attach the previous `sharedDisk0` to the master node:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Project>
    <MasterNode>
        <SharedDisks>
            <AttachedDisk Name="sharedDisk0">
                <MountPoint>/home/sharedDisk/</MountPoint>
            </AttachedDisk>
        </SharedDisks>
    </MasterNode>


    <ComputeNode Name="localhost">
      ...
```

```
        </ComputeNode>
</Project>
```

Notice that the `resources.xml` file can have multiple `SharedDisk` definitions and that the `SharedDisks` tag (either in the `resources.xml` or in the `project.xml` files) can have multiple `AttachedDisk` childrens to mount several shared disks on the same worker or master.

3.6.3.4 Cloud configuration (dynamic resources)

In order to use cloud resources to execute the applications, the following steps have to be followed:

1. Prepare cloud images with the *COMPSs Worker* package or the full *COMPSs Framework* package installed.
2. The application will be deployed automatically during execution but the users need to set up the configuration files to specify the application files that must be deployed.

The COMPSs runtime communicates with a cloud manager by means of connectors. Each connector implements the interaction of the runtime with a given provider's API, supporting four basic operations: ask for the price of a certain VM in the provider, get the time needed to create a VM, create a new VM and terminate a VM. This design allows connectors to abstract the runtime from the particular API of each provider and facilitates the addition of new connectors for other providers.

The `resources.xml` file must contain one or more `<CloudProvider>` tags that include the information about a particular provider, associated to a given connector. The tag **must** have an attribute **Name** to uniquely identify the provider. Next example summarizes the information to be specified by the user inside this tag.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResourcesList>
    <CloudProvider Name="PROVIDER_NAME">
        <Endpoint>
            <Server>https://PROVIDER_URL</Server>
            <ConnectorJar>CONNECTOR_JAR</ConnectorJar>
            <ConnectorClass>CONNECTOR_CLASS</ConnectorClass>
        </Endpoint>
        <Images>
            <Image Name="Image1">
                <Adaptors>
                    <Adaptor Name="es.bsc.compss.nio.master.NIOAdaptor">
                        <SubmissionSystem>
                            <Interactive/>
                        </SubmissionSystem>
                        <Ports>
                            <MinPort>43001</MinPort>
                            <MaxPort>43010</MaxPort>
                        </Ports>
                    </Adaptor>
                </Adaptors>
                <OperatingSystem>
                    <Type>Linux</Type>
                </OperatingSystem>
                <Software>
                    <Application>Java</Application>
                </Software>
                <Price>
                    <TimeUnit>100</TimeUnit>
                    <PricePerUnit>36.0</PricePerUnit>
                </Price>
            </Image>
```

```
                <Image Name="Image2">
                    <Adaptors>
                        <Adaptor Name="es.bsc.compss.nio.master.NIOAdaptor">
                            <SubmissionSystem>
                                <Interactive/>
                            </SubmissionSystem>
                            <Ports>
                                <MinPort>43001</MinPort>
                                <MaxPort>43010</MaxPort>
                            </Ports>
                        </Adaptor>
                    </Adaptors>
                </Image>
            </Images>

            <InstanceTypes>
                <InstanceType Name="Instance1">
                    <Processor Name="P1">
                        <ComputingUnits>4</ComputingUnits>
                        <Architecture>amd64</Architecture>
                        <Speed>3.0</Speed>
                    </Processor>
                    <Processor Name="P2">
                        <ComputingUnits>4</ComputingUnits>
                    </Processor>
                    <Memory>
                        <Size>1000.0</Size>
                    </Memory>
                    <Storage>
                        <Size>2000.0</Size>
                    </Storage>
                </InstanceType>
                <InstanceType Name="Instance2">
                    <Processor Name="P1">
                        <ComputingUnits>4</ComputingUnits>
                    </Processor>
                </InstanceType>
            </InstanceTypes>
        </CloudProvider>
</ResourcesList>
```

The `project.xml` complements the information about a provider listed in the `resources.xml` file. This file can contain a `<Cloud>` tag where to specify a list of providers, each with a `<CloudProvider>` tag, whose **name** attribute must match one of the providers in the `resources.xml` file. Thus, the `project.xml` file **must** contain a subset of the providers specified in the `resources.xml` file. Next example summarizes the information to be specified by the user inside this `<Cloud>` tag.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Project>
    <Cloud>
        <InitialVMs>1</InitialVMs>
        <MinimumVMs>1</MinimumVMs>
        <MaximumVMs>4</MaximumVMs>
        <CloudProvider Name="PROVIDER_NAME">
            <LimitOfVMs>4</LimitOfVMs>
            <Properties>
```

```xml
                <Property Context="C1">
                    <Name>P1</Name>
                    <Value>V1</Value>
                </Property>
                <Property>
                    <Name>P2</Name>
                    <Value>V2</Value>
                </Property>
            </Properties>

            <Images>
                <Image Name="Image1">
                    <InstallDir>/opt/COMPSs/</InstallDir>
                    <WorkingDir>/tmp/Worker/</WorkingDir>
                    <User>user</User>
                    <Application>
                        <Pythonpath>/home/user/apps/</Pythonpath>
                    </Application>
                    <LimitOfTasks>2</LimitOfTasks>
                    <Package>
                        <Source>/home/user/apps/</Source>
                        <Target>/tmp/Worker/</Target>
                        <IncludedSoftware>
                            <Application>Java</Application>
                            <Application>Python</Application>
                        </IncludedSoftware>
                    </Package>
                    <Package>
                        <Source>/home/user/apps/</Source>
                        <Target>/tmp/Worker/</Target>
                    </Package>
                    <Adaptors>
                        <Adaptor Name="es.bsc.compss.nio.master.NIOAdaptor">
                            <SubmissionSystem>
                                <Interactive/>
                            </SubmissionSystem>
                            <Ports>
                                <MinPort>43001</MinPort>
                                <MaxPort>43010</MaxPort>
                            </Ports>
                        </Adaptor>
                    </Adaptors>
                </Image>
                <Image Name="Image2">
                    <InstallDir>/opt/COMPSs/</InstallDir>
                    <WorkingDir>/tmp/Worker/</WorkingDir>
                </Image>
            </Images>
            <InstanceTypes>
                <InstanceType Name="Instance1"/>
                <InstanceType Name="Instance2"/>
            </InstanceTypes>
        </CloudProvider>

        <CloudProvider Name="PROVIDER_NAME2">
            ...
```

<div align="right">(continued from previous page)</div>

```
        </CloudProvider>
    </Cloud>
</Project>
```

For any connector the Runtime is capable to handle the next list of properties:

Table 2: Connector supported properties in the `project.xml` file

Name	Description
provider-user	Username to login in the provider
provider-user-credential	Credential to login in the provider
time-slot	Time slot
estimated-creation-time	Estimated VM creation time
max-vm-creation-time	Maximum VM creation time

Additionally, for any connector based on SSH, the Runtime automatically handles the next list of properties:

Table 3: Properties supported by any SSH based connector in the `project.xml` file

Name	Description
vm-user	User to login in the VM
vm-password	Password to login in the VM
vm-keypair-name	Name of the Keypair to login in the VM
vm-keypair-location	Location (in the master) of the Keypair to login in the VM

Finally, the next sections provide a more accurate description of each of the currently available connector and its specific properties.

Cloud connectors: rOCCI

The connector uses the rOCCI binary client[1] (version newer or equal than 4.2.5) which has to be installed in the node where the COMPSs main application is executed.

This connector needs additional files providing details about the resource templates available on each provider. This file is located under `<COMPSs_INSTALL_DIR>/configuration/xml/templates` path. Additionally, the user must define the virtual images flavors and instance types offered by each provider; thus, when the runtime decides the creation of a VM, the connector selects the appropriate image and resource template according to the requirements (in terms of CPU, memory, disk, etc) by invoking the rOCCI client through Mixins (heritable classes that override and extend the base templates).

Table 4 contains the rOCCI specific properties that must be defined under the `Provider` tag in the `project.xml` file and Table 5 contains the specific properties that must be defined under the `Instance` tag.

---

[1] https://appdb.egi.eu/store/software/rocci.cli

Table 4: rOCCI extensions in the `project.xml` file

Name	Description
auth	Authentication method, x509 only supported
user-cred	Path of the VOMS proxy
ca-path	Path to CA certificates directory
ca-file	Specific CA filename
owner	Optional. Used by the PMES Job-Manager
jobname	Optional. Used by the PMES Job-Manager
timeout	Maximum command time
username	Username to connect to the back-end cloud provider
password	Password to connect to the back-end cloud provider
voms	Enable VOMS authentication
media-type	Media type
resource	Resource type
attributes	Extra resource attributes for the back-end cloud provider
context	Extra context for the back-end cloud provider
action	Extra actions for the back-end cloud provider
mixin	Mixin definition
link	Link
trigger-action	Adds a trigger
log-to	Redirect command logs
skip-ca-check	Skips CA checks
filter	Filters command output
dump-model	Dumps the internal model
debug	Enables the debug mode on the connector commands
verbose	Enables the verbose mode on the connector commands

Table 5: Configuration of the `<resources>.xml` templates file

Instance	Multiple entries of resource templates.
Type	Name of the resource template. It has to be the same name than in the previous files
CPU	Number of cores
Memory	Size in GB of the available RAM
Disk	Size in GB of the storage
Price	Cost per hour of the instance

Cloud connectors: JClouds

The JClouds connector is based on the JClouds API version *1.9.1*. Table Table 6 shows the extra available options under the *Properties* tag that are used by this connector.

Table 6: JClouds extensions in the `<project>.xml` file

Instance	Description
provider	Back-end provider to use with JClouds (i.e. aws-ec2)

Cloud connectors: Docker

This connector uses a Java API client from https://github.com/docker-java/docker-java, version *3.0.3*. It has not additional options. Make sure that the image/s you want to load are pulled before running COMPSs with `docker pull IMAGE`. Otherwise, the connectorn will throw an exception.

Cloud connectors: Mesos

The connector uses the v0 Java API for Mesos which has to be installed in the node where the COMPSs main application is executed. This connector creates a Mesos framework and it uses Docker images to deploy workers, each one with an own IP address.

By default it does not use authentication and the timeout timers are set to 3 minutes (180.000 milliseconds). The list of **optional** properties available from connector is shown in Table 7.

Table 7: Mesos connector options in the `<project>.xml` file

Instance	Description
mesos-framework-name	Framework name to show in Mesos.
mesos-woker-name	Worker names to show in Mesos.
mesos-framework-hostname	Framework hostname to show in Mesos.
mesos-checkpoint	Checkpoint for the framework.
mesos-authenticate	Uses authentication? (`true`/`false`)
mesos-principal	Principal for authentication.
mesos-secret	Secret for authentication.
mesos-framework-register-timeout	Timeout to wait for Framework to register.
mesos-framework-register-timeout-units	Time units to wait for register.
mesos-worker-wait-timeout	Timeout to wait for worker to be created.
mesos-worker-wait-timeout-units	Time units for waiting creation.
mesos-worker-kill-timeout	Number of units to wait for killing a worker.
mesos-worker-kill-timeout-units	Time units to wait for killing.
mesos-docker-command	Command to use at start for each worker.
mesos-containerizer	Containers to use: (`MESOS`/`DOCKER`)
mesos-docker-network-type	Network type to use: (`BRIDGE`/`HOST`/`USER`)
mesos-docker-network-name	Network name to use for workers.
mesos-docker-mount-volume	Mount volume on workers? (`true`/`false`)
mesos-docker-volume-host-path	Host path for mounting volume.
mesos-docker-volume-container-path	Container path to mount volume.

TimeUnit avialable values: `DAYS`, `HOURS`, `MICROSECONDS`, `MILLISECONDS`, `MINUTES`, `NANOSECONDS`, `SECONDS`.

3.6.3.5 Services configuration

To allow COMPSs applications to use WebServices as tasks, the `resources.xml` can include a special type of resource called *Service*. For each WebService it is necessary to specify its wsdl, its name, its namespace and its port.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResourcesList>
    <ComputeNode Name="localhost">
        ...
    </ComputeNode>

    <Service wsdl="http://bscgrid05.bsc.es:20390/hmmerobj/hmmerobj?wsdl">
        <Name>HmmerObjects</Name>
        <Namespace>http://hmmerobj.worker</Namespace>
```

```xml
            <Port>HmmerObjectsPort</Port>
        </Service>
</ResourcesList>
```

When configuring the `project.xml` file it is necessary to include the service as a worker by adding an special entry indicating only the name and the limit of tasks as shown in the following example:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Project>
    <MasterNode/>
    <ComputeNode Name="localhost">
        ...
    </ComputeNode>

    <Service wsdl="http://bscgrid05.bsc.es:20390/hmmerobj/hmmerobj?wsdl">
        <LimitOfTasks>2</LimitOfTasks>
    </Service>
</Project>
```

3.6.3.6 HTTP configuration

To enable execution of HTTP tasks, *Http* resources must be included in the `resources` file as shown in the following example. Please note that the *BaseUrl* attribute is the unique identifier of each Http resource. However, it's possible to assign a single resource to multiple *services* and in the same way one *service* can be executed on various *resources*.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ResourcesList>
    <ComputeNode Name="localhost">
        ...
    </ComputeNode>

    <Http BaseUrl="http://remotehost:1992/test/">
        <ServiceName>service_1</ServiceName>
        <ServiceName>service_2</ServiceName>
    </Http>

    <Http BaseUrl="http://remotehost:2020/print/">
        <ServiceName>service_2</ServiceName>
        <ServiceName>service_3</ServiceName>
    </Http>

</ResourcesList>
```

Configuration of the `project` file must have the Http worker(s) as well, in order to let the runtime know limit of tasks to be executed in parallel on resources.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Project>
    <MasterNode/>
    <ComputeNode Name="localhost">
        ...
    </ComputeNode>

    <Http BaseUrl="http://remotehost:1992/test/">
```

```xml
            <LimitOfTasks>1</LimitOfTasks>
        </Http>

        <Http BaseUrl="http://remotehost:2020/print/">
            <LimitOfTasks>1</LimitOfTasks>
        </Http>

</Project>
```

Chapter 4

Application development

This section is intended to walk you through the development of COMPSs applications.

4.1 Java

This section illustrates the steps to develop a Java COMPSs application, to compile and to execute it. The *Simple* application will be used as reference code. The user is required to select a set of methods, invoked in the sequential application, that will be run as remote tasks on the available resources.

4.1.1 Programming Model

This section shows how the COMPSs programming model is used to develop a Java task-based parallel application for distributed computing. First, We introduce the structure of a COMPSs Java application and with a simple example. Then, we will provide a complete guide about how to define the application tasks. Finally, we will show special API calls and other optimization hints.

4.1.1.1 Application Overview

A COMPSs application is composed of three parts:

- **Main application code:** the code that is executed sequentially and contains the calls to the user-selected methods that will be executed by the COMPSs runtime as asynchronous parallel tasks.
- **Remote methods code:** the implementation of the tasks.
- **Task definition interface:** It is a Java annotated interface which declares the methods to be run as remote tasks along with metadata information needed by the runtime to properly schedule the tasks.

The main application file name has to be the same of the main class and starts with capital letter, in this case it is **Simple.java**. The Java annotated interface filename is *application name + Itf.java*, in this case it is **SimpleItf.java**. And the code that implements the remote tasks is defined in the *application name + Impl.java* file, in this case it is **SimpleImpl.java**.

All code examples are in the `/home/compss/tutorial_apps/java/` folder of the development environment.

Main application code

In COMPSs, the user's application code is kept unchanged, no API calls need to be included in the main application code in order to run the selected tasks on the nodes.

The COMPSs runtime is in charge of replacing the invocations to the user-selected methods with the creation of remote tasks also taking care of the access to files where required. Let's consider the Simple application example that takes an integer as input parameter and increases it by one unit.

The main application code of Simple application is shown in the following code block. It is executed sequentially until the call to the **increment()** method. COMPSs, as mentioned above, replaces the call to this method with the generation of a remote task that will be executed on an available node.

Code 7: Simple in Java (Simple.java)

```java
package simple;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import simple.SimpleImpl;

public class Simple {

  public static void main(String[] args) {
    String counterName = "counter";
    int initialValue = args[0];

    //--------------------------------------------------------------//
    // Creation of the file which will contain the counter variable //
    //--------------------------------------------------------------//
    try {
        FileOutputStream fos = new FileOutputStream(counterName);
        fos.write(initialValue);
        System.out.println("Initial counter value is " + initialValue);
        fos.close();
    }catch(IOException ioe) {
        ioe.printStackTrace();
    }

    //-------------------------------------------//
    //          Execution of the program         //
    //-------------------------------------------//
    SimpleImpl.increment(counterName);

    //-------------------------------------------//
    //    Reading from an object stored in a File   //
    //-------------------------------------------//
    try {
        FileInputStream fis = new FileInputStream(counterName);
        System.out.println("Final counter value is " + fis.read());
        fis.close();
    }catch(IOException ioe) {
        ioe.printStackTrace();
    }
  }
}
```

Remote methods code

The following code contains the implementation of the remote method of the *Simple* application that will be executed remotely by COMPSs.

Code 8: Simple Implementation (SimpleImpl.java)

```java
package simple;

import  java.io.FileInputStream;
import  java.io.FileOutputStream;
import  java.io.IOException;
import  java.io.FileNotFoundException;

public class SimpleImpl {
  public static void increment(String counterFile) {
    try{
      FileInputStream fis = new FileInputStream(counterFile);
      int count = fis.read();
      fis.close();
      FileOutputStream fos = new FileOutputStream(counterFile);
      fos.write(++count);
      fos.close();
    }catch(FileNotFoundException fnfe){
      fnfe.printStackTrace();
    }catch(IOException ioe){
      ioe.printStackTrace();
    }
  }
}
```

Task definition interface

This Java interface is used to declare the methods to be executed remotely along with Java annotations that specify the necessary metadata about the tasks. The metadata can be of three different types:

1. For each parameter of a method, the data type (currently *File* type, primitive types and the *String* type are supported) and its directions (IN, OUT, INOUT, COMMUTATIVE or CONCURRENT).
2. The Java class that contains the code of the method.
3. The constraints that a given resource must fulfill to execute the method, such as the number of processors or main memory size.

The task description interface of the Simple app example is shown in the following figure. It includes the description of the *Increment()* method metadata. The method interface contains a single input parameter, a string containing a path to the file counterFile. In this example there are constraints on the minimum number of processors and minimum memory size needed to run the method.

Code 9: Interface of the Simple application (SimpleItf.java)

```java
package simple;

import  es.bsc.compss.types.annotations.Constraints;
import  es.bsc.compss.types.annotations.task.Method;
import  es.bsc.compss.types.annotations.Parameter;
import  es.bsc.compss.types.annotations.parameter.Direction;
import  es.bsc.compss.types.annotations.parameter.Type;

public interface SimpleItf {
```

```
    @Constraints(computingUnits = "1", memorySize = "0.3")
    @Method(declaringClass = "simple.SimpleImpl")
    void increment(
        @Parameter(type = Type.FILE, direction = Direction.INOUT)
        String file
    );

}
```

The following sections show a detailed guide of how to implement complex applications.

4.1.1.2 Task definition reference guide

The task definition interface is a Java annotated interface where developers define tasks as annotated methods in the interfaces. Annotations can be of three different types:

1. Task-definition annotations are method annotations to indicate which type of task is a method declared in the interface.
2. The Parameter annotation provides metadata about the task parameters, such as data type, direction and other property for runtime optimization.
3. The Constraints annotation describes the minimum capabilities that a given resource must fulfill to execute the task, such as the number of processors or main memory size.
4. The Prolog/Epilog annotations are definitions of binaries to be run before/after the task execution.
5. Scheduler hint annotation provides information about how to deal with tasks of this type at scheduling and execution.

A complete and detailed explanation of the usage of the metadata includes:

Task-definition Annotations

For each declared methods, developers has to define a task type. The following list enumerates the possible task types:

- **@Method:** Defines the Java method as a task
  - **declaringClass** (Mandatory) String specifying the class that implements the Java method.
  - **targetDirection** This field specifies the direction of the target object of an object method. It can be defined as: INOUT" (default value) if the method modifies the target object, "CONCURRENT" if this object modification can be done concurrently, or "IN" if the method does not modify the target object. ().
  - **priority** "true" if the task takes priority and "false" otherwise. This parameter is used by the COMPSs scheduler (it is a String not a Java boolean).
  - **onFailure** Expected behaviour if the task fails. *OnFailure.RETRY* (default value) makes the task be executed again, *OnFailure.CANCEL_SUCCESSORS* ignores the failure and cancels the succesor tasks, *OnFailure.FAIL* stops the whole application in a save mode once a task fails or *OnFailure.IGNORE* ignores the failure and continues with normal runtime execution.
- **@Binary:** Defines the Java method as a binary invokation
  - **binary** (Mandatory) String defining the full path of the binary that must be executed.
  - **workingDir** Full path of the binary working directory inside the COMPSs Worker.
  - **priority** "true" if the task takes priority and "false" otherwise. This parameter is used by the COMPSs scheduler (it is a String not a Java boolean).
- **@MPI:** Defines the Java method as a MPI invokation
  - **mpiRunner** (Mandatory) String defining the mpi runner command.
  - **binary** (Mandatory) String defining the full path of the binary that must be executed.
  - **processes** String defining the number of MPI processes spawn in the task execution. This can be combined with the constraints annotation to create define a MPI+OpenMP task. (Default is 1)

- **scaleByCU** It indicates that the defined *processes* will be scaled by the defined *computingUnits* in the constraints. So, the total MPI processes will be *processes* multiplied by *computingUnits*. This functionality is used to groups MPI processes per node. Number of groups will be set in processes and the number of processes per node will be indicated by *computingUnits*
    - **workingDir** Full path of the binary working directory inside the COMPSs Worker.
    - **priority** "true" if the task takes priority and "false" otherwise. This parameter is used by the COMPSs scheduler (it is a String not a Java boolean).
- **@OmpSs:** Defines the Java method as a OmpSs invokation
    - **binary** (Mandatory) String defining the full path of the binary that must be executed.
    - **workingDir** Full path of the binary working directory inside the COMPSs Worker.
    - **priority** "true" if the task takes priority and "false" otherwise. This parameter is used by the COMPSs scheduler (it is a String not a Java boolean).
- **@Http:** It specifies the HTTP task properties.
    - **serviceName** Mandatory. Name of the HTTP Service that included at least one HTTP resource in the resources file.
    - **resource** Mandatory. URL extension to be concatenated with HTTP resource's base URL.
    - **request** Mandatory. Type of the HTTP request (GET, POST, etc.).
    - **payload** Payload string of POST requests if any. Payload strings can contain any kind of a COMPSs Parameter as long as it is defined between double curly brackets as '{{parameter_name}}'. File parameters can also be used simply by including only the file parameter name.
    - **payloadType** Payload type of POST requests (e.g: 'application/json').
    - **produces** In case of JSON responses, produces string can be used as a template to define 2 things; the first one is where the return value(s) is (are) stored in the retrieved JSON string. Returns are meant to be defined as '{{return_0}}','{{return_1}}', etc. And the second one is for additional parameters to be used 'updates' string. The user assign a value from the JSON response to a parameter and use that param to update an INOUT dictionary.
    - **updates** (PyCOMPSs only) In case of INOUT dictionaries, the user can update the INOUT dict with a value extracted from the JSON response.

For task which are not methods, a representative method has to be defined in an specific class depending on the task type (binary.BINARY in the case of binary tasks, mpi.MPI for mpi tasks, . . .). This is required just for compilation and to enable the invocation of the task from the main code, the runtime will substitute this code by the real execution of the defined task. An example of this representative method can be found in Code 10

Code 10: Representative method for an MPI task

```java
package mpi;

public class MPI {
  public static int mpiExecution(int i, String outFile) {
    // Nothing to do
    return 0
  }
}
```

Parameter-level annotations

For each parameter of task (method declared in the interface), the user must include a **@Parameter** annotation. The properties

- **Direction:** Describes how a task uses the parameter (Default is IN).
    - **Direction.IN:** Task only reads the data.
    - **Direction.INOUT:** Task reads and modifies
    - **Direction.OUT:** Task completely modify the data, or previous content or not modified data is not important.
    - **Direction.COMMUTATIVE:** An INOUT usage of the data which can be re-ordered with other executions of the defined task.
    - **Direction.CONCURRENT:** The task allow concurrent modifications of this data. It requires a storage backend that manages concurrent modifications.

- **Type:** Describes the data type of the task parameter. By default, the runtime infers the type according to the Java datatype. However, it is mandatory to define it for files, directories and Streams.
  COMPSs supports the following types for task parameters:
    - **Basic types:** To indicate a parameter is a Java primitive type use the follwing types: *Type.BOOLEAN, Type.CHAR, Type.BYTE, Type.SHORT, Type.INT, Type.LONG, Type.FLOAT, Type.DOUBLE.* They can only have **IN** direction, since primitive types in Java are always passed by value.
    - **String:** To indicate a parameter is a Java String use *Type.STRING.* It can only have **IN** direction, since Java Strings are immutable.
    - **File:** The real Java type associated with a file parameter is a String that contains the path to the file. However, if the user specifies a parameter as *Type.FILE*, COMPSs will treat it as such. It can have any direction (IN, OUT, INOUT, CONMMUTATIVE or CONCURRENT).
    - **Directory:** The real Java type associated with a directory parameter is a String that contains the path to the directory. However, if the user specifies a parameter as *Type.DIRECTORY*, COMPSs will treat it as such. It can have any direction (IN, OUT, INOUT, CONMMUTATIVE or CONCURRENT).
    - **Object:** An object parameter is defined with *Type.Object.* It can have any direction (IN, INOUT, COMMUTATIVE or CONCURRENT).
    - **Streams:** A Task parameters can be defined as stream with Type.STREAM. It can have direction IN, if the task pull data from the stream, or OUT if the task pushes data to the stream.
- **Return type:** Any object or a generic class object. In this case the direction is always OUT. Basic types are also supported as return types. However, we do not recommend to use them because they cause an implicit synchronization
- **StdIOStream:** For non-native tasks (binaries, MPI, and OmpSs) COMPSs supports the automatic redirection of the Linux streams by specifying StdIOStream.STDIN, StdIOStream.STDOUT or StdIOStream.STDERR. Notice that any parameter annotated with the stream annotation must be of type *Type.FILE*, and with direction *Direction.IN* for *StdIOStream.STDIN* or *Direction.OUT/ Direction.INOUT* for *StdIOStream.STDOUT* and *StdIOStream.STDERR*.
- **Prefix:** For non-native tasks (binaries, MPI, and OmpSs) COMPSs allows to prepend a constant String to the parameter value to use the Linux joint-prefixes as parameters of the binary execution.
- **Weight:** Provides a hint of the size of this parameter compared to a default one. For instance, if a parameters is 3 times larger than the others, set the weigh property of this paramenter to 3.0. (Default is 1.0).
- **keepRename:** Runtime rename files to avoid some data dependencies. It is transparent to the final user because we rename back the filename when invoking the task at worker. This management creates an overhead, if developers know that the task is not name nor extension sensitive (i.e can work with rename), they can set this property to true to reduce the overhead.

Constraints annotations

- **@Constraints:** The user can specify the capabilities that a resource must have in order to run a method. For example, in a cloud execution the COMPSs runtime creates a VM that fulfils the specified requirements in order to perform the execution. A full description of the supported constraints can be found in Table 14.

Prolog & Epilog annotations

- **@Prolog:** Defines a binary to be run right before the task execution.
    - **binary**: the binary to be executed.
    - **params**: describe the command line arguments of the binary.
    - **failByExitValue**: is used to indicate the behaviour when the prolog or epilog returns an exit value different than zero. Users can set the `failByExitValue` to *True*, if they want to consider the exit value as a task failure.
- **@Epilog:** Defines a binary to be run right after the task execution finishes.
    - **binary** , **params**, **failByExitValue** with the same behaviours as Prolog.

Scheduler annotations

- **@SchedulerHints:** It specifies hints for the scheduler about how to treat the task.
  - **isReplicated** "true" if the method must be executed in all the worker nodes when invoked from the main application (it is a String not a Java boolean).
  - **isDistributed** "true" if the method must be scheduled in a forced round robin among the available resources (it is a String not a Java boolean).

4.1.1.3 Alternative method implementations

Since version 1.2, the COMPSs programming model allows developers to define sets of alternative implementations of the same method in the Java annotated interface. Code 11 depicts an example where the developer sorts an integer array using two different methods: merge sort and quick sort that are respectively hosted in the *packagepath.Mergesort* and *packagepath.Quicksort* classes.

Code 11: Alternative sorting method definition example

```java
@Method(declaringClass = "packagepath.Mergesort")
@Method(declaringClass = "packagepath.Quicksort")
void sort(
    @Parameter(type = Type.OBJECT, direction = Direction.INOUT)
    int[] array
);
```

As depicted in the example, the name and parameters of all the implementations must coincide; the only difference is the class where the method is implemented. This is reflected in the attribute *declaringClass* of the *@Method* annotation. Instead of stating that the method is implemented in a single class, the programmer can define several instances of the *@Method* annotation with different declaring classes.

As independent remote methods, the sets of equivalent methods might have common restrictions to be fulfilled by the resource hosting the execution. Or even, each implementation can have specific constraints. Through the *@Constraints* annotation, developers can specify the common constraints for a whole set of methods. In the following example (Code 12) only one core is required to run the method of both sorting algorithms.

Code 12: Alternative sorting method definition with constraint example

```java
@Constraints(computingUnits = "1")
@Method(declaringClass = "packagepath.Mergesort")
@Method(declaringClass = "packagepath.Quicksort")
void sort(
    @Parameter(type = Type.OBJECT, direction = Direction.INOUT)
    int[] array
);
```

However, these sorting algorithms have different memory consumption, thus each algorithm might require a specific amount of memory and that should be stated in the implementation constraints. For this purpose, the developer can add a *@Constraints* annotation inside each *@Method* annotation containing the specific constraints for that implementation. Since the Mergesort has a higher memory consumption than the quicksort, the Code 13 sets a requirement of 1 core and 2GB of memory for the mergesort implementation and 1 core and 500MB of memory for the quicksort.

Code 13: Alternative sorting method definition with specific constraints example

```java
@Constraints(computingUnits = "1")
@Method(declaringClass = "packagepath.Mergesort", constraints = @Constraints(memorySize = "2.0
↪"))
```

```
@Method(declaringClass = "packagepath.Quicksort", constraints = @Constraints(memorySize = "0.5
↪"))
void sort(
    @Parameter(type = Type.OBJECT, direction = Direction.INOUT)
    int[] array
);
```

4.1.1.4 Java API calls

COMPSs also provides a explicit synchronization call, namely *barrier*, which can be used through the COMPSs Java API. The use of *barrier* forces to wait for all tasks that have been submitted before the barrier is called. When all tasks submitted before the *barrier* have finished, the execution continues (Code 14).

Code 14: COMPSs.barrier() example

```
import es.bsc.compss.api.COMPSs;

public class Main {
    public static void main(String[] args) {
        // Setup counterName1 and counterName2 files
        // Execute task increment 1
        SimpleImpl.increment(counterName1);
        // API Call to wait for all tasks
        COMPSs.barrier();
        // Execute task increment 2
        SimpleImpl.increment(counterName2);
    }
}
```

When an object is used in a task, COMPSs runtime store the references of these object in the runtime data structures and generate replicas and versions in remote workers. COMPSs is automatically removing these replicas for obsolete versions. However, the reference of the last version of these objects could be stored in the runtime data-structures preventing the garbage collector to remove it when there are no references in the main code. To avoid this situation, developers can indicate the runtime that an object is not going to use any more by calling the *deregisterObject* API call. Code 15 shows a usage example of this API call.

Code 15: COMPSs.deregisterObject() example

```
import es.bsc.compss.api.COMPSs;

public class Main {
    public static void main(String[] args) {
        final int ITERATIONS = 10;
        for (int i = 0; i < ITERATIONS; ++i) {
            Dummy d = new Dummy(d);
            TaskImpl.task(d);
            /*Allows garbage collector to delete the
              object from memory when the task is finished */
            COMPSs.deregisterObject((Object) d);
        }
    }
}
```

To synchronize files, the *getFile* API call synchronizes a file, returning the last version of file with its original name. Code 16 contains an example of its usage.

Code 16: COMPSs.getFile() example

```
import es.bsc.compss.api.COMPSs;

public class Main {
    public static void main(String[] args) {
        for (int i=0; i<1; i++) {
            TaskImpl.task(FILE_NAME, i);
        }
        /*Waits until all tasks have finished and
          synchronizes the file with its last version*/
        COMPSs.getFile(FILE_NAME);
    }
}
```

4.1.1.5 Managing Failures in Tasks

COMPSs provide mechanism to manage failures in tasks. Developers can specify two properties in the task definition what the runtime should do when a task is blocked or failed.

The *timeOut* property indicates the runtime that a task of this type is considered failed when its duration is larger than the value specified in the property (in seconds)

The *onFailure* property indicates what to do when a task of this type is failed. The possible values are:

- *OnFaiure.RETRY* (Default): The task is executed twice in the same worker and a different worker.
- *OnFailure.CANCEL_SUCCESSORS*: All successors of this task are canceled.
- *OnFailure.FAIL*: The task failure produces a failure of the whole application.
- *OnFailure.IGNORE*: The task failure is ignored and the output parameters are set with empty values.

Usage examples of these properties are shown in Code 17

Code 17: Failure example

```
public interface FailuresItf{
   @Method(declaringClass = "example.Example", timeOut = "3000", onFailure = OnFailure.IGNORE)
   void task_example(@Parameter(type = Type.FILE, direction = Direction.OUT) String fileName);
}
```

4.1.1.6 Tasks Groups and COMPSs exceptions

COMPSs allows users to define task groups which can be combined with an special exception (`COMPSsException`) that the user can use to achieve parallel distributed try/catch blocks; Code 18 shows an example of *COMPSsException* raising. In this case, the group definition is blocking, and waits for all task groups to finish. If a task of the group raises a *COMPSsException*, it will be captured by the runtime which reacts to it by canceling the running and pending tasks of the group and forwarding the COMPSsException to enable the execution except clause. Consequenty, the *COMPSsException* must be combined with task groups.

Code 18: COMPSs Exception example

```
...
    try (COMPSsGroup a = new COMPSsGroup("GroupA")) {
        for (int j = 0; j < N; j++) {
            Test.taskWithCOMPSsException(FILE_NAME);
        }
    } catch (COMPSsException e) {
        Test.otherTask(FILE_NAME);
```

```
    }
...
```

It is possible to use a non-blocking task group for asynchronous behaviour (see Code 19). In this case, the try/catch can be defined later in the code surrounding the *COMPSs.barrierGroup*, enabling to check exception from the defined groups without retrieving data while other tasks are being executed.

Code 19: COMPSs Exception example

```
...
for (int i=0; i<10; i++){
    try (COMPSsGroup a = new COMPSsGroup("Group" + i, false)) {
        for (int j = 0; j < N; j++) {
            Test.taskWithCOMPSsException(FILE_NAME);
        }
    } catch (Exception e) {
        //This is just for compilation. Exception not catch here!
    }
}
for (int i=0; i<10; i++){
    // The group exception will be thrown from the barrier
    try {
        COMPSs.barrierGroup("FailedGroup2");
    } catch (COMPSsException e) {
        System.out.println("Exception caught in barrier!!");
        Test.otherTask(FILE_NAME);
    }
}
```

Attention: Method tasks are executed on top of Java threads, to perform a secure cancellation of a running task in a thread when using the time *timeout* property and *COMPSsExceptions, you have to use the *COMPSs-Worker.cancellationPoint* method to indicate the points where it is secure to cancel a task. When the task code reaches this method, it will check if the current task must be cancelled and perform a save cancellation, otherwise it will continue with this. An example about how to use the cancellation point is shown in Code 20

Code 20: COMPSs Exception example

```
import es.bsc.compss.worker.COMPSsWorker;

public class TasksImpl {

  public static void cancellableTask(String fileName) throws Exception {
    boolean condition = treu
    while (condition) {
        COMPSsWorker.cancellationPoint();
        condition = computeIteration(...);
    }
  }
}
```

4.1.2 Application Compilation

A COMPSs Java application needs to be packaged in a *jar* file containing the class files of the main code, of the methods implementations and of the *Itf* annotation. This jar package can be generated using the commands available in the Java SDK or creating your application as a Apache Maven project.

To integrate COMPSs in the maven compile process you just need to add the *compss-api* artifact as dependency in the application project.

```
<dependencies>
    <dependency>
        <groupId>es.bsc.compss</groupId>
        <artifactId>compss-api</artifactId>
        <version>${compss.version}</version>
    </dependency>
</dependencies>
```

To build the jar in the maven case use the following command

```
$ mvn package
```

Next we provide a set of commands to compile the Java Simple application (detailed at *Java Sample applications*).

```
$ cd tutorial_apps/java/simple/src/main/java/simple/
$~/tutorial_apps/java/simple/src/main/java/simple$ javac *.java
$~/tutorial_apps/java/simple/src/main/java/simple$ cd ..
$~/tutorial_apps/java/simple/src/main/java$ jar cf simple.jar simple/
$~/tutorial_apps/java/simple/src/main/java$ mv ./simple.jar ../../../jar/
```

In order to properly compile the code, the CLASSPATH variable has to contain the path of the *compss-engine.jar* package. The default COMPSs installation automatically add this package to the CLASSPATH; please check that your environment variable CLASSPATH contains the *compss-engine.jar* location by running the following command:

```
$ echo $CLASSPATH | grep compss-engine
```

If the result of the previous command is empty it means that you are missing the *compss-engine.jar* package in your classpath. We recommend to automatically load the variable by editing the *.bashrc* file:

```
$ echo "# COMPSs variables for Java compilation" >> ~/.bashrc
$ echo "export CLASSPATH=$CLASSPATH:/opt/COMPSs/Runtime/compss-engine.jar" >> ~/.bashrc
```

If you are using an IDE (such as Eclipse or NetBeans) we recommend you to add the *compss-engine.jar* file as an external file to the project. The *compss-engine.jar* file is available at your current COMPSs installation under the following path: `/opt/COMPSs/Runtime/compss-engine.jar`

Please notice that if you have performed a custom installation, the location of the package can be different.

4.1.3 Application Execution

A Java COMPSs application is executed through the *runcompss* script. An example of an invocation of the script is:

```
$ runcompss --classpath=/home/compss/tutorial_apps/java/simple/jar/simple.jar simple.Simple 1
```

A comprehensive description of the *runcompss* command is available in the *Executing COMPSs applications* section.

In addition to Java, COMPSs supports the execution of applications written in other languages by means of bindings. A binding manages the interaction of the no-Java application with the COMPSs Java runtime, providing the necessary language translation.

4.2 Python Binding

COMPSs features a binding for Python 2 and 3 applications. The next subsections explain how to program a Python application for COMPSs and a brief overview on how to execute it.

4.2.1 Programming Model

The programming model for Python is structured in the following sections:

4.2.1.1 Task Definition

The task definition is structured in the following sections:

Task Selection

As in the case of Java, a COMPSs Python application is a Python sequential program that contains calls to tasks. In particular, the user can select as a task:

- Functions
- Instance methods: methods invoked on objects
- Class methods: static methods belonging to a class

The task definition in Python is done by means of Python decorators instead of an annotated interface. In particular, the user needs to add a `@task` decorator that describes the task before the definition of the function/method.

As an example (Code 21), let us assume that the application calls a function **foo**, which receives a file path (`file_-path` – string parameter) and a string parameter (`value`). The code of **foo** appends the `value` into `file_path`.

Code 21: Python application example

```python
def foo(file_path, value):
    """ Update the file 'file_path' with the 'value'"""
    with open(file_path, "a") as fd:
        fd.write(value)

def main():
    my_file = "sample_file.txt"
    with open(my_file, "w") as fd:
        fd.write("Hello")
    foo(my_file, "World")

if __name__ == '__main__':
    main()
```

In order to select **foo** as a task, the corresponding `@task` decorator needs to be placed right before the definition of the function, providing some metadata about the parameters of that function. The `@task` decorator has to be imported from the *pycompss* library (Code 22).

Code 22: Python task import

```python
from pycompss.api.task import task

@task(metadata)
def foo(parameters):
    ...
```

See complete example

Code 23: Python application example with @task definition

```python
from pycompss.api.task import task
from pycompss.api.parameter import FILE_INOUT


@task(file_path=FILE_INOUT)
def foo(file_path, value):
    """ Update the file 'file_path' with the 'value'"""
    with open(file_path, "a") as fd:
        fd.write(value)


def main():
    my_file = "sample_file.txt"
    with open(my_file, "w") as fd:
        fd.write("Hello")
    foo(my_file, "World")


if __name__ == '__main__':
    main()
```

Tip: The PyCOMPSs task api also provides the `@task` decorator in camelcase (`@Task`) with the same functionality.

The rationale of providing both `@task` and `@Task` relies on following the PEP8 naming convention. Decorators are usually defined using lowercase, but since the task decorator is implemented following the class pattern, its name is also available as camelcase.

Important: The file that contains tasks definitions **MUST ONLY** contain imports or the `if __name__ == "__main__"` section at the root level. For example, Code 22 includes only the import for the task decorator, and the main code is included into the `main` function.

The rationale of this is due to the fact that the module is loaded from PyCOMPSs. Since the code included at the root level of the file is executed when the module is loaded, this causes the execution to crash.

Function parameters

The *@task* decorator does not interfere with the function parameters, Consequently, the user can define the function parameters as normal python functions (Code 24).

Code 24: Task function parameters example

```python
@task()
def foo(param1, param2):
    ...
```

The use of `*args` and `**kwargs` as function parameters is supported (Code 25).

Code 25: Python task `*args` and `**kwargs` example

```python
@task(returns=int)
def argkwarg_foo(*args, **kwargs):
    ...
```

And even with other parameters, such as usual parameters and *default defined arguments*. Code 26 shows an example of a task with two three parameters (whose one of them (`s`) has a default value (`2`)), `*args` and `**kwargs`.

Code 26: Python task with default parameters example

```python
@task(returns=int)
def multiarguments_foo(v, w, s=2, *args, **kwargs):
    ...
```

Tasks within classes

Functions within classes can also be declared as tasks as normal functions. The main difference is the existence of the `self` parameter which enables to modify the callee object.

For tasks corresponding to instance methods, by default the task is assumed to modify the callee object (the object on which the method is invoked). The programmer can tell otherwise by setting the `target_direction` argument of the *@task* decorator to `IN` (Code 27).

Code 27: Python instance method example

```python
class MyClass(object):
    ...
    @task(target_direction=IN)
    def instance_method(self):
        ... # self is NOT modified here
```

Class methods and static methods can also be declared as tasks. The only requirement is to place the `@classmethod` or `@staticmethod` over the *@task* decorator (Code 28). Note that there is no need to use the `target_direction` flag within the *@task* decorator.

Code 28: Python `@classmethod` and `@staticmethod` tasks example
ple

```python
class MyClass(object):
    ...
    @classmethod
    @task()
    def class_method(cls, a, b, c):
        ...

    @staticmethod
    @task(returns=int)
    def static_method(a, b, c):
        ...
```

Tip: Tasks inheritance and overriding supported!!!

Caution: The objects used as task parameters **MUST BE** serializable:

- Implement the `__getstate__` and `__setstate__` functions in their classes for those objects that are not automatically serializable.
- The classes must not be declared in the same file that contains the main method (if `__name__ == '__main__'`) (known pickle issue).

Important: For instances of user-defined classes, the classes of these objects should have an empty constructor, otherwise the programmer will not be able to invoke task instance methods on those objects (Code 29).

Code 29: Using user-defined classes as task returns

```python
# In file utils.py
from pycompss.api.task import task
class MyClass(object):
    def __init__(self): # empty constructor
        ...

    @task()
    def yet_another_task(self):
        # do something with the self attributes
        ...


    ...

# In file main.py
from pycompss.api.task import task
from utils import MyClass

@task(returns=MyClass)
def ret_foo():
    ...
    myc = MyClass()
    ...
    return myc

def main():
    o = ret_foo()
    # invoking a task instance method on a future object can only
    # be done when an empty constructor is defined in the object's
    # class
    o.yet_another_task()

if __name__=='__main__':
    main()
```

See complete example

Code 30: `utils.py`

```python
from pycompss.api.task import task

class MyClass(object):

    def __init__(self):
        """ Initializes self.value with 0 """
        self.value = 0

    @task()
    def yet_another_task(self):
        """ Increments self.value """
        self.value = self.value + 1
```

Code 31: `main.py`

```python
from pycompss.api.task import task
from utils import MyClass
```

```python
from pycompss.api.api import compss_wait_on

@task(returns=MyClass)
def ret_foo():
    myc = MyClass()
    return myc

def main():
    o = ret_foo()
    o.yet_another_task()
    o = compss_wait_on(o)
    print("Value: %d" % o.value)

if __name__=='__main__':
    main()
```

Task Parameters

The metadata corresponding to a parameter is specified as an argument of the `@task` decorator, whose name is the formal parameter's name and whose value defines the type and direction of the parameter. The parameter types and directions can be:

Types
- *Primitive types* (integer, long, float, boolean, strings)
- *Objects* (instances of user-defined classes, dictionaries, lists, tuples, complex numbers)
- *Files*
- *Collections* (instances of lists)
- *Dictionaries* (instances of dictionary)
- *Streams*
- *IO streams* (for binaries)

Direction
- Read-only (`IN` - default or `IN_DELETE`)
- Read-write (`INOUT`)
- Write-only (`OUT`)
- Concurrent (`CONCURRENT`)
- Commutative (`COMMUTATIVE`)

COMPSs is able to automatically infer the parameter type for primitive types, strings and objects, while the user needs to specify it for files. On the other hand, the direction is only mandatory for `INOUT`, `OUT`, `CONCURRENT` and `COMMUTATIVE` parameters.

Note: Please note that in the following cases there is no need to include an argument in the *@task* decorator for a given task parameter:

- Parameters of primitive types (integer, long, float, boolean) and strings: the type of these parameters can be automatically inferred by COMPSs, and their direction is always `IN`.
- Read-only object parameters: the type of the parameter is automatically inferred, and the direction defaults to `IN`.

The parameter metadata is available from the *pycompss* library (Code 32)

Code 32: Python task parameters import

```python
from pycompss.api.parameter import *
```

Objects

The default type for a parameter is object. Consequently, there is no need to use a specific keyword. However, it is necessary to indicate its direction (unless for input parameters):

PARAME-TER	DESCRIPTION
`IN`	The parameter is read-only. The type will be inferred.
`IN_DELETE`	The parameter is read-only. The type will be inferred. Will be automatically removed after its usage.
`INOUT`	The parameter is read-write. The type will be inferred.
`OUT`	The parameter is write-only. The type will be inferred.
`CONCURRENT`	The parameter is read-write with concurrent access. The type will be inferred.
`COMMUTATIVE`	The parameter is read-write with commutative access. The type will be inferred.

Continuing with the example, in Code 33 the decorator specifies that `foo` has a parameter called `obj`, of type object and `INOUT` direction. Note how the second parameter, `i`, does not need to be specified, since its type (integer) and direction (`IN`) are automatically inferred by COMPSs.

<div align="center">Code 33: Python task example with input output object (<code>INOUT</code>)
and input object (<code>IN</code>)</div>

```python
from pycompss.api.task import task
from pycompss.api.parameter import INOUT, IN


@task(obj=INOUT, i=IN)
def foo(obj, i):
    ...
```

The previous task definition can be simplified due to the default `IN` direction for objects (Code 34):

<div align="center">Code 34: Python task example with input output object (<code>INOUT</code>)
simplified</div>

```python
from pycompss.api.task import task
from pycompss.api.parameter import INOUT


@task(obj=INOUT)
def foo(obj, i):
    ...
```

Tip: In order to choose the apropriate direction, a good exercise is to think if the function only consumes the object (`IN`), modifies the object (`INOUT`), or produces an object (`OUT`).

Tip: The `IN_DELETE` definition is intended to one use objects. Consequently, the information related to the object will be released as soon as possible.

The user can also define that the access to a object is concurrent with `CONCURRENT` (Code 35). Tasks that share a `CONCURRENT` parameter will be executed in parallel, if any other dependency prevents this. The `CONCURRENT` direction allows users to have access from multiple tasks to the same object/file during their executions.

Code 35: Python task example with `CONCURRENT`

```python
from pycompss.api.task import task
from pycompss.api.parameter import CONCURRENT


@task(obj=CONCURRENT)
def foo(obj, i):
    ...
```

> **Caution:** COMPSs does not manage the interaction with the objects used/modified concurrently. Taking care of the access/modification of the concurrent objects is responsibility of the developer.

Or even, the user can also define that the access to a parameter is commutative with `COMMUTATIVE` (Code 36). The execution order of tasks that share a `COMMUTATIVE` parameter can be changed by the runtime following the commutative property.

Code 36: Python task example with `COMMUTATIVE`

```python
from pycompss.api.task import task
from pycompss.api.parameter import COMMUTATIVE


@task(obj=COMMUTATIVE)
def foo(obj, i):
    ...
```

Files

It is possible to define that a parameter is a file (`FILE`), and its direction:

PARAMETER	DESCRIPTION
FILE/FILE_IN	The parameter is a file. The direction is assumed to be `IN`.
FILE_INOUT	The parameter is a read-write file.
FILE_OUT	The parameter is a write-only file.
FILE_CONCURRENT	The parameter is a concurrent read-write file.
FILE_COMMUTATIVE	The parameter is a commutative read-write file.

Continuing with the example, in Code 37 the decorator specifies that `foo` has a parameter called `f`, of type `FILE` and `INOUT` direction (`FILE_INOUT`).

Code 37: Python task example with input output file (`FILE_INOUT`)

```python
from pycompss.api.task import task
from pycompss.api.parameter import FILE_INOUT


@task(f=FILE_INOUT)
def foo(f):
    fd = open(f, 'a+')
    ...
    # append something to fd
    ...
    fd.close()


def main():
    f = "/path/to/file.extension"
```

```
    # Populate f
    foo(f)
```

Tip: The value for a FILE (e.g. `f`) is a string pointing to the file to be used at `foo` task. However, it can also be `None` if it is optional. Consequently, the user can define task that can receive a FILE or not, and act accordingly. For example (Code 38):

Code 38: Python task example with optional input file (`FILE_IN`)

```python
from pycompss.api.task import task
from pycompss.api.parameter import FILE_IN

@task(f=FILE_IN)
def foo(f):
    if f:
        # Do something with the file
        with open(f, 'r') as fd:
            num_lines = len(rd.readlines())
        return num_lines
    else:
        # Do something when there is no input file
        return -1

def main():
    f = "/path/to/file.extension"
    # Populate f
    num_lines_f = foo(f)   # num_lines_f == actual number of lines of file.extension
    g = None
    num_lines_g = foo(g)   # num_lines_g == -1
```

The user can also define that the access to file parameter is concurrent with `FILE_CONCURRENT` (Code 39). Tasks that share a `FILE_CONCURRENT` parameter will be executed in parallel, if any other dependency prevents this. The `CONCURRENT` direction allows users to have access from multiple tasks to the same file during their executions.

Code 39: Python task example with `FILE_CONCURRENT`

```python
from pycompss.api.task import task
from pycompss.api.parameter import FILE_CONCURRENT

@task(f=FILE_CONCURRENT)
def foo(f, i):
    ...
```

Caution: COMPSs does not manage the interaction with the files used/modified concurrently. Taking care of the access/modification of the concurrent files is responsibility of the developer.

Or even, the user can also define that the access to a parameter is a file `FILE_COMMUTATIVE` (Code 40). The execution order of tasks that share a `FILE_COMMUTATIVE` parameter can be changed by the runtime following the commutative property.

Code 40: Python task example with `FILE_COMMUTATIVE`

```python
from pycompss.api.task import task
from pycompss.api.parameter import FILE_COMMUTATIVE


@task(f=FILE_COMMUTATIVE)
def foo(f, i):
    ...
```

Directories

In addition to files, it is possible to define that a parameter is a directory (`DIRECTORY`), and its direction:

PARAMETER	DESCRIPTION
DIRECTORY_-IN	The parameter is a directory and the direction is IN. The directory will be compressed before any transfer amongst nodes.
DIRECTORY_-INOUT	The parameter is a read-write directory. The directory will be compressed before any transfer amongst nodes.
DIRECTORY_-OUT	The parameter is a write-only directory. The directory will be compressed before any transfer amongst nodes.

The definition of a `DIRECTORY` parameter is shown in Code 41. The decorator specifies that `foo` has a parameter called d, of type `DIRECTORY` and `INOUT` direction.

Code 41: Python task example with input output directory (`DIRECTORY_INOUT`)

```python
from pycompss.api.task import task
from pycompss.api.parameter import DIRECTORY_INOUT


@task(d=DIRECTORY_INOUT)
def foo(d):
    ...
```

Collections

It is possible to specify that a parameter is a collection of elements (e.g. list) and its direction.

PARAMETER	DESCRIPTION
COLLECTION_IN	The parameter is read-only collection.
COLLECTION_IN_-DELETE	The parameter is read-only collection for single usage (will be automatically removed after its usage).
COLLECTION_INOUT	The parameter is read-write collection.
COLLECTION_OUT	The parameter is write-only collection.

In this case (Code 42), the list may contain sub-objects that will be handled automatically by the runtime. It is important to annotate data structures as collections if in other tasks there are accesses to individual elements of these collections as parameters. Without this annotation, the runtime will not be able to identify data dependences between the collections and the individual elements.

Code 42: Python task example with `COLLECTION (IN)`

```python
from pycompss.api.task import task
from pycompss.api.parameter import COLLECTION

@task(my_collection=COLLECTION)
def foo(my_collection):
    for element in my_collection:
        ...
```

> **Caution:** The current support for collections is limited to **static number of elements** lists.
>
> Consequently, *the length of the collection must be kept* during the execution, and it is *NOT possible to append or delete elements* from the collection in the tasks (only to receive elements or to modify the existing if they are not primitives).

The sub-objects of the collection can be collections of elements (and recursively). In this case, the runtime also keeps track of all elements contained in all sub-collections. In order to improve the performance, the depth of the sub-objects can be limited through the use of the `depth` parameter (Code 43)

Code 43: Python task example with `COLLECTION_IN` and `Depth`

```python
from pycompss.api.task import task
from pycompss.api.parameter import COLLECTION_IN

@task(my_collection={Type:COLLECTION_IN, Depth:2})
def foo(my_collection):
    for inner_collection in my_collection:
        for element in inner_collection:
            # The contents of element will not be tracked
            ...
```

Tip: A collection can contain dictionaries, and will be analyzed automatically.

Tip: If the collection is intended to be used only once with `IN` direction, the `COLLECTION_IN_DELETE` type is recommended, since it automatically removes the entire collection after the task. This enables to release as soon as possible memory and storage.

Collections of files

It is also possible to specify that a parameter is a collection of files (e.g. list) and its direction.

PARAMETER	DESCRIPTION
COLLECTION_FILE/COLLECTION_FILE_IN	The parameter is read-only collection of files.
COLLECTION_FILE_INOUT	The parameter is read-write collection of files.
COLLECTION_FILE_OUT	The parameter is write-only collection of files.

In this case (Code 44), the list may contain files that will be handled automatically by the runtime. It is important to annotate data structures as collections if in other tasks there are accesses to individual elements of these collections as parameters. Without this annotation, the runtime will not be able to identify data dependences between the collections and the individual elements.

Code 44: Python task example with `COLLECTION_FILE` (IN)

```python
from pycompss.api.task import task
from pycompss.api.parameter import COLLECTION_FILE


@task(my_collection=COLLECTION_FILE)
def foo(my_collection):
    for file in my_collection:
        ...
```

The file of the collection can be collections of elements (and recursively). In this case, the runtime also keeps track of all files contained in all sub-collections. In order to improve the performance, the depth of the sub-files can be limited through the use of the `depth` parameter as with objects (Code 43)

> **Caution:** The current support for collections of files is also limited to a **static number of elements**, as with *Collections*.

Dictionaries

It is possible to specify that a parameter is a dictionary of elements (e.g. dict) and its direction.

PARAMETER	DESCRIPTION
DICTIONARY_IN	The parameter is read-only dictionary.
DICTIONARY_IN_-DELETE	The parameter is read-only dictionary for single usage (will be automatically removed after its usage).
DICTIONARY_INOUT	The parameter is read-write dictionary.

As with the collections, it is possible to specify that a parameter is a dictionary of elements (e.g. dict) and its direction (DICTIONARY_IN or DICTIONARY_INOUT) (Code 45), whose sub-objects will be handled automatically by the runtime.

Code 45: Python task example with `DICTIONARY` (IN)

```python
from pycompss.api.task import task
from pycompss.api.parameter import DICTIONARY


@task(my_dictionary=DICTIONARY)
def foo(my_dictionary):
    for k, v in my_dictionary.items():
        ...
```

> **Caution:** The current support for dictionaries is also limited to a **static number of elements**, as with *Collections*.

The sub-objects of the dictionary can be collections or dictionary of elements (and recursively). In this case, the runtime also keeps track of all elements contained in all sub-collections/sub-dictionaries. In order to improve the performance, the depth of the sub-objects can be limited through the use of the `depth` parameter (Code 46)

Code 46: Python task example with `DICTIONARY_IN` and `Depth`

```python
from pycompss.api.task import task
from pycompss.api.parameter import DICTIONARY_IN
```

```python
@task(my_dictionary={Type:DICTIONARY_IN, Depth:2})
def foo(my_dictionary):
    for key, inner_dictionary in my_dictionary.items():
        for sub_key, sub_value in inner_dictionary.items():
            # The contents of element will not be tracked
            ...
```

Tip: A dictionary can contain collections, and will be analyzed automatically.

Tip: If the dictionary is intended to be used only once with `IN` direction, the `DICTIONARY_IN_DELETE` type is recommended, since it automatically removes the entire dictionary after the task. This enables to release as soon as possible memory and storage.

Streams

It is possible to use streams as input or output of the tasks by defining that a parameter is `STREAM` and its direction.

PARAMETER	DESCRIPTION
STREAM_IN	The parameter is a read-only stream.
STREAM_OUT	The parameter is a write-only stream.

For example, Code 47 shows an example using `STREAM_IN` or `STREAM_OUT` parameters This parameters enable to mix a task-driven workflow with a data-driven workflow.

Code 47: Python task example with `STREAM_IN` and `STREAM_OUT`

```python
from pycompss.api.task import task
from pycompss.api.parameter import STREAM_IN
from pycompss.api.parameter import STREAM_OUT

@task(ods=STREAM_OUT)
def write_objects(ods):
    ...
    for i in range(NUM_OBJECTS):
        # Build object
        obj = MyObject()
        # Publish object
        ods.publish(obj)
        ...
    ...
    # Mark the stream for closure
    ods.close()

@task(ods=STREAM_IN, returns=int)
def read_objects(ods):
    ...
    num_total = 0
    while not ods.is_closed():
        # Poll new objects
        new_objects = ods.poll()
        # Process files
```

```
        ...
        # Accumulate read files
        num_total += len(new_objects)
    ...
    # Return the number of processed files
    return num_total
```

The stream parameter also supports Files (Code 48).

Code 48: Python task example with `STREAM_IN` and `STREAM_OUT` for files

```python
from pycompss.api.task import task
from pycompss.api.parameter import STREAM_IN
from pycompss.api.parameter import STREAM_OUT

@task(fds=STREAM_OUT)
def write_files(fds):
    ...
    for i in range(NUM_FILES):
        file_name = str(uuid.uuid4())
        # Write file
        with open(file_path, 'w') as f:
            f.write("Test " + str(i))
        ...
    ...
    # Mark the stream for closure
    fds.close()

@task(fds=STREAM_IN, returns=int)
def read_files(fds):
    ...
    num_total = 0
    while not fds.is_closed():
        # Poll new files
        new_files = fds.poll()
        # Process files
        for nf in new_files:
            with open(nf, 'r') as f:
                ...
        # Accumulate read files
        num_total += len(new_files)
        ...
    ...
    # Return the number of processed files
    return num_total
```

In addition, the stream parameter can also be defined for binary tasks (Code 49).

Code 49: Python task example with `STREAM_OUT` for binaries

```python
from pycompss.api.task import task
from pycompss.api.binary import binary
from pycompss.api.parameter import STREAM_OUT

@binary(binary="file_generator.sh")
@task(fds=STREAM_OUT)
```

```python
def write_files(fds):
    # Equivalent to: ./file_generator.sh > fds
    pass
```

Code 50 shows an example of how streams are used in the main code. In this code snippet we can see how the object representing the data stream is created how the a producer task is invoqued and how the stream data generated at tasks can be poll from the main code.

Code 50: Python task example using streams in the main code

```python
from pycompss.api.task import task
from pycompss.api.parameter import STREAM_OUT
from pycompss.streams.distro_stream import ObjectDistroStream


@task(ods=STREAM_OUT)
def write_objects(ods):
    ...
    for i in range(NUM_OBJECTS):
        # Build object
        obj = MyObject()
        # Publish object
        ods.publish(obj)
        ...
    ...
    # Mark the stream for closure
    ods.close()

@task()
def process_object(obj):
    ...
    # Do something with obj
    ...

if __name__=='__main__':

    ods = ObjectDistroStream()

    # Create producers
    for _ in range(num_producers):
        write_objects(ods, producer_sleep)

    # Process stream
    while not ods.is_closed():
        # Poll new objects
        new_objects = ods.poll()

        # Process received objects
        for obj in new_objects:
            res = process_object(obj)
    ...
```

Standard Streams

Finally, a parameter can also be defined as the standard input, standard output, and standard error.

PARAMETER	DESCRIPTION
STDIN	The parameter is a IO stream for standard input redirection.
STDOUT	The parameter is a IO stream for standard output redirection.
STDERR	The parameter is a IO stream for standard error redirection.

Caution: STDIN, STDOUT and STDERR are only supported in binary tasks

This is particularly useful with binary tasks that consume/produce from standard IO streams, and the user wants to redirect the standard input/output/error to a particular file. Code 51 shows an example of a binary task that invokes *output_ generator.sh* which produces the result in the standard output, and the task takes that output and stores it into *fds*.

Code 51: Python task example with STDOUT for binaries

```python
from pycompss.api.task import task
from pycompss.api.binary import binary
from pycompss.api.parameter import STDOUT

@binary(binary="output_generator.sh")
@task(fds=STDOUT)
def write_files(fds):
    # Equivalent to: ./file_generator.sh > fds
    pass
```

Other Task Parameters

Task time out

The user is also able to define the time out of a task within the @task decorator with the time_out=<TIME_IN_-SECONDS> hint. The runtime will cancel the task if the time to execute the task exceeds the time defined by the user. For example, Code 52 shows how to specify that the unknown_duration_task maximum duration before canceling (if exceeded) is one hour.

Code 52: Python task *time_out* example

```python
@task(time_out=3600)
def unknown_duration_task(self):
    ...
```

Scheduler hints

The programmer can provide hints to the scheduler through specific arguments within the *@task* decorator.

For instance, the programmer can mark a task as a high-priority task with the `priority` argument of the `@task` decorator (Code 53). In this way, when the task is free of dependencies, it will be scheduled before any of the available low-priority (regular) tasks. This functionality is useful for tasks that are in the critical path of the application's task dependency graph.

Code 53: Python task *priority* example

```python
@task(priority=True)
def func():
    ...
```

Moreover, the user can also mark a task as distributed with the *is_distributed* argument or as replicated with the *is_replicated* argument (Code 54). When a task is marked with *is_distributed=True*, the method must be scheduled in a forced round robin among the available resources. On the other hand, when a task is marked with *is_replicated=True*, the method must be executed in all the worker nodes when invoked from the main application. The default value for these parameters is False.

Code 54: Python task *is_distributed* and *is_replicated* examples

```python
@task(is_distributed=True)
def func():
    ...

@task(is_replicated=True)
def func2():
    ...
```

On failure task behaviour

In case a task fails, the whole application behaviour can be defined using the *@on_failure* decorator on top of the *@task* decorator (Code 55). It has four possible values that can be defined with the **management** parameter: **'RETRY'**, **'CANCEL_SUCCESSORS'**, **'FAIL'** and **'IGNORE'**. *'RETRY'* is the default behaviour, making the task to be executed again (on the same worker or in another worker if the failure remains). *'CANCEL_-SUCCESSORS'* ignores the failed task and cancels the execution of the successor tasks, *'FAIL'* stops the whole execution once a task fails and *'IGNORE'* ignores the failure and continues with the normal execution.

Code 55: Python task *@on_failure* decorator example

```python
from pycompss.api.task import task
from pycompss.api.on_failure import on_failure

@on_failure(management ='CANCEL_SUCCESSORS')
@task()
def func():
    ...
```

Since the **'CANCEL_SUCCESSORS'** and **'IGNORE'** policies enable to continue the execution accepting that tasks may have failed, it is possible to define the value for the objects and/or files produced by the failed tasks (INOUT, OUT, FILE_INOUT, FILE_OUT and return). This is considered as the default output objects/files. For example, Code 56 shows a the `func` task which returns one integer. In the case of failure within `func`, the execution of the workflow will continue since the on failure management policy is set to *'IGNORE'*, with 0 as return value.

Code 56: Python task *@on_failure* example with default return value

```python
from pycompss.api.task import task
from pycompss.api.on_failure import on_failure


@on_failure(management='IGNORE', returns=0)
@task(returns=int)
def func():
    ...
```

For the INOUT parameters, the default value can be set by using the parameter name of `func` in the *@on_failure* decorator. Code 57 shows how to define the default value for a FILE_INOUT parameter (named `f_inout`). The example is also valid for FILE_OUT values.

Code 57: Python task *@on_failure* example with default FILE_-INOUT value

```python
from pycompss.api.task import task
from pycompss.api.on_failure import on_failure
from pycompss.api.parameter import FILE_INOUT


@on_failure(management='IGNORE', f_inout="/path/to/default.file")
@task(f_inout=FILE_INOUT)
def func(f_inout):
    ...
```

Tip: The default FILE_INOUT/FILE_OUT can be generated at task generation time by calling a function instead of providing a static file path. Code 58 shows an example of this case, where the default value for the output file produced by `func` is defined by the `generate_empty` function.

Code 58: Python task *@on_failure* example with default FILE_-
OUT value from function

```python
from pycompss.api.task import task
from pycompss.api.on_failure import on_failure
from pycompss.api.parameter import FILE_OUT

def generate_empty(msg, name):
    empty_file = "/tmp/empty_file_" + name
    with open(empty_file, 'w') as f:
        f.write("EMPTY FILE " + msg)
    return empty_file

@on_failure(management='IGNORE', f_out=generate_empty("OUT", "out.tmp"))
@task(f_out=FILE_OUT)
def func(f_inout):
    ...
```

Task Parameters Summary

:numref:`task_arguments` summarizes all arguments that can be found in the *@task* decorator.

Table 8: Arguments of the *@task* decorator

Argument	Value	
Formal parameter name	**(default: empty)**	The parameter is an object or a simple
	IN	Read-only parameter, all types.
	IN_DELETE	Read-only parameter, all types. Automa
	INOUT	Read-write parameter, all types except
	OUT	Write-only parameter, all types except
	CONCURRENT	Concurrent read-write parameter, all ty
	COMMUTATIVE	Commutative read-write parameter, all
	FILE(_IN)	Read-only file parameter.
	FILE_INOUT	Read-write file parameter.
	FILE_OUT	Write-only file parameter.
	FILE_CONCURRENT	Concurrent read-write file parameter.
	FILE_COMMUTATIVE	Commutative read-write file parameter.
	DIRECTORY(_IN)	The parameter is a read-only directory.
	DIRECTORY_INOUT	The parameter is a read-write directory.
	DIRECTORY_OUT	the parameter is a write-only directory.
	COLLECTION(_IN)	Read-only collection parameter (`list`).
	COLLECTION_IN_DELETE	Single usage read-only collection parame
	COLLECTION_INOUT	Read-write collection parameter (`list`).
	COLLECTION_OUT	Read-only collection parameter (`list`).
	COLLECTION_FILE(_IN)	Read-only collection of files parameter (
	COLLECTION_FILE_INOUT	Read-write collection of files parameter
	COLLECTION_FILE_OUT	Read-only collection of files parameter (
	DICTIONARY(_IN)	Read-only dictionary parameter (`dict`).
	DICTIONARY_IN_DELETE	Single usage read-only collection diction
	DICTIONARY_INOUT	Read-write dictionary parameter (`dict`)
	STREAM_IN	The parameter is a read-only stream.
	STREAM_OUT	The parameter is a write-only stream.
	STDIN	The parameter is a file for standard inpu
	STDOUT	The parameter is a file for standard out
	STDERR	The parameter is a file for standard erro

Table 8 – continued from previous page

Argument	Value
	Explicit: {Type:(empty=object)/FILE/COLLECTION/DICTIONARY, Direction:(empty=IN)/IN/ INOUT/OUT/CONCURRENT}
returns	Return type or number of returned elements
target_direction	INOUT (default), IN or CONCURRENT
priority	`True` or `False` (default)
is_distributed	`True` or `False` (default)
is_replicated	`True` or `False` (default)
on_failure	'RETRY' (default), 'CANCEL_SUCCESSORS', 'FAIL' or 'IGNORE'
time_out	`int` (time in seconds)
cache_returns	`True` or `False` (default) - Requires to enable the cache
is_reduce	`True` or `False` (default)
chunk_size	Reduction chunk size (`int`)
numba	True or False (default) or mode (`string`)
numba_flags	Numba flags (`dictionary` of strings)
numba_signature	Numba signature (`list` of strings)
numba_declaration	Numba declaration (`string`)

Task Return

If the function or method returns a value, the programmer can use the *returns* argument within the *@task* decorator. In this argument, the programmer can specify the type of that value (Code 59).

Code 59: Python task returns example

```python
@task(returns=int)
def ret_func():
    return 1
```

Moreover, if the function or method returns more than one value, the programmer can specify how many and their type in the *returns* argument. Code 60 shows how to specify that two values (an integer and a list) are returned.

Code 60: Python task with multireturn example

```python
@task(returns=(int, list))
def ret_func():
    return 1, [2, 3]
```

Alternatively, the user can specify the number of return statements as an integer value (Code 61). This way of specifying the amount of return eases the *returns* definition since the user does not need to specify explicitly the type of the return arguments. However, it must be considered that the type of the object returned when the task is invoked will be a future object. This consideration may lead to an error if the user expects to invoke a task defined within an object returned by a previous task. In this scenario, the solution is to specify explicitly the return type.

Code 61: Python task returns with integer example

```python
@task(returns=1)
def ret_func():
    return "my_string"


@task(returns=2)
def ret_func():
    return 1, [2, 3]
```

Important: If the programmer selects as a task a function or method that returns a value, that value is not generated until the task executes (Code 62).

Code 62: Task return value generation

```python
@task(return=MyClass)
def ret_func():
    return MyClass(...)


...


if __name__=='__main__':
    o = ret_func()  # o is a future object
```

The object returned can be involved in a subsequent task call, and the COMPSs runtime will automatically find the corresponding data dependency. In the following example, the object *o* is passed as a parameter and callee of two subsequent (asynchronous) tasks, respectively (Code 63).

Code 63: Task return value subsequent usage

```python
if __name__=='__main__':
    # o is a future object
    o = ret_func()

    ...

    another_task(o)

    ...

    o.yet_another_task()
```

Tip: PyCOMPSs is able to infer if the task returns something and its amount in most cases. Consequently, the user can specify the task without *returns* argument. But this is discouraged since it requires code analysis, including an overhead that can be avoided by using the *returns* argument.

Tip: PyCOMPSs is compatible with Python 3 type hinting. So, if type hinting is present in the code, PyCOMPSs is able to detect the return type and use it (there is no need to use the *returns*):

Code 64: Python task returns with type hinting

```python
@task()
def ret_func() -> str:
    return "my_string"

@task()
def ret_func() -> (int, list):
    return 1, [2, 3]
```

Other task types

In addition to this API functions, the programmer can use a set of decorators for other purposes.

Important: **NOTE:** If defined, these decorators must be placed after (below) the *@constraint* decorator, and before (on top of) the *@task* decorator.

The following subparagraphs describe their usage.

Binary decorator

The *@binary* (or @Binary) decorator shall be used to define that a task is going to invoke a binary executable.

In this context, the *@task* decorator parameters will be used as the binary invocation parameters (following their order in the function definition). Since the invocation parameters can be of different nature, information on their type can be provided through the *@task* decorator.

Code 65 shows the most simple binary task definition without/with constraints (without parameters); please note that @constraint decorator has to be provided on top of the others.

Code 65: Binary task example

```python
from pycompss.api.task import task
from pycompss.api.binary import binary

@binary(binary="mybinary.bin")
@task()
def binary_func():
    pass

@constraint(computing_units="2")
@binary(binary="otherbinary.bin")
@task()
def binary_func2():
    pass
```

The invocation of these tasks would be equivalent to:

```
$ ./mybinary.bin
$ ./otherbinary.bin     # in resources that respect the constraint.
```

The `@binary` decorator supports the `working_dir` parameter to define the working directory for the execution of the defined binary.

Code 66 shows a more complex binary invocation, with files as parameters:

Code 66: Binary task example 2

```python
from pycompss.api.task import task
from pycompss.api.binary import binary
from pycompss.api.parameter import *

@binary(binary="grep", working_dir=".")
@task(infile={Type:FILE_IN_STDIN}, result={Type:FILE_OUT_STDOUT})
def grepper():
    pass

# This task definition is equivalent to the following, which is more verbose:

@binary(binary="grep", working_dir=".")
@task(infile={Type:FILE_IN, StdIOStream:STDIN}, result={Type:FILE_OUT, StdIOStream:STDOUT})
def grepper(keyword, infile, result):
    pass

if __name__=='__main__':
    infile = "infile.txt"
    outfile = "outfile.txt"
    grepper("Hi", infile, outfile)
```

The invocation of the *grepper* task would be equivalent to:

```
$ # grep keyword < infile > result
$ grep Hi < infile.txt > outfile.txt
```

Please note that the *keyword* parameter is a string, and it is respected as is in the invocation call. Another way of passing task parameters to binary execution command is to use `args` parameter in the binary definition. In this case, task parameters should be defined between curly braces and the full string with parameter replacements will be added to the command. In the following example, value of 'param_1' is added to the execution command after '-d' arg:

Code 67: Binary task example 3

```python
from pycompss.api.task import task
from pycompss.api.binary import binary
from pycompss.api.parameter import *


@binary(binary="date", args= "-d {{param_1}}")
@task()
def print_date(param_1):
    pass

if __name__=='__main__':
    print_date("next Monday")
```

The invocation of the *print_date* task would be equivalent to:

```
$ # date -d param_1
$ date -d "next Monday"
```

Thus, PyCOMPSs can also deal with prefixes for the given parameters. Code 68 performs a system call (ls) with specific prefixes:

Code 68: Binary task example 4

```python
from pycompss.api.task import task
from pycompss.api.binary import binary
from pycompss.api.parameter import *


@binary(binary="ls")
@task(hide={Type:FILE_IN, Prefix:"--hide="}, sort={Prefix:"--sort="})
def myLs(flag, hide, sort):
    pass


if __name__=='__main__':
    flag = '-l'
    hideFile = "fileToHide.txt"
    sort = "time"
    myLs(flag, hideFile, sort)
```

The invocation of the *myLs* task would be equivalent to:

```
$ # ls -l --hide=hide --sort=sort
$ ls -l --hide=fileToHide.txt --sort=time
```

This particular case is intended to show all the power of the *@binary* decorator in conjunction with the *@task* decorator. Please note that although the *hide* parameter is used as a prefix for the binary invocation, the *fileToHide.txt* would also be transfered to the worker (if necessary) since its type is defined as FILE_IN. This feature enables to build more complex binary invocations.

In addition, the `@binary` decorator also supports the `fail_by_exit_value` parameter to define the failure of the task by the exit value of the binary (Code 69). It accepts a boolean (`True` to consider the task failed if the exit value is not 0, or `False` to ignore the failure by the exit value (**default**)), or a string to determine the environment variable that defines the fail by exit value (as boolean). The default behaviour (`fail_by_exit_value=False`) allows users to receive the exit value of the binary as the task return value, and take the necessary decissions based on this value.

Code 69: Binary task example with `fail_by_exit_value`

```python
@binary(binary="mybinary.bin", fail_by_exit_value=True)
@task()
def binary_func():
    pass
```

OmpSs decorator

The *@ompss* (or @OmpSs) decorator shall be used to define that a task is going to invoke a OmpSs executable (Code 70).

Code 70: OmpSs task example

```python
from pycompss.api.ompss import ompss

@ompss(binary="ompssApp.bin")
@task()
def ompss_func():
    pass
```

The OmpSs executable invocation can also be enriched with parameters, files and prefixes as with the *@binary* decorator through the function parameters and *@task* decorator information. Please, check *Binary decorator* for more details.

MPI decorator

The *@mpi* (or @Mpi) decorator shall be used to define that a task is going to invoke a MPI executable (Code 71).

Code 71: MPI task example

```python
from pycompss.api.mpi import mpi

@mpi(binary="mpiApp.bin", runner="mpirun", processes=2)
@task()
def mpi_func():
    pass
```

The MPI executable invocation can also be enriched with parameters, files and prefixes as with the *@binary* decorator through the function parameters and *@task* decorator information. Please, check *Binary decorator* for more details.

The *@mpi* decorator can be also used to execute a MPI for python (mpi4py) code. To indicate it, developers only need to remove the binary field and include the Python MPI task implementation inside the function body as shown in the following example (Code 72).

Code 72: Python MPI task example.

```python
from pycompss.api.mpi import mpi

@mpi(processes=4)
@task()
def layout_test_with_all():
    from mpi4py import MPI
    rank = MPI.COMM_WORLD.rank
    return rank
```

In both cases, users can also define, MPI + OpenMP tasks by using `processes` property to indicate the number of MPI processes and `computing_units` in the Task Constraints to indicate the number of OpenMP threads per MPI process.

Users can also limit the distribution of the MPI processes through the nodes by using the `processes_per_node` property. In the following example (Code 73) the four MPI processes defined in the task will be divided in two groups of two processes. And all the processes of each group will be allocated to the same node. It will ensure that the defined MPI task will use up to two nodes.

Code 73: MPI task example grouping MPI processes

```python
from pycompss.api.mpi import mpi

@mpi(processes=4, processes_per_node=2)
@task()
def layout_test_with_all():
    from mpi4py import MPI
    rank = MPI.COMM_WORLD.rank
    return rank
```

The *@mpi* decorator can be combined with collections to allow the process of a list of parameters in the same MPI execution. By the default, all parameters of the list will be deserialized to all the MPI processes. However, a common pattern in MPI is that each MPI processes performs the computation in a subset of data. So, all data serialization is not needed. To indicate the subset used by each MPI process, developers can use the `data_layout` notation inside the MPI task declaration.

Code 74: MPI task example with collections and data layout

```python
from pycompss.api.mpi import mpi


@mpi(processes=4, col_layout={block_count: 4, block_length: 2, stride: 1})
@task(col=COLLECTION_IN, returns=4)
def layout_test_with_all(col):
    from mpi4py import MPI
    rank = MPI.COMM_WORLD.rank
    return data[0]+data[1]+rank
```

Figure (Code 74) shows an example about how to combine MPI tasks with collections and data layouts. In this example, we have define a MPI task with an input collection (`col`). We have also defined a data layout with the property `<arg_name>_layout` and we specify the number of blocks (`block_count`), the elements per block (`block_length`), and the number of element between the starting block points (`stride`).

Users can specify the MPI runner command with the `runner` how ever the arguments passed to the `mpirun` command differs depending on the implementation. To ensure that the correct arguments are passed to the runner, users can define the `COMPSS_MPIRUN_TYPE` environment variable. The current supported values are `impi` for Intel MPI and *ompi* for OpenMPI. Other MPI implementation can be supported by adding its corresponding properties file in the folder `$COMPSS_HOME/Runtime/configuration/mpi`.

MPMD MPI decorator

The *@mpmd_mpi* decorator can be used to define Multiple Program Multiple Data (MPMD) MPI tasks as shown in the following example (Code 75):

Code 75: MPMD MPI task example

```python
from pycompss.api.mpmd_mpi import mpmd_mpi


@mpmd_mpi(runner="mpirun",
          programs=[
              dict(binary="hostname", processes=2),
              dict(binary="date", processes=2)
          ])
@task()
def basic():
    pass
```

The definition implies that MPMD MPI command will be run by 'mpirun', and will execute 2 processes for 'hostname', and 2 processes to show the ' date'. It's not mandatory to specify total number of programs as long as they are added inside `programs` list of dictionaries argument.

Each of the MPMD MPI programs must at least have `binary`, but also can have `processes` and `args` string (Code 76):

Code 76: MPMD MPI task example

```python
from pycompss.api.mpmd_mpi import mpmd_mpi


@mpmd_mpi(runner="mpirun",
          programs=[
              dict(binary="date", processes=2, args="-d {{first}}"),
              dict(binary="date", processes=4, args="-d {{second}}")
          ])
@task()
```

(continues on next page)

```python
def task_args(first, second):
    pass


def print_monday_friday(self):
    task_args("next monday", "next friday")
    compss_barrier()
```

When executed, this MPMD MPI program would invoke 2 MPI processes to print the date of next Monday, and 4 processes for next Friday. "args" string replaces every parameter that is 'called' between double curly braces with their real value. This allows using multiple `FILE_IN` parameters for multiple MPI programs. Moreover, output of the full MPMD MPI programs can be forwarded to an `FILE_OUT_STDOUT` param:

Code 77: MPMD MPI task example

```python
from pycompss.api.mpmd_mpi import mpmd_mpi

@mpmd_mpi(runner="mpirun",
          programs=[
                dict(binary="grep", args="{{keyword}} {{in_file_1}}"),
                dict(binary="grep", args="{{keyword}} {{in_file_2}}"),
          ])
@task(in_file=FILE_IN, result={Type: FILE_OUT_STDOUT})
def std_out(keyword, in_file_1, in_file_2, result):
    pass
```

Other parameters of *@mpmd_mpi* decorator such as `working_dir`, `fail_by_exit_value`, `processes_per_node`, have the same behaviors as in *@mpi*.

I/O decorator

The *@IO* decorator is used to declare a task as an I/O task. I/O tasks exclusively perform I/O (i.e., reading or writing) and should not perform any computations.

Code 78: I/O task example

```python
from pycompss.api.IO import IO

@IO()
@task()
def io_func(text):
    fh = open("dump_file", "w")
    fh.write(text)
    fh.close()
```

The execution of I/O tasks can overlap with the execution of non-IO tasks (i.e., tasks that do not use the *@IO* decorator) if there are no dependencies between them. In addition to that, the scheduling of I/O tasks does not depend on the availability of computing units. For instance, an I/O task can be still scheduled and executed on a certain node even if all the CPUs on that node are busy executing non-I/O tasks. Hence, increasing parallelism level.

The *@IO* decorator can be also used on top of the *@mpi* decorator (*MPI decorator*) to declare a task that performs parallel I/O. Example Code 79 shows a MPI-IO task that does collective I/O with a NumPy array.

Code 79: Python MPI-IO task example.

```python
from pycompss.api.IO import IO
from pycompss.api.mpi import mpi
```

```
@IO()
@mpi(processes=4)
@task()
def mpi_io_func(text_chunks):
    from mpi4py import MPI
    import numpy as np

    fmode = MPI.MODE_WRONLY|MPI.MODE_CREATE
    fh = MPI.File.Open(MPI.COMM_WORLD, "dump_file", fmode)

    buffer = np.empty(20, dtype=np.int)
    buffer[:] = MPI.COMM_WORLD.Get_rank()

    offset = MPI.COMM_WORLD.Get_rank() * buffer.nbytes
    fh.Write_at_all(offset, buffer)

    fh.Close()
```

COMPSs decorator

The *@compss* (or @COMPSs) decorator shall be used to define that a task is going to be a COMPSs application (Code 80). It enables to have nested PyCOMPSs/COMPSs applications.

Code 80: COMPSs task example

```
from pycompss.api.compss import compss

@compss(runcompss="${RUNCOMPSS}", flags="-d",
        app_name="/path/to/simple_compss_nested.py", computing_nodes="2")
@task()
def compss_func():
    pass
```

The COMPSs application invocation can also be enriched with the flags accepted by the *runcompss* executable. Please, check execution manual for more details about the supported flags.

Multinode decorator

The *@multinode* (or @Multinode) decorator shall be used to define that a task is going to use multiple nodes (e.g. using internal parallelism) (Code 81).

Code 81: Multinode task example

```
from pycompss.api.multinode import multinode

@multinode(computing_nodes="2")
@task()
def multinode_func():
    pass
```

The only supported parameter is *computing_nodes*, used to define the number of nodes required by the task (the default value is 1). The mechanism to get the number of nodes, threads and their names to the task is through the *COMPSS_NUM_NODES*, *COMPSS_NUM_THREADS* and *COMPSS_HOSTNAMES* environment variables respectively, which are exported within the task scope by the COMPSs runtime before the task execution.

HTTP decorator

The *@http* decorator can be used for the tasks to be executed on a remote Web Service via HTTP requests. In order to create HTTP tasks, it is obligatory to define HTTP resource(s) in `resources` and `project` files (see *HTTP configuration*). Following code snippet (Code 82) is a basic HTTP task with all required parameters. At the time of execution, the runtime will search for HTTP resource from resources file which allows execution of 'service_1' and send a GET request to its 'Base URL'. Moreover, python parameters can be added to the request query as shown in the example (between double curly brackets).

Code 82: HTTP Task example.

```python
from pycompss.api.task import task
from pycompss.api.http import http


@http(service_name="service_1", request="GET",
      resource="get_length/{{message}}")
@task(returns=int)
def an_example(message):
    pass
```

For POST requests it is possible to send a parameter as the request body by adding it to the `payload` arg. In this case, payload type can also be specified ('application/json' by default). If the parameter is a FILE type, then the content of the file is read in the master and added to the request as request body.

Code 83: HTTP Task with POST request.

```python
from pycompss.api.task import task
from pycompss.api.http import http


@http(service_name="service_1", request="POST", resource="post_json/",
      payload="{{payload}}", payload_type="application/json")
@task(returns=str)
def post_with_param(payload):
    pass
```

For the cases where the response body is a JSON formatted string, PyCOMPSs' HTTP decorator allows response string formatting by defining the return values within the `produces` parameter. In the following example, the return value of the task would be extracted from 'length' key of the JSON response string:

Code 84: HTTP Task with return value to be extracted from a
JSON string.

```python
from pycompss.api.task import task
from pycompss.api.http import http



@http(service_name="service_1", request="GET",
      resource="produce_format/{{message}}",
      produces="{'length':'{{return_0}}'}")
@task(returns=int)
def an_example(message):
    pass
```

Note that if the task has multiple returns, 'return_0', 'return_1', return_2, etc. all must be defined in the `produces` string.

It is also possible to take advantages of INOUT python dicts within HTTP tasks. In this case, `updates` string can be used to update the INOUT dict:

Code 85: HTTP Task with return value to be extracted from a
JSON string.

```
@http(service_name="service_1", request="GET",
      resource="produce_format/test",
      produces="{'length':'{{return_0}}', 'child_json':{'depth_1':'one', 'message':'{{param}}
↪'}}",
      updates='{{event}}.some_key = {{param}}')
@task(event=INOUT)
def http_updates(event):
    """
    """
    pass
```

In the example above, 'some_key' key of the INOUT dict param will be updated according to the response. Please note that the `{{param}}` is defined inside `produces`. In other words, parameters that are defined inside `produces` string can be used in `updates` to update INOUT dicts.

Important: **Disclaimer:** Due to serialization limitations, with the current implementation, outputs of regular PyCOMPSs `tasks` cannot be passed as input parameters to `http` tasks.

Disclaimer: COLLECTION_* and DICTIONARY_* type of parameters are not supported within HTTP tasks. However, Python lists and dictionary objects can be used.

Reduction decorator

The *@reduction* (or @Reduction) decorator shall be used to define that a task is going to be subdivided into smaller tasks that take as input a subset of the input data (one `COLLECTION`).

The only supported parameter is *chunk_size*, used to define the size of the data that the generated tasks will get as input parameter. The data given as input to the main reduction task is subdivided into chunks of the set size.

Code 86 shows how to declare a reduction task. In detail, this application calls 10 times to `calculate_area` and appends the results into `areas` list. Then, invokes the `sum_reduction` task (that is declared as a reduction task) with the `areas` list and has `chunk_size=2`. Although it is invoked once, the COMPSs runtime splits the input data (`areas`) into chunks of 2 elements, and applies the `sum_reduction` function to them until the final result is achieved. Then, the `compss_wait_on` retrieves the final result and it is printed.

Code 86: Reduction task example

```
from pycompss.api.reduction import reduction
from pycompss.api.task import task
from pycompss.api.parameter import COLLECTION_IN
from pycompss.api.api import compss_wait_on


@task(returns=int)
def calculate_area(height, width):
    return height * width


@reduction(chunk_size="2")
@task(returns=int, areas=COLLECTION_IN)
def sum_reduction(areas):
    total_area = 0
    for area in areas:
```

(continues on next page)

```python
        total_area += area
    return total_area


def main():
    areas = []
    for i in range(10):
        areas.append(calculate_area(i, i))
    result = sum_reduction(areas)
    result = compss_wait_on(result)
    print("Result: %d" % result)


if __name__ == "__main__":
    main()
```

> **Caution:** The task decorated with `@reduction` can have multiple parameters, but **ONLY ONE** `COLLECTION_IN` **parameter**, which will be splitted into chunks to perform the reduction.

Container decorator

The `@container` (or `@Container`) decorator shall be used to define that a task is going to be executed within a container (Code 87).

Code 87: Container task example

```python
from pycompss.api.compss import container
from pycompss.api.task import task
from pycompss.api.parameter import *
from pycompss.api.api import compss_wait_on

@container(engine="DOCKER",
           image="compss/compss")
@task(returns=1, num=IN, in_str=IN, fin=FILE_IN)
def container_fun(num, in_str, fin):
    # Sample task body:
    with open(fin, "r") as fd:
        num_lines = len(fd.readlines())
    str_len = len(in_str)
    result = num * str_len * num_lines

    # You can import and use libraries available in the container

    return result

if __name__=='__main__':
    result = container_fun(5, "hello", "dataset.txt")
    result = compss_wait_on(result)
    print("result: %s" % result)
```

The *container_fun* task will be executed within the container defined in the *@container* decorator using the *docker* engine with the compss/compss *image*. This task is pure python and you can import and use any library available in the container

This feature allows to use specific containers for tasks where the library dependencies are met.

Tip: Singularity is also supported, and can be selected by setting the engine to SINGULARITY:

```
@container(engine=SINGULARITY)
```

In addition, the *@container* decorator can be placed on top of the *@binary*, *@ompss* or *@mpi* decorators. Code 88 shows how to execute the same example described in the *Binary decorator* section, but within the `compss/compss` container using docker. This will execute the binary/ompss/mpi binary within the container.

Code 88: Container binary task example

```python
from pycompss.api.compss import container
from pycompss.api.task import task
from pycompss.api.binary import binary
from pycompss.api.parameter import *

@container(engine="DOCKER",
           image="compss/compss")
@binary(binary="grep", working_dir=".")
@task(infile={Type:FILE_IN_STDIN}, result={Type:FILE_OUT_STDOUT})
def grepper():
    pass


if __name__=='__main__':
    infile = "infile.txt"
    outfile = "outfile.txt"
    grepper("Hi", infile, outfile)
```

Software decorator

The *@software* decorator is useful in order to move definitions of several PyCOMPSs decorators to a JSON file. It allows the users to 'define' their decorator definitions from an external file, which can be generated by another resource. Thus, the only supported argument is the 'config_file' that should contain the path to the JSON configuration file. Following example shows a basic usage of the @software decorator:

Code 89: Software decorator definition example.

```python
from pycompss.api.task import task
from pycompss.api.software import software

@software(config_file="example.json")
@task(returns=1)
def example():
    return "hola"
```

Configuration files can contain different key-values depending on the user's needs. Details of the configuration of the software execution can be defined in the value of the "execution" key. There the user can define the "type" of the execution and other necessary configuration parameters the *software* requires. Moreover, parameters of the PyCOMPSs *@task* can be added in the "parameters".

For example, If we wanted to move the *@task@* definition from the example above, inside the software the code and config file it would look like as follows:

Code 90: Example: task definition is inside the config file.

```python
from pycompss.api.task import task
from pycompss.api.software import software
```

(continues on next page)

```python
@software(config_file="example.json")
def example():
    return "hola"
```

Code 91: Task definition inside a software config file.

```json
{
  "execution" : {
    "type":"task"
  },
  "parameters" : {
    "returns" : 1
  }
}
```

If the user wants to define an MPI task, then "mpi" value should be set for the "type" key. Moreover, arguments of PyCOMPSs' *@mpi* decorator can be added. A basic configuration file for an MPI task would look like in the example below:

Code 92: JSON configuration file of an MPI definition.

```json
{
  "execution" : {
    "type":"mpi",
    "runner": "mpirun",
    "binary":"date",
    "working_dir": "/tmp"
    },
  "parameters" : {
    "returns" : 1
  }
}
```

As we see "runner", "processes", and "binary" are the regular parameters of @mpi decorator, and are added as part of the execution configuration. However, parameters of the *@task* definition is are in the "parameters" key.

If we wanted to combine @constraint and @mpi decorators together, the JSON file would have been extended:

Code 93: JSON configuration file of an MPI definition.

```json
{
    "execution": {
        "type":"mpi",
        "runner":"mpirun",
        "processes": 2,
        "binary":"~/app_mpi.bin",
            "args": "-d {{a}} {{b}}"
    },
    "parameters" : {
        "returns" : 1
    },
    "constraints":{
        "computing_units": 2
    }
}
```

It's also possible to add @prolog and @epilog definitions in the configuration files:

Code 94: Prolog and Epilog definitions in configuration files.

```json
{
    "execution": {
        "type":"mpi",
        "runner":"mpirun",
        "processes": 2,
        "binary":"app_mpi.bin",
            "args": "-d {{a}} {{b}}"
    },
    "parameters" : {
        "returns" : 1
    },
    "constraints":{
        "computing_units": 2
    },
    "prolog":{
        "binary":"echo",
        "args":"greetings from prolog."
    },
    "epilog":{
        "binary":"echo",
        "args":"execution finished."
    },
}
```

Next table provides more detailed information about JSON configuration files:

Key	Description
execution	(Mandatory) Contains all the software execution details such as "type" and arguments of the type's decorator.
execution.type	(Mandatory) Type of the software invocation. Supported values are 'task', 'workflow', 'mpi', 'binary', 'mpmd_mpi', 'multinode', 'http', and 'compss'.
parameters	A dictionary containing parameters regarding the "@task" definition.
prolog	Replaces the @prolog definition and expects @prolog parameters in a dictionary.
epilog	Replaces the @epilog definition and expects @epilog parameters in a dictionary.
constraints	Replaces the @constraint definition and expects @constraint parameters in a dictionary.
container	Replaces the @container definition and expects @container parameters in a dictionary.

Please check *Other task types summary* for the full list of the parameters for each decorator.

Julia decorator

The *@julia* (or *@Julia*) decorator shall be used to define that a task is going to invoke a Julia executable, which can be parallelized with Julia Parallel ClusterManagers described in the Julia documentation.

In this context, the *@task* decorator parameters will be used as the julia invocation parameters (following their order in the function definition). Since the invocation parameters can be of different nature, information on their type can be provided through the *@task* decorator.

Code 95 shows the most simple julia task definition without constraints and without parameters.

Code 95: Julia task example

```python
from pycompss.api.task import task
from pycompss.api.julia import julia


@julia(script="my_julia_app.jl")
@task()
def julia_func():
    pass
```

Code 96: my_julia_app.jl code

```julia
println("Hello world")
```

The invocation of the *julia_func* task would be equivalent to:

```
$ julia my_julia_app.jl
Hello world
```

The `@julia` decorator supports the `working_dir` parameter to define the working directory for the execution of the defined julia script.

Code 97 shows a more complex julia invocation, with parameters (x and y) and a file (that captures the standard output stream during the *mandelbrot.jl* execution) as parameters:

Code 97: Julia task example using *mandelbrot.jl* application (*julia_decorator_test.py*)

```python
from pycompss.api.task import task
from pycompss.api.julia import julia
from pycompss.api.parameter import *


@julia(script="mandelbrot.jl", working_dir=".")
@task(result={Type:FILE_OUT_STDOUT})
def julia_mandelbrot(x, y, result):
    pass

# This task definition is equivalent to the following, which is more verbose:
#
# @julia(script="mandelbrot.jl", working_dir=".")
# @task(result={Type:FILE_OUT, StdIOStream:STDOUT})
# def julia_mandelbrot(x, y, result):
#     pass

if __name__=='__main__':
    outfile = "fractal.txt"
    julia_mandelbrot(-0.05, 0.0315, outfile)
```

Code 98: Julia Mandelbrot implementation (mandelbrot.jl)

```julia
function mandelbrot(a)
    z = 0
    for i=1:50
        z = z^2 + a
    end
    return z
end

Y = parse(Float32, ARGS[1])
X = parse(Float32, ARGS[2])

for y=1.0:Y:-1.0
    for x=-2.0:X:0.5
        abs(mandelbrot(complex(x, y))) < 2 ? print("*") : print(" ")
    end
    println()
end

# Taken from: https://rosettacode.org/wiki/Mandelbrot_set#Julia
# Added X and Y command line parse.
```

The invocation of the *julia_mandelbrot* task would be equivalent to:

```
$ # julia mandelbrot.jl x y > result
$ julia mandelbrot.jl -0.05, 0.0315 > fractal.txt
```

And the final result of *fractal.txt* after executing the is:

```
$ runcompss julia_decorator_test.py
[ INFO ] Inferred PYTHON language
[ INFO ] Using default location for project file: /opt/COMPSs//Runtime/configuration/xml/
↪projects/default_project.xml
[ INFO ] Using default location for resources file: /opt/COMPSs//Runtime/configuration/xml/
↪resources/default_resources.xml
[ INFO ] Using default execution type: compss

----------------- Executing julia_decorator_test.py --------------------------

WARNING: COMPSs Properties file is null. Setting default values
[(930)    API] - Starting COMPSs Runtime v3.0.rc2210 (build 20221014-1030.
↪reba7fbb482a79b596e249b2c3b6b17509a05652a)
[(5300)    API] - Execution Finished

------------------------------------------------------------
$ cat fractal.txt




                                            **
                                          ******
                                         ********
                                          ******
                                        ******** **    *
                              ***    *****************
                              *********************** ***
```

(continues on next page)

```
                        ****************************
                      ******************************
                    ********************************
                  ************************************
        *             ************************************
    ** ***** *        ************************************
    ***********     **************************************
  ************** ****************************************
  *****************************************************
  *****************************************************
** * **********************************************************
    *******************************************************
    *****************************************************
    ************** ***************************************
    ***********     **************************************
    ** ***** *        ************************************
        *             **********************************
                  ************************************
                    *******************************
                    *****************************
                      **************************
                      *************************** ***
                  ***     ******************
                        ******** **   *
                          ******
                        ********
                          ******
                            **
```

Please note that the *keyword* parameter is a string, and it is respected as is in the invocation call. Another way of passing task parameters to julia execution command is to use `args` parameter in the julia definition. In this case, task parameters should be defined between curly braces and the full string with parameter replacements will be added to the command. In the following example, value of 'param_1' is added to the execution command after '-d' arg:

Code 99: Julia task example with args

```python
from pycompss.api.task import task
from pycompss.api.julia import julia
from pycompss.api.parameter import *


@julia(script="my_julia_app.jl", args= "-d {{param_1}}")
@task()
def julia_task(param_1):
    pass

if __name__=='__main__':
    julia_task("hello")
```

The invocation of the *julia_task* task would be equivalent to:

```
$ # julia my_julia_app.jl -d param_1
$ julia -d hello
```

Thus, PyCOMPSs can also deal with prefixes for the given parameters:

Code 100: Julia task example 4

```python
from pycompss.api.task import task
from pycompss.api.julia import julia
from pycompss.api.parameter import *


@julia(script="my_julia_app.jl")
@task(hide={Type:FILE_IN, Prefix:"--hide="}, sort={Prefix:"--sort="})
def julia_task(flag, hide, sort):
    pass


if __name__=='__main__':
    flag = '-l'
    hideFile = "fileToHide.txt"
    sort = "time"
    julia_task(flag, hideFile, sort)
```

The invocation of the *julia_task* task would be equivalent to:

```
$ # julia my_julia_app.jl -l --hide=hide --sort=sort
$ julia my_julia_app.jl -l --hide=fileToHide.txt --sort=time
```

This particular case is intended to show all the power of the *@julia* decorator in conjunction with the *@task* decorator. Please note that although the *hide* parameter is used as a prefix for the julia invocation, the *fileToHide.txt* would also be transfered to the worker (if necessary) since its type is defined as *FILE_IN*. This feature enables to build more complex julia invocations.

In addition, the `@julia` decorator also supports the `fail_by_exit_value` parameter to define the failure of the task by the exit value of the julia (Code 101). It accepts a boolean (`True` to consider the task failed if the exit value is not 0, or `False` to ignore the failure by the exit value (**default**)), or a string to determine the environment variable that defines the fail by exit value (as boolean). The default behaviour (`fail_by_exit_value=False`) allows users to receive the exit value of the julia as the task return value, and take the necessary decissions based on this value.

Code 101: Julia task example with `fail_by_exit_value`

```python
@julia(script="my_julia_app.jl", fail_by_exit_value=True)
@task()
def julia_task():
    pass
```

In addition, to all previous possibilities, a *@julia* task can also be defined with constraints. To this end, the *@constraint* decorator has to be provided on top of the *@julia* decorator:

Code 102: Julia task example using *mandelbrot.jl* application (*julia_decorator_test.py*) with constraint

```python
from pycompss.api.task import task
from pycompss.api.julia import julia
from pycompss.api.parameter import *
from pycompss.api.constraint import constraint


@constraint(computing_units="2")
@julia(script="mandelbrot.jl", working_dir=".")
@task(result={Type:FILE_OUT_STDOUT})
def julia_mandelbrot(x, y, result):
    pass
```

(continues on next page)

```
# This task definition is equivalent to the following, which is more verbose:
#
# @constraint(computing_units="2")
# @julia(script="mandelbrot.jl", working_dir=".")
# @task(result={Type:FILE_OUT, StdIOStream:STDOUT})
# def julia_mandelbrot(x, y, result):
#     pass


if __name__=='__main__':
    outfile = "fractal.txt"
    julia_mandelbrot(-0.05, 0.0315, outfile)
```

Code 102 extends the Code 97 with the *@constraint* decorator in order to define that the *julia_ mandelbrot* task requires 2 computing nodes (cores). In this scenario, the julia script (*mandelbrot.jl*) needs to implement a mechanism to exploit multiple cores.

Finally, the PyCOMPSs integration with Julia also enables to use multiple computing nodes, enabling to have two levels of parallelism (PyCOMPSs and Julia Parallel ClusterManagers) However, this feature is limited to SLURM enabled clusters (i.e. supercomputers with SLURM queuing system).

The following code snippet (Code 103) shows the definition of a Julia task that requires to be executed using 2 nodes and with 2 processes on each node (4 total processes). The julia script executed as task (Code 104) used the Julia Parallel ClusterManagers library to spawn the processes in the nodes where COMPSs runtime has enabled, and on each node and process prints its identifier and node name.

Code 103: Julia task example using multiple nodes

```
from pycompss.api.task import task
from pycompss.api.julia import julia
from pycompss.api.parameter import *
from pycompss.api.constraint import constraint
from pycompss.api.multinode import multinode


@multinode(computing_nodes="2")
@constraint(computing_units="2")
@julia(script="distributed_app.jl")
@task(result={Type:FILE_OUT_STDOUT})
def julia_distributed_app(result):
    pass


# This task definition can also be defined as follows:
#
# @constraint(computing_units="2")
# @julia(script="distributed_app.jl", computing_nodes="2")
# @task(result={Type:FILE_OUT_STDOUT})
# def julia_distributed_app(result):
#     pass


if __name__=='__main__':
    outfile = "fractal.txt"
    julia_mandelbrot(-0.05, 0.0315, outfile)
```

Code 104: Julia application using distributed parallelism (*distributed_ app.jl*)

```
using Distributed, ClusterManagers
addprocs_slurm(parse(Int, ENV["SLURM_NTASKS"]))
```

```
@everywhere using Distributed
@everywhere println(myid())
@everywhere println(gethostname())

println("Hello world")
```

Tip: If the julia script sets the number or processes based on the *SLURM_NTASKS* environment variable allows to change the number of total processes and nodes without modifying the julia script. This enables to adapt the julia script parallelism in terms of the *computing_units* and *computing_nodes* defined in the *@constraint* and *@multinode* decorators accordingly.

Other task types summary

Next tables summarizes the parameters of these decorators. Please note that 'working_dir' and 'args' ae the only decorator properties that can contain task parameters defined in curly braces.

- *Binary decorator* (**@binary**)

Parameter	Description
binary	(Mandatory) String defining the full path of the binary that must be executed.
working_dir	Full path of the binary working directory inside the COMPSs Worker.
args	Args string to be added to end of the execution command of the binary. It can contain python task parameters defined in curly braces.

- *OmpSs decorator* (**@ompss**)

Parameter	Description
binary	(Mandatory) String defining the full path of the binary that must be executed.
working_dir	Full path of the binary working directory inside the COMPSs Worker.

- *MPI decorator* (**@mpi**)

Parameter	Description
binary	String defining the full path of the binary that must be executed. Empty indicates python MPI code.
working_dir	Full path of the binary working directory inside the COMPSs Worker.
runner	(Mandatory) String defining the MPI runner command.
processes	Integer defining the number of MPI processes spawned by the task. (Default 1)
processes_per_node	Integer defining the number of co-allocated MPI processses per node. The `processes` value should be multiple of this value
args	Args string to be added to end of the execution command of the binary. It can contain python task parameters defined in curly braces.

- *MPMD MPI decorator* (**@mpmd_mpi**)

Parameter	Description
runner	(Mandatory) String defining the MPMD MPI runner command.
working_dir	Defines mpi job's working directory.
processes_-per_node	Integer defining the number of co-allocated MPI processes per node. The `processes` value should be multiple of this value
fail_by_-exit_value	If set to 'False', and `returns` value of the 'task' definition is 'int', exit code of the MPI command will be returned.
programs	List of single MPI program dictionaries where program specific parameters (`binary`, `processes`, `args`) are defined.

- *I/O decorator* (@io)
- *COMPSs decorator* (@compss)

Parameter	Description
runcompss	(Mandatory) String defining the full path of the runcompss binary that must be executed.
flags	String defining the flags needed for the runcompss execution.
app_name	(Mandatory) String defining the application that must be executed.
computing_nodes	Integer defining the number of computing nodes reserved for the COMPSs execution (only a single node is reserved by default).

- *Multinode decorator* (@multinode)

Parameter	Description
computing_nodes	Integer defining the number of computing nodes reserved for the task execution (only a single node is reserved by default).

- *HTTP decorator* (@http)

Parameter	Description
service_-name	(Mandatory) Name of the HTTP Service that included at least one HTTP resource in the resources file.
resource	(Mandatory) URL extension to be concatenated with HTTP resource's base URL.
request	(Mandatory) Type of the HTTP request (GET, POST, etc.).
produces	In case of JSON responses, produces string defines where the return value(s) is (are) stored in the retrieved JSON string.
payload	Payload string of POST requests if any.
pay-load_-type	Payload type of POST requests (e.g: 'application/json').
updates	To define INOUT parameter key to be updated with a value from HTTP response.

- *Reduction decorator* (@reduction)

Parameter	Description
chunk_size	Size of data fragments to be given as input parameter to the reduction function.

- *Container decorator* (@container)

Parameter	Description
engine	Container engine to use (e.g. DOCKER or SINGULARITY).
image	Container image to be deployed and used for the task execution.

- *Software decorator* (@software)

Parameter	Description
config_file	Path to the JSON configuration file.

- *Julia decorator* (@julia)

Parameter	Description
executor	String defining the julia binary executor (default: *julia*).
script	(Mandatory) String defining the full path of the Julia script that must be executed.
fail_by_exit_value	If set to 'False', and `returns` value of the 'task' definition is 'int', exit code of the Julia script execution will be returned.
working_dir	Full path of the julia script working directory inside the COMPSs Worker.
computing_nodes	Integer defining the number of computing nodes reserved for the task execution (default: "1" - overrides *@multinode* decorator).
args	Args string to be added to end of the execution command of the Julia script. It can contain python task parameters defined in curly braces.

In addition to the parameters that can be used within the *@task* decorator, Table 9 summarizes the *StdIOStream* parameter that can be used within the *@task* decorator for the function parameters when using the @binary, @ompss and @mpi decorators. In particular, the *StdIOStream* parameter is used to indicate that a parameter is going to be considered as a *FILE* but as a stream (e.g. $>$, $<$ and $2>$ in bash) for the @binary, @ompss and @mpi calls.

Table 9: Supported StdIOStreams for the @binary, @ompss and @mpi decorators

Parameter	Description
(default: empty)	Not a stream.
STDIN	Standard input.
STDOUT	Standard output.
STDERR	Standard error.

Moreover, there are some shorcuts that can be used for files type definition as parameters within the *@task* decorator (Table 10). It is not necessary to indicate the *Direction* nor the *StdIOStream* since it may be already be indicated with the shorcut.

Table 10: File parameters definition shortcuts

Alias	Description
COLLECTION(_IN)	Type: COLLECTION, Direction: IN
COLLECTION_IN_DELETE	Type: COLLECTION, Direction: IN_DELETE
COLLECTION_INOUT	Type: COLLECTION, Direction: INOUT
COLLECTION_OUT	Type: COLLECTION, Direction: OUT
DICTIONARY(_IN)	Type: DICTIONARY, Direction: IN
DICTIONARY_IN_DELETE	Type: DICTIONARY, Direction: IN_DELETE
DICTIONARY_INOUT	Type: DICTIONARY, Direction: INOUT
COLLECTION_FILE(_IN)	Type: COLLECTION (File), Direction: IN
COLLECTION_FILE_INOUT	Type: COLLECTION (File), Direction: INOUT
COLLECTION_FILE_OUT	Type: COLLECTION (File), Direction: OUT
FILE(_IN)_STDIN	Type: File, Direction: IN, StdIOStream: STDIN
FILE(_IN)_STDOUT	Type: File, Direction: IN, StdIOStream: STDOUT
FILE(_IN)_STDERR	Type: File, Direction: IN, StdIOStream: STDERR
FILE_OUT_STDIN	Type: File, Direction: OUT, StdIOStream: STDIN
FILE_OUT_STDOUT	Type: File, Direction: OUT, StdIOStream: STDOUT
FILE_OUT_STDERR	Type: File, Direction: OUT, StdIOStream: STDERR
FILE_INOUT_STDIN	Type: File, Direction: INOUT, StdIOStream: STDIN
FILE_INOUT_STDOUT	Type: File, Direction: INOUT, StdIOStream: STDOUT
FILE_INOUT_STDERR	Type: File, Direction: INOUT, StdIOStream: STDERR
FILE_CONCURRENT	Type: File, Direction: CONCURRENT
FILE_CONCURRENT_STDIN	Type: File, Direction: CONCURRENT, StdIOStream: STDIN
FILE_CONCURRENT_STDOUT	Type: File, Direction: CONCURRENT, StdIOStream: STDOUT
FILE_CONCURRENT_STDERR	Type: File, Direction: CONCURRENT, StdIOStream: STDERR
FILE_COMMUTATIVE	Type: File, Direction: COMMUTATIVE
FILE_COMMUTATIVE_STDIN	Type: File, Direction: COMMUTATIVE, StdIOStream: STDIN
FILE_COMMUTATIVE_STD-OUT	Type: File, Direction: COMMUTATIVE, StdIOStream: STDOUT
FILE_COMMUTATIVE_-STDERR	Type: File, Direction: COMMUTATIVE, StdIOStream: STDERR

These parameter keys, as well as the shortcuts, can be imported from the PyCOMPSs library:

```
from pycompss.api.parameter import *
```

Task Constraints

It is possible to define constraints for each task. To this end, the *@constraint* (or @Constraint) decorator followed by the desired constraints needs to be placed ON TOP of the @task decorator (Code 105).

Important: Please note the the order of *@constraint* and *@task* decorators is important.

Code 105: Constrained task example

```
from pycompss.api.task import task
from pycompss.api.constraint import constraint
from pycompss.api.parameter import INOUT

@constraint(computing_units="4")
@task(c=INOUT)
def func(a, b, c):
```

(continues on next page)

```
    c += a * b
    ...
```

This decorator enables the user to set the particular constraints for each task, such as the amount of Cores required explicitly. Alternatively, it is also possible to indicate that the value of a constraint is specified in a environment variable (Code 106).

For example:

Code 106: Constrained task with environment variable example

```
from pycompss.api.task import task
from pycompss.api.constraint import constraint
from pycompss.api.parameter import INOUT

@constraint(computing_units="4",
            app_software="numpy,scipy,gnuplot",
            memory_size="$MIN_MEM_REQ")
@task(c=INOUT)
def func(a, b, c):
    c += a * b
    ...
```

Or another example requesting a CPU core and a GPU (Code 107).

Code 107: CPU and GPU constrained task example

```
from pycompss.api.task import task
from pycompss.api.constraint import constraint

@constraint(processors=[{'processorType':'CPU', 'computingUnits':'1'},
                        {'processorType':'GPU', 'computingUnits':'1'}])
@task(returns=1)
def func(a, b, c):
    ...
    return result
```

When the task requests a GPU, COMPSs provides the information about the assigned GPU through the *COMPSS_BINDED_GPUS*, *CUDA_VISIBLE_DEVICES* and *GPU_DEVICE_ORDINAL* environment variables. This information can be gathered from the task code in order to use the GPU.

Please, take into account that in order to respect the constraints, the peculiarities of the infrastructure must be defined in the *resources.xml* file.

Supported constraints

A full description of the supported constraints can be found in Table 14.

Special constraints

There is a special constraint when **considering the COMPSs agents deployment** (*Agents Deployments*) to specify that the task MUST be executed in the node that received the task. This constraint is indicated in the *@constraint* decorator with the `is_local` argument equal a boolean (`True` or `False`) (Code 108) in addition to other constraints.

Code 108: is_local task example

```python
from pycompss.api.task import task
from pycompss.api.constraint import constraint

@constraint(is_local=True)
@task(c=INOUT)
def func(a, b, c):
    c += a * b
```

Important: The `is_local` constraint has NO effect with the default COMPSs deployment (master-workers) (*Master-Worker Deployments*).

Multiple Task Implementations

As in Java COMPSs applications, it is possible to define multiple implementations for each task. In particular, a programmer can define a task for a particular purpose, and multiple implementations for that task with the same objective, but with different constraints (e.g. specific libraries, hardware, etc). To this end, the *@implement* (or @Implement) decorator followed with the specific implementations constraints (with the *@constraint* decorator, see Section [subsubsec:constraints]) needs to be placed ON TOP of the @task decorator. Although the user only calls the task that is not decorated with the *@implement* decorator, when the application is executed in a heterogeneous distributed environment, the runtime will take into account the constraints on each implementation and will try to invoke the implementation that fulfills the constraints within each resource, keeping this management invisible to the user (Code 109).

Code 109: Multiple task implementations example

```python
from pycompss.api.implement import implement

@implement(source_class="sourcemodule", method="main_func")
@constraint(app_software="numpy")
@task(returns=list)
def myfunctionWithNumpy(list1, list2):
    # Operate with the lists using numpy
    return resultList

@task(returns=list)
def main_func(list1, list2):
    # Operate with the lists using built-int functions
    return resultList
```

Please, note that if the implementation is used to define a binary, OmpSs, MPI, COMPSs, multinode or reduction task invocation (see *Other task types*), the @implement decorator must be always on top of the decorators stack,

followed by the @constraint decorator, then the @binary/@ompss/@mpi/@compss/@multinode decorator, and finally, the @task decorator in the lowest level.

Prolog & Epilog

The *@prolog* and *@epilog* decorators are definitions of binaries to be executed before / after `task` execution on the worker. All kind of PyCOMPSs tasks can have a *@prolog* or an *@epilog*, or both at the same time. A basic usage is shown in the example below:

Important: Please note that *@prolog* and *@epilog* definitions should be on top of *@task* decorators.

Code 110: Prolog and Epilog definitions.

```python
from pycompss.api.epilog import epilog
from pycompss.api.prolog import prolog
from pycompss.api.task import task


@prolog(binary="start_some_service.bin")
@task()
def basic():
    ...
    return 1

@epilog(binary="shut_down.bin")
@task()
def basic():
    ...
    return 1
```

Both decorators have the same syntax and have 3 parameters: `binary` is the only mandatory parameter where `args` and `fail_by_exit_value` are optional. `args` describe the command line arguments of the binary. Users can also pass the task parameters as arguments. In this case, the task parameter should be surrounded by double curly braces (*"{{" and "}}")* in the 'args' string. These parameters can be results of previous tasks and PyCOMPSs will handle data dependencies between tasks:

Important: Task parameters used in 'args' strings can be type of primitive types such as int, float, string, and boolean.

Code 111: Task parameter in Prolog definition.

```python
from pycompss.api.prolog import prolog
from pycompss.api.task import task

@prolog(binary="mkdir", args="{{param_1}}")
@task()
def task_1(param_1):
    ...
    return 1

# call to the task function
task_1("/home/dir_to_be_created_before_task_exec")
```

`fail_by_exit_value` is used to indicate the behaviour when the prolog or epilog returns an exit value different than zero. Users can set the `fail_by_exit_value` to *True*, if they want to consider the exit value as a task

failure. If set to *False*, failure of the prolog will be ignored and task execution will start as usual. The same rule applies for the `epilog` as well. Default value of 'fail_by_exit_value' is *True* for Prolog and *False* for Epilog:

Code 112: Prolog & Epilog with 'fail_by_exit_value'.

```python
from pycompss.api.epilog import epilog
from pycompss.api.prolog import prolog
from pycompss.api.task import task

@prolog(binary="mkdir", args="-p {{sandbox_path}}", fail_by_exit_value=True)
@epilog(binary="rm", args="-r {{sandbox_path}}", fail_by_exit_value=False)
@task()
def task_2(sandbox_path):
    ...
    return 1


# call to the task function
task_2("/tmp/my_task_sandbox")
```

In the example above, if creation of the 'sandbox_path' fails, the task execution won't start at all and task will be considered as failed. However, if removing the sandbox is not crucial and can be ignored, `fail_by_exit_value` in the Epilog can be set to *False*.

Data Transformation

The *@data_transformation* (or just *@dt*) decorator is used for the execution of a data transformation function that should be applied on a given `PyCOMPSs task` parameter. It means, by specifying the parameter name and a python function, users can assure that the parameter will go through transformation process by the given function. Then the result of the data transformation function will be used in the task instead of the initial value of the parameter.

Data transformation decorator has a simple order for the definition. The first argument of the decorator is a string name of the parameter we want to transform. The second argument is the data transformation function (NOT as a string, but actual reference) that expects at least one input which will the transformation will be applied to. If the transformation function needs more parameters, they can be added to the *@dt* definition as `kwargs`.

Code 113: Arguments list of the data transformation decorator.

```python
@dt("<parameter_name>", "<dt_function>", "<kwargs_of_dt_function>")
@task()
def task_func(...):
    ...
```

Important: Please note that data transformation definitions should be on top of the *@task* (or *@software*) decorator.

Adding data transformation on top of the `@task` decorator allows the PyCOMPSs Runtime generate an intermediate task. This task method applies the given DT to the given input and the output is sent to the *original* task as the input. Following code snippet is an example of basic usage of the *@dt* decorator:

Code 114: An example of data transformation decorator.

```python
from pycompss.api.data_transformation import dt
from pycompss.api.task import task
from pycompss.api.api import compss_wait_on
```

```python
def append_dt(A):
    A.append("from_dt")
    return A


@dt("A", append_dt)
@task()
def task_func(A):
    A.append("from_task_itself")
    return A


def main():
    A = ["initial_value"]
    A = compss_wait_on(task_func(A))
    print(A)
```

When the `main` function called, a new list called `A` will be initialized with one element and will be sent to a PyCOMPSs task called `task_func`. Then, a call to "task_func" method results in generation of 2 tasks by the PyCOMPSs Runtime. The first task accepts the initial list and adds "from_dt" element to it. Then the modified list is passed to the "task_func" defined by the user and "from_task_itself" is added to the list. As a result, the output of code above is a list with 3 elements including those were added by the data transformation and the "task_func" tasks.

If the user wants to use a workflow as a data transformation function and thus avoid the intermediate task creation, PyCOMPSs provides the `is_workflow` argument to do so (by default *False*). This gives the flexibility of importing workflow from different libraries.

It is also possible to define multiple data transformation functions for the same parameter, as well as for the multiple parameters from the same task. In both cases each data transformation with "is_workflow=False" will take place in a different task:

Code 115: Example: multiple data transformations for a single task method.

```python
from pycompss.api.data_transformation import dt
from pycompss.api.task import task
from pycompss.api.api import compss_wait_on


@task()
def bb(A):
    A.append("from_bb")
    return A


@task()
def aa(A):
    A.append("from_aa")
    return A


# calls 2 @task functions for a given input
def workflow_dt(A):
    return aa(bb(A))


# regular python task that appends a given value to the input list
def appender_w_param(a_list, item):
    a_list.append(item)
    return a_list
```

```
@dt("A", appender_w_param, item="dt_no_workflow")
@dt("A", workflow_dt, is_workflow=True)
@dt("B", appender_w_param, item="dt_no_workflow")
@dt("B", workflow_dt, is_workflow=True)
@task()
def task_func(A, B):
    A.append("task itself")
    B.append("task itself")
    return A, B
```

In the example above, input parameter A is meant to be modified 2 times sequentially: first, "appender_w_param" function is called within a separate task and its output is sent to the next data transformation which is a *workflow*. The "workflow_dt" function consists of 2 PyCOMPSs tasks executed one after another. Only applying all these transformations to the initial value of *A*, it's passed to the "task_func" as the input. The same scenario applies for the parameter B.

PyCOMPSs API also provides Data Transformation Object class which gives the flexibility of the data transformation definitions. Any task function can be decorated with an empty **@dt** and simply by passing *DTO*(s) as a task parameter the user can achieve the same behaviour. Same as the decorator itself, DTO accepts the arguments in the same order (*"<parameter_name>", "<dt_function>", "<kwargs_of_dt_function>"*). A list of DTO objects is also accepted for the same or various parameters:

Code 116: Data Transformation Object example.

```
from pycompss.api.data_transformation import dto
from pycompss.api.data_transformation import dt
from pycompss.api.task import task
from pycompss.api.api import compss_wait_on


@dt()
@task()
def dto_basic(A, B):
    A.append("from_task")
    B.append("from_task")
    return A

def appender(a_list):
    a_list.append("from_dt")
    return a_list

def dto_example(self):
    A = ["initial"]
    B = ["initial_B"]

    # create Data Transformation Objects
    dt_1 = dto("A", appender)
    dt_2 = dto("B", appender, is_workflow=False)

    # send DT Objects to the task function as input
    A = cwo(dto_basic(A, B, dt=[dt_1, dt_2]))
```

4.2.1.2 API

PyCOMPSs provides an API for data synchronization and other functionalities, such as task group definition and automatic function parameter synchronization (local decorator).

Synchronization

The main program of the application is a sequential code that contains calls to the selected tasks. In addition, when synchronizing for task data from the main program, there exist six API functions that can be invoked:

compss_open(file_name, mode='r') Similar to the Python *open()* call. It synchronizes for the last version of file *file_name* and returns the file descriptor for that synchronized file. It can have an optional parameter *mode*, which defaults to *'r'*, containing the mode in which the file will be opened (the open modes are analogous to those of Python *open()*).

compss_wait_on_file(*file_name) Synchronizes for the last version of the file/s specified by *file_name*. Returns True if success (False otherwise).

compss_wait_on_directory(*directory_name) Synchronizes for the last version of the directory/ies specified by *directory_name*. Returns True if success (False otherwise).

compss_barrier(no_more_tasks=False) Performs a explicit synchronization, but does not return any object. The use of *compss_barrier()* forces to wait for all tasks that have been submitted before the *compss_barrier()* is called. When all tasks submitted before the *compss_barrier()* have finished, the execution continues. The *no_more_tasks* is used to specify if no more tasks are going to be submitted after the *compss_barrier()*.

compss_barrier_group(group_name) Performs a explicit synchronization over the tasks that belong to the group *group_name*, but does not return any object. The use of *compss_barrier_group()* forces to wait for all tasks that belong to the given group submitted before the *compss_barrier_group()* is called. When all group tasks submitted before the *compss_barrier_group()* have finished, the execution continues. See *Task Groups* for more information about task groups.

compss_wait_on(*obj, mode="r" | "rw") Synchronizes for the last version of object/s specifed by *obj* and returns the synchronized object. It can have an optional string parameter *mode*, which defaults to *rw*, that indicates whether the main program will modify the returned object. It is possible to wait on a list of objects. In this particular case, it will synchronize all future objects contained in the list recursively.

To illustrate the use of the aforementioned API functions, the following example (Code 117) first invokes a task *func* that writes a file, which is later synchronized by calling *compss_open()*. Later in the program, an object of class *MyClass* is created and a task method *method* that modifies the object is invoked on it; the object is then synchronized with *compss_wait_on*, so that it can be used in the main program from that point on.

Then, a loop calls again ten times to *func* task. Afterwards, the *compss_barrier()* call performs a synchronization, and the execution of the main user code will not continue until the ten *func* tasks have finished. This call does not retrieve any information.

Code 117: PyCOMPSs Synchronization API functions usage

```python
from pycompss.api.api import compss_open
from pycompss.api.api import compss_wait_on
from pycompss.api.api import compss_wait_on_file
from pycompss.api.api import compss_wait_on_directory
from pycompss.api.api import compss_barrier

if __name__=='__main__':
    my_file = 'file.txt'
    func(my_file)
    fd = compss_open(my_file)
    ...

    my_file2 = 'file2.txt'
    func(my_file2)
    compss_wait_on_file(my_file2)
```

(continues on next page)

```
    ...

    my_directory = '/tmp/data'
    func_dir(my_directory)
    compss_wait_on_directory(my_directory)
    ...

    my_obj2 = MyClass()
    my_obj2.method()
    my_obj2 = compss_wait_on(my_obj2)
    ...

    for i in range(10):
        func(str(i) + my_file)
    compss_barrier()
    ...
```

The corresponding task definition for the example above would be (Code 118):

Code 118: PyCOMPSs Synchronization API usage tasks

```
@task(f=FILE_OUT)
def func(f):
    ...


class MyClass(object):
    ...

    @task()
    def method(self):
        ... # self is modified here
```

Tip: It is possible to synchronize a list of objects. This is particularly useful when the programmer expect to synchronize more than one elements (using the *compss_wait_on* function) (Code 119). This feature also works with dictionaries, where the value of each entry is synchronized. In addition, if the structure synchronized is a combination of lists and dictionaries, the *compss_wait_on* will look for all objects to be synchronized in the whole structure.

Code 119: Synchronization of a list of objects

```
if __name__=='__main__':
    # l is a list of objects where some/all of them may be future objects
    l = []
    for i in range(10):
        l.append(ret_func())

    ...

    l = compss_wait_on(l)
```

Important: In order to make the COMPSs Python binding function correctly, the programmer **should not use relative imports** in the code. Relative imports can lead to ambiguous code and they are discouraged in Python, as explained in: http://docs.python.org/2/faq/programming.html# what-are-the-best-practices-for-using-import-in-a-module

Local Decorator

Besides the synchronization API functions, the programmer has also a decorator for automatic function parameters synchronization at his disposal. The *@local* decorator can be placed over functions that are not decorated as tasks, but that may receive results from tasks (Code 120). In this case, the *@local* decorator synchronizes the necessary parameters in order to continue with the function execution without the need of using explicitly the *compss_- wait_on* call for each parameter.

Code 120: @local decorator example

```python
from pycompss.api.task import task
from pycompss.api.api import compss_wait_on
from pycompss.api.parameter import INOUT
from pycompss.api.local import local


@task(v=INOUT)
def append_three_ones(v):
    v += [1, 1, 1]


@local
def scale_vector(v, k):
    return [k*x for x in v]


if __name__=='__main__':
    v = [1,2,3]
    append_three_ones(v)
    # v is automatically synchronized when calling the scale_vector function.
    w = scale_vector(v, 2)
```

File/Object deletion

PyCOMPSs also provides two functions within its API for object/file deletion. These calls allow the runtime to clean the infrastructure explicitly, but the deletion of the objects/files will be performed as soon as the objects/files dependencies are released.

compss_delete_file(*file_name) Notifies the runtime to delete a file/s.
compss_delete_object(*object) Notifies the runtime to delete all the associated files to a given object/s.

The following example (Code 121) illustrates the use of the aforementioned API functions.

Code 121: PyCOMPSs delete API functions usage

```python
from pycompss.api.api import compss_delete_file
from pycompss.api.api import compss_delete_object


if __name__=='__main__':
    my_file = 'file.txt'
    func(my_file)
    compss_delete_file(my_file)
    ...

    my_obj = MyClass()
    my_obj.method()
    compss_delete_object(my_obj)
    ...
```

The corresponding task definition for the example above would be (Code 122):

Code 122: PyCOMPSs delete API usage tasks

```python
@task(f=FILE_OUT)
def func(f):
    ...


class MyClass(object):
    ...

    @task()
    def method(self):
        ... # self is modified here
```

Task Groups

COMPSs also enables to specify task groups. To this end, COMPSs provides the *TaskGroup* context (Code 123) which can be tuned with the group name, and a second parameter (boolean) to perform an implicit barrier for the whole group. Users can also define task groups within task groups.

TaskGroup(group_name, implicit_barrier=True) Python context to define a group of tasks. All tasks submitted within the context will belong to *group_name* context and are sensitive to wait for them while the rest are being executed. Tasks groups are depicted within a box into the generated task dependency graph.

Code 123: PyCOMPSs Task group definiton

```python
from pycompss.api.task import task
from pycompss.api.api import TaskGroup
from pycompss.api.api import compss_barrier_group

@task()
def func1():
    ...

@task()
def func2():
    ...

def test_taskgroup():
    # Creation of group
    with TaskGroup('Group1', False):
        for i in range(NUM_TASKS):
            func1()
            func2()
        ...
    ...
    compss_barrier_group('Group1')
    ...

if __name__=='__main__':
    test_taskgroup()
```

Other

PyCOMPSs also provides other function within its API to check if a file exists.

compss_file_exists(*file_name) Checks if a file or files exist. If it does not exist, the function checks if the file has been accessed before by calling the runtime.

Code 124 illustrates its usage.

Code 124: PyCOMPSs API file exists usage

```python
from pycompss.api.api import compss_file_exists

if __name__=='__main__':
    my_file = 'file.txt'
    func(my_file)
    if compss_file_exists(my_file):
        print("Exists")
    else:
        print("Not exists")
    ...
```

The corresponding task definition for the example above would be (Code 125):

Code 125: PyCOMPSs delete API usage tasks

```python
@task(f=FILE_OUT)
def func(f):
    ...
```

API Summary

Finally, Table 11 summarizes the API functions to be used in the main program of a COMPSs Python application.

Table 11: COMPSs Python API functions

Type	API Function	Description
Synchroniza-tion	compss_open(file_name, mode='r')	Synchronizes for the last version of a file and returns its file descriptor.
	compss_wait_on_file(*file_-name)	Synchronizes for the last version of the specified file/s.
	compss_wait_on_direc-tory(*directory_name)	Synchronizes for the last version of the specified direc-tory/ies.
	compss_barrier(no_more_-tasks=False)	Wait for all tasks submitted before the barrier.
	compss_barrier_group(group_-name)	Wait for all tasks that belong to *group_name* group sub-mitted before the barrier.
	compss_wait_on(*obj, mode="r" | "rw")	Synchronizes for the last version of an object (or a list of objects) and returns it.
File/Object deletion	compss_delete_file(*file_name)	Notifies the runtime to remove the given file/s.
	compss_delete_object(*object)	Notifies the runtime to delete the associated file to the object/s.
Task Groups	TaskGroup(group_name, im-plicit_barrier=True)	Context to define a group of tasks. *implicit_barrier* forces waiting on context exit.
Other	compss_file_exists(*file_name)	Check if a file or files exist.

4.2.1.3 Failures and Exceptions

COMPSs is able to deal with failures and exceptions raised during the execution of the applications. In this case, if a user/python defined exception happens, the user can choose the task behaviour using the *on_failure* argument within the *@task* decorator.

The possible values are:
- **'RETRY'** (Default): The task is executed twice in the same worker and a different worker.
- **'CANCEL_SUCCESSORS'**: All successors of this task are canceled.
- **'FAIL'**: The task failure produces a failure of the whole application.
- **'IGNORE'**: The task failure is ignored and the output parameters are set with empty values.

A part from failures, COMPSs can also manage blocked tasks executions. Users can use the *time_out* property in the task definition to indicate the maximum duration of a task. If the task execution takes more seconds than the specified in the property. The task will be considered failed. This property can be combined with the *on_failure* mechanism.

Code 126: Task failures example

```python
from pycompss.api.task import task


@task(time_out=60, on_failure='IGNORE')
def foo(v):
    ...
```

Tip: The *on_failure* behaviour can also be defined with the `@on_failure` decorator placed over the `@task` decorator, which provides more options. For example:

Code 127: Task failures example with @on_failure decorator

```python
from pycompss.api.task import task
from pycompss.api.on_failure import on_failure
from pycompss.api.parameter import INOUT

from myclass import generate_empty   # private function that generates empty object

@on_failure(management='IGNORE', returns=0, w=generate_empty())
@task(time_out=60, w=INOUT, returns=int)
def foo(v, w):
    ...
```

This example depicts a task named `foo` that has two parameters (`v` (IN) and `w` (INOUT)) and has a timeout of 60 seconds. If the timeout is reached or an exception is thrown, the task will be considered as failed, and the management action defined in the `@on_failure` decorator applied, which in this example is to ignore the failure and continue. However, when continuing with the execution, the `foo` task should have produced a return element and modifies the `w` parameter. Consequently, the return and `w` values when the task fails are defined in the `@on__failure` decorator. The return value will be 0 when the task fails, and `w` will contain the object produced by `generate_empty` function.

COMPSs provides an special exception (`COMPSsException`) that the user can raise when necessary and can be catched in the main code for user defined behaviour management. Code 128 shows an example of *COMPSsException* raising. In this case, the group definition is blocking, and waits for all task groups to finish. If a task of the group raises a *COMPSsException* it will be captured by the runtime. It will react to it by canceling the running and pending tasks of the group and raising the COMPSsException to enable the execution except clause. Consequently, the *COMPSsException* must be combined with task groups.

In addition, the tasks which belong to the group will be affected by the *on_failure* value defined in the *@task* decorator.

Code 128: COMPSs Exception with task group example

```python
from pycompss.api.task import task
from pycompss.api.exceptions import COMPSsException
from pycompss.api.api import TaskGroup


@task()
def foo(v):
    ...
    if v == 8:
        raise COMPSsException("8 found!")


...


if __name__=='__main__':
    try:
        with TaskGroup('exceptionGroup1'):
            for i in range(10):
                foo(i)
    except COMPSsException:
        ...  # React to the exception (maybe calling other tasks or with other parameters)
```

It is possible to use a non-blocking task group for asynchronous behaviour (see Code 129). In this case, the *try-except* can be defined later in the code surrounding the *compss_barrier_group*, enabling to check exception from the defined groups without retrieving data while other tasks are being executed.

Code 129: Asynchronous COMPSs Exception with task group example

```python
from pycompss.api.task import task
from pycompss.api.api import TaskGroup
from pycompss.api.api import compss_barrier_group


@task()
def foo1():
    ...


@task()
def foo2():
    ...


def test_taskgroup():
    # Creation of group
    for i in range(10):
        with TaskGroup('Group' + str(i), False):
            for i in range(NUM_TASKS):
                foo1()
                foo2()
            ...
    for i in range(10):
        try:
            compss_barrier_group('Group' + str(i))
        except COMPSsException:
            ...  # React to the exception (maybe calling other tasks or with other parameters)
    ...


if __name__=='__main__':
```

```
    test_taskgroup()
```

Important: To ensure the COMPSs Exception is catched, they must be always combined with TaskGroups.

4.2.1.4 Integration with Numba

PyCOMPSs can also be used with Numba. Numba (http://numba.pydata.org/) is an Open Source JIT compiler for Python which provides a set of decorators and functionalities to translate Python functions to optimized machine code.

Basic usage

PyCOMPSs' tasks can be decorated with Numba's `@jit`/`@njit` decorator (with the appropiate parameters) just below the @task decorator in order to apply Numba to the task.

```python
from pycompss.api.task import task      # Import @task decorator
from numba import jit

@task(returns=1)
@jit()
def numba_func(a, b):
    ...
```

The task will be optimized by Numba within the worker node, enabling COMPSs to use the most efficient implementation of the task (and exploiting the compilation cache – any task that has already been compiled does not need to be recompiled in subsequent invocations).

Advanced usage

PyCOMPSs can be also used in conjuntion with the Numba's `@vectorize`, `@guvectorize`, `@stencil` and `@cfunc`. But since these decorators do not preserve the original argument specification of the original function, their usage is done through the *numba* parameter withih the `@task` decorator. The *numba* parameter accepts:

- **Boolean:** `True`: Applies *jit* to the function.
- **Dictionary{k, v}:** Applies *jit* with the dictionary parameters to the function (allows to specify specific jit parameters (e.g. `nopython=True`)).
- **String:**
    - `"jit"`: Applies *jit* to the function.
    - `"njit"`: Applies *jit* with *nopython=True* to the function.
    - `"generated_jit"`: Applies *generated_jit* to the function.
    - `"vectorize"`: Applies *vectorize* to the function. Needs some extra flags in the *@task* decorator:
        * *numba_signature*: String with the *vectorize* signature.
    - `"guvectorize"`: Applies *guvectorize* to the function. Needs some extra flags in the *@task* decorator:
        * *numba_signature*: String with the *guvectorize* signature.
        * *numba_declaration*: String with the *guvectorize* declaration.
    - `"stencil"`: Applies *stencil* to the function.
    - `"cfunc"`: Applies *cfunc* to the function. Needs some extra flags in the *@task* decorator:
        * *numba_signature*: String with the *cfunc* signature.

Moreover, the *@task* decorator also allows to define specific flags for the *jit*, *njit*, *generated_jit*, *vectorize*, *guvectorize* and *cfunc* functionalities with the *numba_flags* hint. This hint is used to declare a dictionary with the flags expected to use with these numba functionalities. The default flag included by PyCOMPSs is the `cache=True` in order to exploit the function caching of Numba across tasks.

For example, to apply Numba *jit* to a task:

```
from pycompss.api.task import task


@task(numba='jit')   # Aternatively: @task(numba=True)
def jit_func(a, b):
    ...
```

And if the developer wants to use specific flags with *jit* (e.g. `parallel=True`), the *numba_flags* must be defined with a dictionary where the key is the numba flag name, and the value, the numba flag value to use):

```
from pycompss.api.task import task


@task(numba='jit', numba_flags={'parallel':True})
def jit_func(a, b):
    ...
```

Other Numba's functionalities require the specification of the function signature and declaration. In the next example a task that will use the *vectorize* with three parameters and a specific flag to target the CPU is shown:

```
from pycompss.api.task import task


@task(returns=1,
      numba='vectorize',
      numba_signature=['float32(float32, float32, float32)'],
      numba_flags={'target':'cpu'})
def vectorize_task(a, b, c):
    return a * b * c
```

Using Numba with GPUs

In addition, Numba is also able to optimize python code for GPUs that can be used within PyCOMPSs' tasks. *Task using Numba and a GPU* shows an example of a task that performsa matrix multiplication in GPU (code from Numba documentation).

The `main` function creates the input and output matrices, and invokes the `do_matmul` task which has a constraint of one CPU and one GPU. This task first transfers the necessary data to the GPU using Numba's `cuda` module, then invokes the `matmul` function (that is decorated with the Numba's `@cuda.jit`). When the execution in the GPU of the ``matmul finishes, the result is transfered to the cpu with the `copy_to_host` function and the task result is returned.

Code 130: Task using Numba and a GPU

```
import math
from numba import cuda, float64
import numpy as np
from pycompss.api.task import task
from pycompss.api.api import compss_wait_on
from pycompss.api.constraint import constraint


TPB = 16


@cuda.jit
def matmul(A, B, C):
    """Perform square matrix multiplication of C = A * B
    """
    i, j = cuda.grid(2)
```

(continues on next page)

```python
    if i < C.shape[0] and j < C.shape[1]:
        tmp = 0.
        for k in range(A.shape[1]):
            tmp += A[i, k] * B[k, j]
        C[i, j] = tmp

@constraint(processors=[{'ProcessorType':'CPU', 'ComputingUnits':'1'},
                        {'ProcessorType':'GPU', 'ComputingUnits':'1'}])
@task(returns=1)
def do_matmul(a, b, c):
    gpu_a = cuda.to_device(a)
    gpu_b = cuda.to_device(b)
    gpu_c = cuda.to_device(c)

    threadsperblock = (TPB, TPB)
    blockspergrid_x = math.ceil(gpu_c.shape[0] / threadsperblock[0])
    blockspergrid_y = math.ceil(gpu_c.shape[1] / threadsperblock[1])
    blockspergrid = (blockspergrid_x, blockspergrid_y)

    matmul[blockspergrid, threadsperblock](gpu_a, gpu_b, gpu_c)
    c = gpu_c.copy_to_host()
    return c

def main():
    a = np.random.uniform(1, 2, (4, 4))
    b = np.random.uniform(1, 2, (4, 4))
    c = np.zeros((4, 4))

    result = do_matmul(a, b, c)
    result = compss_wait_on(result)

    print("a: \n %s" % str(a))
    print("b: \n %s" % str(b))
    print("Result: \n %s" % str(result))

    print("Verification result: ")
    print(a @ b)


if __name__=="__main__":
    main()
```

> **Caution:** The function compiled with Numba for GPU can not be a task since the step to transfer the data to the GPU and backwards needs to be explicitly performed by the user.
>
> For this reason, the appropiate structure is composed by a task that has the necessary constraints, deals with the data movements and invokes the function compiled with Numba for GPU.
>
> The main application can then invoke the task.

Important: In order to run with GPUs in local machine, you need to define the available GPUs in the `project.xml` file.

As example, the following `project.xml` and `resources.xml` shall be used with the `--project` and `--resources` correspondingly:

- `project.xml`
- `resources.xml`

More details about Numba and the specification of the signature, declaration and flags can be found in the Numba's webpage (http://numba.pydata.org/).

4.2.2 Application Execution

The next subsections describe how to execute applications with the COMPSs Python binding.

4.2.2.1 Environment

The following environment variables must be defined before executing a COMPSs Python application:

JAVA_HOME Java JDK installation directory (e.g. `/usr/lib/jvm/java-8-openjdk/`)

4.2.2.2 Command

In order to run a Python application with COMPSs, the `runcompss` script can be used, like for Java and C/C++ applications. An example of an invocation of the script is:

```
compss@bsc:~$ runcompss \
              --lang=python \
              --pythonpath=$TEST_DIR \
              $TEST_DIR/application.py arg1 arg2
```

Or alternatively, use the `pycompss` module:

```
compss@bsc:~$ python -m pycompss \
              --pythonpath=$TEST_DIR \
              $TEST_DIR/application.py arg1 arg2
```

Tip: The `runcompss` command is able to detect the application language. Consequently, the `--lang=python` is not mandatory.

Tip: The `--pythonpath` flag enables the user to add directories to the `PYTHONPATH` environment variable and export them into the workers, so that the tasks can resolve successfully its imports.

Tip: PyCOMPSs applications can also be launched without parallelization (as a common python script) by avoiding the `-m pycompss` and its flags when using `python`:

```
compss@bsc:~$ python $TEST_DIR/application.py arg1 arg2
```

The main limitation is that the application must only contain `@task`, `@binary` and/or `@mpi` decorators and Py-COMPSs needs to be installed.

For full description about the options available for the runcompss command please check the *Executing COMPSs applications* Section.

4.2.3 Integration with Jupyter notebook

PyCOMPSs can also be used within Jupyter notebooks. This feature allows users to develop and run their PyCOMPSs applications in a Jupyter notebook, where it is possible to modify the code during the execution and experience an interactive behaviour.

4.2.3.1 Environment Variables

The following libraries must be present in the appropiate environment variables in order to enable PyCOMPSs within Jupyter notebook:

PYTHONPATH The path where PyCOMPSs is installed (e.g. `/opt/COMPSs/Bindings/python/`). Please, note that the path contains the folder `2` and/or `3`. This is due to the fact that PyCOMPSs is able to choose the appropiate one depending on the kernel used with jupyter.

LD_LIBRARY_PATH The path where the `libbindings-commons.so` library is located (e.g. `<COMPSS_-INSTALLATION_PATH>/Bindings/bindings-common/lib/`) and the path where the `libjvm.so` library is located (e.g. `/usr/lib/jvm/java-8-openjdk/jre/lib/amd64/server/`).

4.2.3.2 API calls

In this case, the user is responsible of **starting** and **stopping** the COMPSs runtime during the jupyter notebook execution. To this end, PyCOMPSs provides a module with two main API calls: one for starting the COMPSs runtime, and another for stopping it.

This module can be imported from the *pycompss* library:

```python
import pycompss.interactive as ipycompss
```

And contains two main functions: *start* and *stop*. These functions can then be invoked as follows for the COMPSs runtime deployment with default parameters:

```python
# Previous user code/cells

import pycompss.interactive as ipycompss
ipycompss.start()

# User code/cells that can benefit from PyCOMPSs

ipycompss.stop()

# Subsequent code/cells
```

Between the *start* and *stop* function calls, the user can write its own python code including PyCOMPSs imports, decorators and synchronization calls described in the *Programming Model* Section. The code can be splitted into multiple cells.

The *start* and *stop* functions accept parameters in order to customize the COMPSs runtime (such as the flags that can be selected with the `runcompss` command). Table 12 summarizes the accepted parameters of the *start* function. Table 13 summarizes the accepted parameters of the *stop* function.

Parameter Name	Parameter Type	Description
log_level	String	Log level Options: `"off"`, `"info"` and `"debug"`. (Default: `"off"`)
debug	Boolean	COMPSs runtime debug (Default: `False`) (overrides log level)
o_c	Boolean	Object conversion to string when possible (Default: `False`)
graph	Boolean	Task dependency graph generation (Default: `False`)
trace	Boolean	Paraver trace generation (Default: `False`)

Parameter Name	Parameter Type	Description
monitor	Integer	Monitor refresh rate (Default: `None` - Monitoring disabled)
project_xml	String	Path to the project XML file (Default: `"$COMPSS/Runtime/configura`
resources_xml	String	Path to the resources XML file (Default: `"$COMPSs/Runtime/configu`
summary	Boolean	Show summary at the end of the execution (Default: `False`)
storage_impl	String	Path to an storage implementation (Default: `None`)
storage_conf	String	Storage configuration file path (Default: `None`)
task_count	Integer	Number of task definitions (Default: `50`)
app_name	String	Application name (Default: `"Interactive"`)
uuid	String	Application uuid (Default: `None` - Will be random)
base_log_dir	String	Base directory to store COMPSs log files (a .COMPSs/ folder will be
specific_log_dir	String	Use a specific directory to store COMPSs log files (the folder MUST e
extrae_cfg	String	Sets a custom extrae config file. Must be in a shared disk between all
comm	String	Class that implements the adaptor for communications. Supported ad
conn	String	Class that implements the runtime connector for the cloud. Supported
master_name	String	Hostname of the node to run the COMPSs master (Default: `""`)
master_port	String	Port to run the COMPSs master communications (Only for NIO adap
scheduler	String	Class that implements the Scheduler for COMPSs. Supported schedul
jvm_workers	String	Extra options for the COMPSs Workers JVMs. Each option separed b
cpu_affinity	String	Sets the CPU affinity for the workers. Supported options: `"disabled"`
gpu_affinity	String	Sets the GPU affinity for the workers. Supported options: `"disabled"`
profile_input	String	Path to the file which stores the input application profile (Default: `""`
profile_output	String	Path to the file to store the application profile at the end of the execu
scheduler_config	String	Path to the file which contains the scheduler configuration (Default: `"`
external_adaptation	Boolean	Enable external adaptation (this option will disable the Resource Opti
propatage_virtual_environment	Boolean	Propagate the master virtual environment to the workers (Default: `Fa`
verbose	Boolean	Verbose mode (Default: `False`)

Table 13: PyCOMPSs **stop** function for Jupyter notebook

Parameter Name	Parameter Type	Description
sync	Boolean	Synchronize the objects left on the user scope. (Default: `False`)

The following code snippet shows how to start a COMPSs runtime with tracing and graph generation enabled (with *trace* and *graph* parameters), as well as enabling the monitor with a refresh rate of 2 seconds (with the *monitor* parameter). It also synchronizes all remaining objects in the scope with the *sync* parameter when invoking the *stop* function.

```python
# Previous user code

import pycompss.interactive as ipycompss
ipycompss.start(graph=True, trace=True, monitor=2000)

# User code that can benefit from PyCOMPSs

ipycompss.stop(sync=True)

# Subsequent code
```

Attention: Once the COMPSs runtime has been stopped it, the value of the variables that have not been synchronized will be lost.

4.2.3.3 Notebook execution

The application can be executed as a common Jupyter notebook by steps or the whole application.

Important: A message showing the failed task/s will pop up if an exception within them happens.

This pop up message will also allow you to continue the execution without PyCOMPSs, or to restart the COMPSs runtime. Please, note that in the case of COMPSs restart, the tracking of some objects may be lost (will need to be recomputed).

More information on the Notebook execution can be found in the Execution Environments *Jupyter Notebook* Section.

4.2.3.4 Notebook example

Sample notebooks can be found in the *PyCOMPSs Notebooks* Section.

4.2.4 Integration with emcee

PyCOMPSs can also be used with emcee in order to enable its execution in distributed environments.

4.2.4.1 Usage

Enabling *emcee* with PyCOMPSs is easy. Assuming that you have *emcee* and COMPSs installed, there are two requirements:

1. Define the sampling function as task
2. Import the PyCOMPSs map module (`from pycompss.functions import map as pycompss_pool`) and use it in the EnsembleSampler pool parameter.

4.2.4.2 Sample Application

The following code (Code 131) shows how to enable *emcee* applications with PyCOMPSs, highlighting the modifications required.

> Code 131: emcee with PyCOMPSs application example (sampling_pycompss.py)

```python
import time
import numpy as np
import emcee
from pycompss.api.task import task
from pycompss.functions import map as pycompss_pool


def execution_params():
    """Define execution parameters."""
    np.random.seed(42)
    initial = np.random.randn(32, 5)
    nwalkers, ndim = initial.shape
    nsteps = 10
    return initial, nwalkers, ndim, nsteps


@task(returns=1)
```

(continues on next page)

```python
def log_prob(theta):
    """Sampling function to apply."""
    time.sleep(0.2)  # Computation load simulation
    return -0.5 * np.sum(theta**2)


def emcee_pycompss(params):
    """emcee usage with PyCOMPSs."""
    initial, nwalkers, ndim, nsteps = params
    sampler = emcee.EnsembleSampler(nwalkers, ndim, log_prob, pool=pycompss_pool)
    start = time.time()
    result = sampler.run_mcmc(initial, nsteps, progress=True)
    end = time.time()
    print("PyCOMPSs took {0:.1f} seconds".format(end - start))
    return result

if __name__ == "__main__":
    params = execution_params()
    result_pycompss = emcee_pycompss(params)
```

Tip: The integration is not limited to its usage with the `pycompss_pool`. It is possible to define more tasks and invoke them from the `emcee_pycompss` function in order to parallelize any preprocessing of the `initial` data or any postprocessing of the `result`.

4.2.4.3 Execution

An *emcee* application parallelized with PyCOMPSs **MUST** be executed as any COMPSs application (for full description about the execution environments and options please check the *Execution Environments* Section.).

For example, we can run Code 131 locally (using the PyCOMPSs CLI) with the following script:

```
pycompss run \
  --graph \
  sampling_pycompss.py
```

The execution output is:

```
[ INFO ] Inferred PYTHON language
[ INFO ] Using default location for project file: /opt/COMPSs//Runtime/configuration/xml/
↪projects/default_project.xml
[ INFO ] Using default location for resources file: /opt/COMPSs//Runtime/configuration/xml/
↪resources/default_resources.xml
[ INFO ] Using default execution type: compss

---------------- Executing sampling_pycompss.py ------------------------

WARNING: COMPSs Properties file is null. Setting default values
[(647)    API]  -  Starting COMPSs Runtime v2.10.rc2205 (build 20220527-0842.
↪r791bf7461bad1a1fab8f45853be7ba1c28b7bf93)
100
↪%|XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
↪10/10 [00:25<00:00,  2.51s/it]
PyCOMPSs took 31.8 seconds
[(34243)    API]  -  Execution Finished
```

```
--------------------------------------------------------
```

And the task dependency graph achieved:
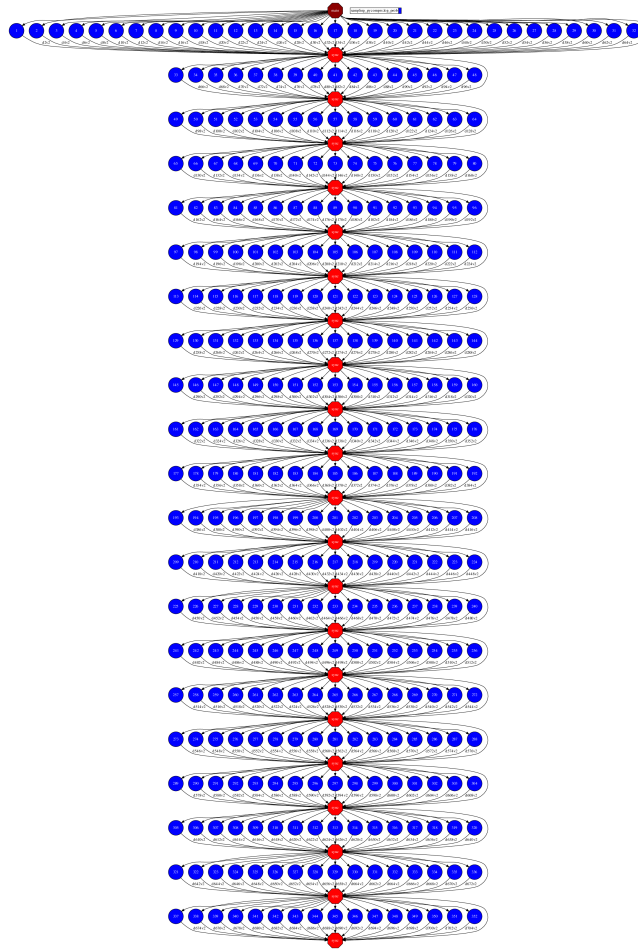
Figure 6: Task dependency graph of the Code 131 execution

Tip: The larger the `initial` array, the more parallelism can be achieved (larger width in the task dependency graph enabling to be executed in more resources).

If the `nsteps` is increased, the more iterations will be performed (larger height in the task dependency graph).

4.3 C/C++ Binding

COMPSs provides a binding for C and C++ applications. The new C++ version in the current release comes with support for objects as task parameters and the use of class methods as tasks.

4.3.1 Programming Model

As in Java, the application code is divided in 3 parts: the Task definition interface, the main code and task implementations. These files must have the following notation,: <app_ame>.idl, for the interface file, <app_-name>.cc for the main code and <app_name>-functions.cc for task implementations. Next paragraphs provide an example of how to define this files for matrix multiplication parallelised by blocks.

Task Definition Interface

As in Java the user has to provide a task selection by means of an interface. In this case the interface file has the same name as the main application file plus the suffix "idl", i.e. Matmul.idl, where the main file is called Matmul.cc.

Code 132: Matmul.idl

```
interface Matmul
{
    // C functions
    void initMatrix(inout Matrix matrix,
                    in int mSize,
                    in int nSize,
                    in double val);

    void multiplyBlocks(inout Block block1,
                        inout Block block2,
                        inout Block block3);
};
```

The syntax of the interface file is shown in the previous code. Tasks can be declared as classic C function prototypes, this allow to keep the compatibility with standard C applications. In the example, initMatrix and multiplyBlocks are functions declared using its prototype, like in a C header file, but this code is C++ as they have objects as parameters (objects of type Matrix, or Block).

The grammar for the interface file is:

```
["static"] return-type task-name ( parameter {, parameter }* );

return-type = "void" | type

ask-name = <qualified name of the function or method>

parameter = direction type parameter-name

direction = "in" | "out" | "inout"

type = "char" | "int" | "short" | "long" | "float" | "double" | "boolean" |
       "char[<size>]" | "int[<size>]" | "short[<size>]" | "long[<size>]" |
       "float[<size>]" | "double[<size>]" | "string" | "File" | class-name

class-name = <qualified name of the class>
```

Main Program

The following code shows an example of matrix multiplication written in C++.

Code 133: Matrix multiplication

```cpp
#include "Matmul.h"
#include "Matrix.h"
#include "Block.h"
int N; //MSIZE
int M; //BSIZE
double val;
int main(int argc, char **argv)
{
    Matrix A;
    Matrix B;
    Matrix C;

    N = atoi(argv[1]);
    M = atoi(argv[2]);
    val = atof(argv[3]);

    compss_on();

    A = Matrix::init(N,M,val);

    initMatrix(&B,N,M,val);
    initMatrix(&C,N,M,0.0);

    cout << "Waiting for initialization...\n";

    compss_wait_on(B);
    compss_wait_on(C);

    cout << "Initialization ends...\n";

    C.multiply(A, B);

    compss_off();
    return 0;
}
```

The developer has to take into account the following rules:

1. A header file with the same name as the main file must be included, in this case **Matmul.h**. This header file is automatically generated by the binding and it contains other includes and type-definitions that are required.
2. A call to the **compss_on** binding function is required to turn on the COMPSs runtime.
3. As in C language, out or inout parameters should be passed by reference by means of the "**&**" operator before the parameter name.
4. Synchronization on a parameter can be done calling the **compss_wait_on** binding function. The argument of this function must be the variable or object we want to synchronize.
5. There is an **implicit synchronization** in the init method of Matrix. It is not possible to know the address of "A" before exiting the method call and due to this it is necessary to synchronize before for the copy of the returned value into "A" for it to be correct.
6. A call to the **compss_off** binding function is required to turn off the COMPSs runtime.

Functions file

The implementation of the tasks in a C or C++ program has to be provided in a functions file. Its name must be the same as the main file followed by the suffix "-functions". In our case Matmul-functions.cc.

```cpp
#include  "Matmul.h"
#include  "Matrix.h"
#include  "Block.h"

void initMatrix(Matrix *matrix,int mSize,int nSize,double val){
     *matrix = Matrix::init(mSize, nSize, val);
}

void multiplyBlocks(Block *block1,Block *block2,Block *block3){
     block1->multiply(*block2, *block3);
}
```

In the previous code, class methods have been encapsulated inside a function. This is useful when the class method returns an object or a value and we want to avoid the explicit synchronization when returning from the method.

Additional source files

Other source files needed by the user application must be placed under the directory "**src**". In this directory the programmer must provide a **Makefile** that compiles such source files in the proper way. When the binding compiles the whole application it will enter into the src directory and execute the Makefile.

It generates two libraries, one for the master application and another for the worker application. The directive COMPSS_MASTER or COMPSS_WORKER must be used in order to compile the source files for each type of library. Both libraries will be copied into the lib directory where the binding will look for them when generating the master and worker applications.

The following sections provide a more detailed view of the C++ Binding. It will include the available API calls, how to deal with objects and having tasks as method objects as well as how to define constraints and task versions.

4.3.1.1 Binding API

Besides the aforementioned **compss_on**, **compss_off** and **compss_wait_on** functions, the C/C++ main program can make use of a variety of other API calls to better manage the synchronization of data generated by tasks. These calls are as follows:

void compss_ifstream(char * filename, ifstream* & * ifs) Given an uninitialized input stream *ifs* and a file *filename*, this function will synchronize the content of the file and initialize *ifs* to read from it.

void compss_ofstream(char * filename, ofstream* & * ofs) Behaves the same way as *compss_ifstream*, but in this case the opened stream is an output stream, meaning it will be used to write to the file.

FILE* compss_fopen(char * file_name, char * mode) Similar to the C/C++ *fopen* call. Synchronizes with the last version of file *file_name* and returns the FILE* pointer to further reference it. As the mode parameter it takes the same that can be used in *fopen* (*r, w, a, r+, w+* and *a+*).

void compss_wait_on(T & * obj) or T compss_wait_on(T* & * obj)** Synchronizes for the last version of object obj, meaning that the execution will stop until the value of *obj* up to that point of the code is received (and thus all tasks that can modify it have ended).

void compss_delete_file(char * file_name) Makes an asynchronous delete of file *filename*. When all previous tasks have finished updating the file, it is deleted.

void compss_delete_object(T & * obj)** Makes an asynchronous delete of an object. When all previous tasks have finished updating the object, it is deleted.

void compss_barrier() Similarly to the Python binding, performs an explicit synchronization without a return. When a *compss_barrier* is encountered, the execution will not continue until all the tasks submitted before the *compss_barrier* have finished.

4.3.1.2 Functions file

The implementation of the tasks in a C or C++ program has to be provided in a functions file. Its name must be the same as the main file followed by the suffix "-functions". In our case Matmul-functions.cc.

```cpp
#include "Matmul.h"
#include "Matrix.h"
#include "Block.h"

void initMatrix(Matrix *matrix,int mSize,int nSize,double val){
    *matrix = Matrix::init(mSize, nSize, val);
}

void multiplyBlocks(Block *block1,Block *block2,Block *block3){
    block1->multiply(*block2, *block3);
}
```

In the previous code, class methods have been encapsulated inside a function. This is useful when the class method returns an object or a value and we want to avoid the explicit synchronization when returning from the method.

4.3.1.3 Additional source files

Other source files needed by the user application must be placed under the directory "**src**". In this directory the programmer must provide a **Makefile** that compiles such source files in the proper way. When the binding compiles the whole application it will enter into the src directory and execute the Makefile.

It generates two libraries, one for the master application and another for the worker application. The directive COMPSS_MASTER or COMPSS_WORKER must be used in order to compile the source files for each type of library. Both libraries will be copied into the lib directory where the binding will look for them when generating the master and worker applications.

4.3.1.4 Class Serialization

In case of using an object as method parameter, as callee or as return of a call to a function, the object has to be serialized. The serialization method has to be provided inline in the header file of the object's class by means of the "**boost**" library. The next listing contains an example of serialization for two objects of the Block class.

```cpp
#ifndef BLOCK_H
#define BLOCK_H

#include    <vector>
#include    <boost/archive/text_iarchive.hpp>
#include    <boost/archive/text_oarchive.hpp>
#include    <boost/serialization/serialization.hpp>
#include    <boost/serialization/access.hpp>
#include    <boost/serialization/vector.hpp>

using namespace std;
using namespace boost;
using namespace serialization;

class Block {
public:
    Block(){};
    Block(int bSize);
    static Block *init(int bSize, double initVal);
    void multiply(Block block1, Block block2);
```

(continues on next page)

```
    void print();

private:
    int M;
    std::vector< std::vector< double > > data;

    friend class::serialization::access;
    template<class Archive>
    void serialize(Archive & ar, const unsigned int version) {
        ar & M;
        ar & data;
    }
};
#endif
```

For more information about serialization using "boost" visit the related documentation at *www.boost.org
<www.boost.org>*.

4.3.1.5 Method - Task

A task can be a C++ class method. A method can return a value, modify the *this* object, or modify a parameter.

If the method has a return value there will be an implicit synchronization before exit the method, but for the *this* object and parameters the synchronization can be done later after the method has finished.

This is because the *this* object and the parameters can be accessed inside and outside the method, but for the variable where the returned value is copied to, it can't be known inside the method.

```
#include "Block.h"

Block::Block(int bSize) {
        M = bSize;
        data.resize(M);
        for (int i=0; i<M; i++) {
                data[i].resize(M);
        }
}

Block *Block::init(int bSize, double initVal) {
        Block *block = new Block(bSize);
        for (int i=0; i<bSize; i++) {
                for (int j=0; j<bSize; j++) {
                        block->data[i][j] = initVal;
                }
        }
        return block;
}

#ifdef COMPSS_WORKER

void Block::multiply(Block block1, Block block2) {
        for (int i=0; i<M; i++) {
                for (int j=0; j<M; j++) {
                        for (int k=0; k<M; k++) {
                                data[i][j] += block1.data[i][k] * block2.data[k][j];
                        }
```

```
                }
        }
        this->print();
}

# endif

void Block::print() {
        for (int i=0; i<M; i++) {
                for (int j=0; j<M; j++) {
                        cout << data[i][j] << " ";
                }
                cout << "\r\n";
        }
}
```

4.3.1.6 Task Constraints

The C/C++ binding also supports the definition of task constraints. The task definition specified in the IDL file must be decorated/annotated with the *@Constraints*. Below, you can find and example of how to define a task with a constraint of using 4 cores. The list of constraints which can be defined for a task can be found in Section [sec:Constraints]

```
interface Matmul
{
        @Constraints(ComputingUnits = 4)
        void multiplyBlocks(inout Block block1,
                            in Block block2,
                            in Block block3);

};
```

4.3.1.7 Task Versions

Another COMPSs functionality supported in the C/C++ binding is the definition of different versions for a tasks. The following code shows an IDL file where a function has two implementations, with their corresponding constraints. It show an example where the *multiplyBlocks_GPU* is defined as a implementation of *multiplyBlocks* using the annotation/decoration *@Implements*. It also shows how to set a processor constraint which requires a GPU processor and a CPU core for managing the offloading of the computation to the GPU.

```
interface Matmul
{
        @Constraints(ComputingUnits=4);
        void multiplyBlocks(inout Block block1,
                            in Block block2,
                            in Block block3);

        // GPU implementation
        @Constraints(processors={
                @Processor(ProcessorType=CPU, ComputingUnits=1)});
                @Processor(ProcessorType=GPU, ComputingUnits=1)});
        @Implements(multiplyBlocks);
        void multiplyBlocks_GPU(inout Block block1,
                                in Block block2,
```

```
                              in Block block3);

};
```

4.3.2 Use of programming models inside tasks

To improve COMPSs performance in some cases, C/C++ binding offers the possibility to use programming models inside tasks. This feature allows the user to exploit the potential parallelism in their application's tasks.

4.3.2.1 OmpSs

COMPSs C/C++ binding supports the use of the programming model OmpSs. To use OmpSs inside COMPSs tasks we have to annotate the implemented tasks. The implementation of tasks was described in section [sec:functionsfile]. The following code shows a COMPSs C/C++ task without the use of OmpSs.

```
void compss_task(int* a, int N) {
  int i;
  for (i = 0; i < N; ++i) {
    a[i] = i;
  }
}
```

This code will assign to every array element its position in it. A possible use of OmpSs is the following.

```
void compss_task(int* a, int N) {
  int i;
  for (i = 0; i < N; ++i) {
   #pragma omp task
   {
    a[i] = i;
   }
  }
}
```

This will result in the parallelization of the array initialization, of course this can be applied to more complex implementations and the directives offered by OmpSs are much more. You can find the documentation and specification in https://pm.bsc.es/ompss.

There's also the possibility to use a newer version of the OmpSs programming model which introduces significant improvements, OmpSs-2. The changes at user level are minimal, the following image shows the array initialization using OmpSs-2.

```
void compss_task(int* a, int N) {
    int i;

    for (i = 0; i < N; ++i) {
     #pragma oss task
     {
      a[i] = i;
     }
    }
}
```

Documentation and specification of OmpSs-2 can be found in https://pm.bsc.es/ompss-2.

4.3.3 Application Compilation

To compile user's applications with the C/C++ binding two commands are used: The "**compss_build_app**"
command allows to compile applications for a single architecture, and the "**compss_build_app_multi_arch**"
command for multiple architectures. Both commands must be executed in the directory of the main application
code.

4.3.3.1 Single architecture

The user command "**compss_build_app**" compiles both master and worker for a single architecture (e.g. x86-64,
armhf, etc). Thus, whether you want to run your application in Intel based machine or ARM based machine, this
command is the tool you need.

When the target is the native architecture, the command to execute is very simple;

```
$~/matmul_objects> compss_build_app Matmul
[ INFO ] Java libraries are searched in the directory: /usr/lib/jvm/java-1.8.0-openjdk-amd64//
↪jre/lib/amd64/server
[ INFO ] Boost libraries are searched in the directory: /usr/lib/


...


[Info] The target host is: x86_64-linux-gnu

Building application for master...
g++ -g -O3 -I. -I/Bindings/c/share/c_build/worker/files/ -c Block.cc Matrix.cc
ar rvs libmaster.a Block.o Matrix.o
ranlib libmaster.a

Building application for workers...
g++ -DCOMPSS_WORKER -g -O3 -I. -I/Bindings/c/share/c_build/worker/files/ -c Block.cc -o Block.
↪o
g++ -DCOMPSS_WORKER -g -O3 -I. -I/Bindings/c/share/c_build/worker/files/ -c Matrix.cc -o␣
↪Matrix.o
ar rvs libworker.a Block.o Matrix.o
ranlib libworker.a


...


Command successful.
```

In order to build an application for a different architecture e.g. *armhf*, an environment must be provided, indicating
the compiler used to cross-compile, and also the location of some COMPSs dependencies such as java or boost
which must be compliant with the target architecture. This environment is passed by flags and arguments;

Please note that to use cross compilation features and multiple architecture builds, you need to do the proper
installation of COMPSs, find more information in the builders README.

```
$~/matmul_objects> compss_build_app --cross-compile --cross-compile-prefix=arm-linux-
↪gnueabihf- --java_home=/usr/lib/jvm/java-1.8.0-openjdk-armhf Matmul
[ INFO ] Java libraries are searched in the directory: /usr/lib/jvm/java-1.8.0-openjdk-armhf/
↪jre/lib/arm/server
[ INFO ] Boost libraries are searched in the directory: /usr/lib/
[ INFO ] You enabled cross-compile and the prefix to be used is: arm-linux-gnueabihf-


...


[ INFO ] The target host is: arm-linux-gnueabihf
```

<div align="right">(continues on next page)</div>

```
Building application for master...
g++ -g -O3 -I. -I/Bindings/c/share/c_build/worker/files/ -c Block.cc Matrix.cc
ar rvs libmaster.a Block.o Matrix.o
ranlib libmaster.a

Building application for workers...
g++ -DCOMPSS_WORKER -g -O3 -I. -I/Bindings/c/share/c_build/worker/files/ -c Block.cc -o Block.
↪o
g++ -DCOMPSS_WORKER -g -O3 -I. -I/Bindings/c/share/c_build/worker/files/ -c Matrix.cc -o␣
↪Matrix.o
ar rvs libworker.a Block.o Matrix.o
ranlib libworker.a


...

Command successful.
```

[The previous outputs have been cut for simplicity]

The –**cross-compile** flag is used to indicate the users desire to cross-compile the application. It enables the use of –**cross-compile-prefix** flag to define the prefix for the cross-compiler. Setting $CROSS_COMPILE environment variable will also work (in case you use the environment variable, the prefix passed by arguments is overrided with the variable value). This prefix is added to *$CC* and *$CXX* to be used by the user *Makefile* and lastly by the *GNU toolchain* . Regarding java and boost, –**java_home** and –**boostlib** flags are used respectively. In this case, users can also use teh *$JAVA_HOME* and *$BOOST_LIB* variables to indicate the java and boost for the target architecture. Note that these last arguments are purely for linkage, where *$LD_LIBRARY_PATH* is used by *Unix/Linux* systems to find libraries, so feel free to use it if you want to avoid passing some environment arguments.

4.3.3.2 Multiple architectures

The user command "**compss_build_app_multi_arch**" allows a to compile an application for several architectures. Users are able to compile both master and worker for one or more architectures. Environments for the target architectures are defined in a file specified by ***c*fg** flag. Imagine you wish to build your application to run the master in your Intel-based machine and the worker also in your native machine and in an ARM-based machine, without this command you would have to execute several times the command for a single architecture using its cross compile features. With the multiple architecture command is done in the following way.

```
$~/matmul_objects> compss_build_app_multi_arch --master=x86_64-linux-gnu --worker=arm-linux-
↪gnueabihf,x86_64-linux-gnu Matmul

[ INFO ] Using default configuration file: /opt/COMPSs/Bindings/c/cfgs/compssrc.
[ INFO ] Java libraries are searched in the directory: /usr/lib/jvm/java-1.8.0-openjdk-amd64/
↪jre/lib/amd64/server
[ INFO ] Boost libraries are searched in the directory: /usr/lib/


...

Building application for master...
g++ -g -O3 -I. -I/Bindings/c/share/c_build/worker/files/ -c Block.cc Matrix.cc
ar rvs libmaster.a Block.o Matrix.o
ranlib libmaster.a

Building application for workers...
g++ -DCOMPSS_WORKER -g -O3 -I. -I/Bindings/c/share/c_build/worker/files/ -c Block.cc -o Block.
↪o
```

```
g++ -DCOMPSS_WORKER -g -O3 -I. -I/Bindings/c/share/c_build/worker/files/ -c Matrix.cc -o␣
↪Matrix.o
ar rvs libworker.a Block.o Matrix.o
ranlib libworker.a


...


Command successful. # The master for x86_64-linux-gnu compiled successfuly


...


[ INFO ] Java libraries are searched in the directory: /usr/lib/jvm/java-1.8.0-openjdk-armhf/
↪jre/lib/arm/server
[ INFO ] Boost libraries are searched in the directory: /opt/install-arm/libboost


...


Building application for master...
arm-linux-gnueabihf-g++ -g -O3 -I. -I/Bindings/c/share/c_build/worker/files/ -c Block.cc␣
↪Matrix.cc
ar rvs libmaster.a Block.o Matrix.o
ranlib libmaster.a

Building application for workers...
arm-linux-gnueabihf-g++ -DCOMPSS_WORKER -g -O3 -I. -I/Bindings/c/share/c_build/worker/files/ -
↪c Block.cc -o Block.o
arm-linux-gnueabihf-g++ -DCOMPSS_WORKER -g -O3 -I. -I/Bindings/c/share/c_build/worker/files/ -
↪c Matrix.cc -o Matrix.o
ar rvs libworker.a Block.o Matrix.o
ranlib libworker.a


...


Command successful. # The worker for arm-linux-gnueabihf compiled successfuly


...


[ INFO ] Java libraries are searched in the directory: /usr/lib/jvm/java-1.8.0-openjdk-amd64/
↪jre/lib/amd64/server
[ INFO ] Boost libraries are searched in the directory: /usr/lib/


...


Building application for master...
g++ -g -O3 -I. -I/Bindings/c/share/c_build/worker/files/ -c Block.cc Matrix.cc
ar rvs libmaster.a Block.o Matrix.o
ranlib libmaster.a

Building application for workers...
g++ -DCOMPSS_WORKER -g -O3 -I. -I/Bindings/c/share/c_build/worker/files/ -c Block.cc -o Block.
↪o
g++ -DCOMPSS_WORKER -g -O3 -I. -I/Bindings/c/share/c_build/worker/files/ -c Matrix.cc -o␣
↪Matrix.o
ar rvs libworker.a Block.o Matrix.o
ranlib libworker.a
```

```
...

Command successful. # The worker for x86_64-linux-gnu compiled successfuly
```

[The previous output has been cut for simplicity]

Building for single architectures would lead to a directory structure quite different than the one obtained using the script for multiple architectures. In the single architecture case, only one master and one worker directories are expected. In the multiple architectures case, one master and one worker is expected per architecture.

```
.
|-- arm-linux-gnueabihf
|    `-- worker
|        `-- gsbuild
|            `-- autom4te.cache
|-- src
|-- x86_64-linux-gnu
|    |-- master
|    |    `-- gsbuild
|    |        `-- autom4te.cache
|    `-- worker
|        `-- gsbuild
|            `-- autom4te.cache
`-- xml

(Note than only directories are shown).
```

4.3.3.3 Using OmpSs

As described in section [sec:ompss] applications can use OmpSs and OmpSs-2 programming models. The compilation process differs a little bit compared with a normal COMPSs C/C++ application. Applications using OmpSs must be compiled using the `--ompss` option in the compss_build_app command.

```
$~/matmul_objects> compss_build_app --ompss Matmul
```

Executing the previous command will start the compilation of the application. Sometimes due to configuration issues OmpSs can not be found, the option `--with_ompss=/path/to/ompss` specifies the OmpSs path that the user wants to use in the compilation.

Applications using OmpSs-2 are similarly compiled. The options to compile with OmpSs-2 are `--ompss-2` and `--with_ompss-2=/path/to/ompss-2`

```
$~/matmul_objects> compss_build_app --with_ompss-2=/home/mdomingu/ompss-2 --ompss-2 Matmul
```

Remember that additional source files can be used in COMPSs C/C++ applications, if the user expects OmpSs or OmpSs-2 to be used in those files she, must be sure that the files are properly compiled with OmpSs or OmpSs-2.

4.3.4 Application Execution

The following environment variables must be defined before executing a COMPSs C/C++ application:

JAVA_HOME Java JDK installation directory (e.g. /usr/lib/jvm/java-8-openjdk/)

After compiling the application, two directories, master and worker, are generated. The master directory contains a binary called as the main file, which is the master application, in our example is called Matmul. The worker directory contains another binary called as the main file followed by the suffix "-worker", which is the worker application, in our example is called Matmul-worker.

The `runcompss` script has to be used to run the application:

```
$ runcompss /home/compss/tutorial_apps/c/matmul_objects/master/Matmul 3 4 2.0
```

The complete list of options of the runcompss command is available in Section *Executing COMPSs applications*.

4.3.5 Task Dependency Graph

COMPSs can generate a task dependency graph from an executed code. It is indicating by a

```
$ runcompss -g /home/compss/tutorial_apps/c/matmul_objects/master/Matmul 3 4 2.0
```

The generated task dependency graph is stored within the `$HOME/.COMPSs/<APP_NAME>_<00-99>/monitor` directory in dot format. The generated graph is `complete_graph.dot` file, which can be displayed with any dot viewer. COMPSs also provides the `compss_gengraph` script which converts the given dot file into pdf.

```
$ cd $HOME/.COMPSs/Matmul_02/monitor
$ compss_gengraph complete_graph.dot
$ evince complete_graph.pdf   # or use any other pdf viewer you like
```

The following figure depicts the task dependency graph for the Matmul application in its object version with 3x3 blocks matrices, each one containing a 4x4 matrix of doubles. Each block in the result matrix accumulates three block multiplications, i.e. three multiplications of 4x4 matrices of doubles.

Figure 7: Matmul Execution Graph.

The light blue circle corresponds to the initialization of matrix "A" by means of a method-task and it has an implicit synchronization inside. The dark blue circles correspond to the other two initializations by means of

---

function-tasks; in this case the synchronizations are explicit and must be provided by the developer after the task call. Both implicit and explicit synchronizations are represented as red circles.

Each green circle is a partial matrix multiplication of a set of 3. One block from matrix "A" and the correspondent one from matrix "B". The result is written in the right block in "C" that accumulates the partial block multiplications. Each multiplication set has an explicit synchronization. All green tasks are method-tasks and they are executed in parallel.

4.4 Constraints

This section provides a detailed information about all the supported constraints by the COMPSs runtime for **Java**, **Python** and **C/C++** languages. The constraints are defined as key-value pairs, where the key is the name of the constraint. Table 14 details the available constraints names for *Java*, *Python* and *C/C++*, its value type, its default value and a brief description.

Table 14: Arguments of the *@constraint* decorator

Java	Python	C / C++	Value type	Default value	Description
computingUnits	computing_-units	ComputingU-nits	<string>	"1"	Required number of computing units
isLocal	is_local	•	<"true"|"false">	"false"	The task must be executed in the node it's detected
processorName	processor_-name	ProcessorName	<string>	"[unassigned]"	Required processor name
processorSpeed	processor_-speed	ProcessorSpeed	<string>	"[unassigned]"	Required processor speed
processorArchitecture	processor_architecture	ProcessorArchitecture	<string>	"[unassigned]"	Required processor architecture
processorType	processor_type	ProcessorType	<string>	"[unassigned]"	Required processor type
processorPropertyName	processor_-property_name	ProcessorPropertyName	<string>	"[unassigned]"	Required processor property
processorPropertyValue	processor_-property_value	ProcessorPropertyValue	<string>	"[unassigned]"	Required processor property value
processorInternalMemorySize	processor_internal_memory_size	ProcessorInternalMemorySize	<string>	"[unassigned]"	Required internal device memory
processors	processors	•	List<@Processor>	"{}"	Required processors (check Table 15 for Processor details)
memorySize	memory_size	MemorySize	<string>	"[unassigned]"	Required memory size in GBs
memoryType	memory_type	MemoryType	<string>	"[unassigned]"	Required memory type (SRAM, DRAM, etc.)
storageSize	storage_size	StorageSize	<string>	"[unassigned]"	Required storage size in GBs
storageType	storage_type	StorageType	<string>	"[unassigned]"	Required storage type (HDD, SSD, etc.)
operatingSystemType	operating_system_type	OperatingSystemType	<string>	"[unassigned]"	Required operating system type (Windows, MacOS, Linux, etc.)
operatingSystemDistribution	operating_system_distribution	OperatingSystemDistribution	<string>	"[unassigned]"	Required operating system distribution (XP, Sierra, openSUSE, etc.)
operatingSystemVersion	operating_system_version	OperatingSystemVersion	<string>	"[unassigned]"	Required operating system version
wallClockLimit	wall_clock_-limit	WallClockLimit	<string>	"[unassigned]"	Maximum wall clock time
hostQueues	host_queues	HostQueues	<string>	"[unassigned]"	Required queues
appSoftware	app_software	AppSoftware	<string>	"[unassigned]"	Required applications that

All constraints are defined with a simple value except the *HostQueue* and *AppSoftware* constraints, which allow multiple values.

The *processors* constraint allows the users to define multiple processors for a task execution. This constraint is specified as a list of @Processor annotations that must be defined as shown in Table 15

Table 15: Arguments of the *@Processor* decorator

Annotation	Value type	Default value	Description
processorType	<string>	"CPU"	Required processor type (e.g. CPU or GPU)
computingUnits	<string>	"1"	Required number of computing units
name	<string>	"[unassigned]"	Required processor name
speed	<string>	"[unassigned]"	Required processor speed
architecture	<string>	"[unassigned]"	Required processor architecture
propertyName	<string>	"[unassigned]"	Required processor property
propertyValue	<string>	"[unassigned]"	Required processor property value
internalMemorySize	<string>	"[unassigned]"	Required internal device memory

Chapter 5

Execution Environments

This section is intended to show how to execute the COMPSs applications.

5.1 Schedulers

This section provides detailed information about all the schedulers that are implemented in COMPSs and can be used for the executions of the applications. Depending on the scheduler selected for your executions the tasks will be scheduled in a way or another and this will result in different execution times depending on the scheduler used.

COMPSs schedulers are organized in three families:

- Order strict: Policies give a priority to those tasks that become dependency free tasks. Only the dependency-free task with a higher priority can be submitted to execution. Tasks with lower priority can not overtake the execution of higher-priority tasks even if there are free resources that could host the execution of the former ones.
- Lookahead: As with o the order-strict family, policies give tasks a priority when they become dependency free. However, in this case, if there are not enough resources to host the execution of the highest-priority dependency-free task, another task with a lower priority can be submitted for execution overtaking the execution of the most prioritary one.
    - Successors: Within this family, an important group of schedulers give a higher priority to the tasks that become dependency-free when trying to submit an action to fill the resources released by their data predecessor.
- Full graph: Unlike the other two families that only consider dependency-free tasks, full-graph policies schedule the whole graph of the application on the currently available resources. Besides task dependencies, full-graph policies declare resources dependencies among tasks to guarantee resource constraints, and redefines them dynamically to optimize the execution.

Schedulers provided within the COMPSs release:

Table 16: Schedulers

Class name	Family	Description	Comments
es.bsc.compss.scheduler.orderstrict.fifo.FifoTS	orderstrict	Prioritizes task generation order (FIFO).	
es.bsc.compss.scheduler.lookahead.fifo.FifoTS	lookahead	Prioritizes task generation order (FIFO).	
es.bsc.compss.scheduler.lookahead.lifo.LifoTS	lookahead	Prioritizes task generation order (LIFO).	
es.bsc.compss.scheduler.lookahead.locality.LocalityTS	lookahead	Prioritizes data location and then (FIFO) task generation.	Default on runcompss executions
es.bsc.compss.scheduler.lookahead.successors.locality.LocalityTS	lookahead - successors	Prioritizes successors of the ended task, then the data locality on the worker and then the generation order.	Default for local disk executions on SCs
es.bsc.compss.scheduler.lookahead.mt.successors.locality.LocalityTS	lookahead - mt - successors	Prioritizes successors of the ended task, then the data locality on the worker and then the generation order.	Multi-threaded implementation.
es.bsc.compss.scheduler.lookahead.successors.fifo.FifoTS	lookahead - successors	Prioritizes successors of the ended task, and then the generation order.	
es.bsc.compss.scheduler.lookahead.mt.successors.fifo.FifoTS	lookahead - mt - successors	Prioritizes successors of the ended task, and then the generation order.	Multi-threaded implementation. Default for shared disk executions on SCs
es.bsc.compss.scheduler.lookahead.successors.lifo.LifoTS	lookahead - successors	Prioritizes successors of the ended task, and then the inverse generation order.	
es.bsc.compss.scheduler.lookahead.mt.successors.lifo.LifoTS	lookahead - mt - successors	Prioritizes successors of the ended task, and then the inverse generation order.	Multi-threaded implementation.
es.bsc.compss.scheduler.lookahead.successors.constraints.fifo.ConstraintsFifoTS	lookahead - successors	Prioritizes the successors of the ended task, then the task constraints (computing_units) and then generation order (FIFO).	
es.bsc.compss.scheduler.lookahead.mt.successors.constraints.fifo.ConstraintsFifoTS	lookahead - mt - successors	Prioritizes the successors of the ended task, then the task constraints (computing_units) and then generation order (FIFO).	Multi-threaded implementation
es.bsc.compss.scheduler.fullgraph.multiobjective.MOScheduler	fullgraph - multiobjective	Based on MO multi-objective function (time, energy, cost).	

Specifying the `--scheduler=<class>` option when launching a COMPSs execution with `enqueue_compss` or `runcompss` selects the scheduler that will drive the execution. In the case of having an agents deployment, the option indicates the scheduler used by that agent; agents deployment allows combining different scheduling strategies by setting up a different policy on each agent.

With the `--input_profile=<path>` option, application users can pass in to COMPSs the task profiles obtained from previous executions. Thus, the scheduler makes better decisions from an early time of the execution. To indicate the runtime a file where to save these profiles at the end of the execution, it is necessary that the user specifies the `--output_profile=<path>` option. If both paths match, the runtime will update its content.

5.2 Checkpointing

COMPSs and PyCOMPSs allow for task-level checkpointing. This feature allows the user to combine different checkpointing mechanisms to save the progress of an application execution (i.e., completed tasks and their output values) to recover it in the case of a failure. This section provides information on how to use the checkpointing recovery system.

Application developers can request the COMPSs runtime to checkpoint the application progress with the snapshot method of the API. When this method is invoked, the final version of each data value produced by any task of the application will be checkpointed. Upcoming executions will be able to resume the execution from that point with no additional development effort.

Java example:

```
import es.bsc.compss.api.COMPSs;

COMPSs.snapshot();
```

Python example:

```
from pycompss.api.api import compss_snapshot

compss_snapshot()
```

In addition, the COMPSs runtime system provides three mechanisms to perform an automatic checkpointing of the application: * Periodic checkpointing: periodically saves the application progress in configurable intervals of **n** hours, minutes, or seconds. * Finished tasks: triggers the checkpointing of the application progress upon the completion on **n** non-checkpointed tasks. * Tasks groups: this mechanism allows the definition of custom policies to checkpoint the application progress. A customizable policy assigns each task to a checkpointing group at task instantiation time. When all the tasks within the group have been instantiated – the policy closes the group –, the checkpoint manager determines the final version of each data produced by the tasks within the group. As tasks producing these values complete their computation, checkpoint manager requests a copy to checkpoint that value.

To develop checkpointing policies, checkpointing policy developer need to create a Java class extending the CheckpointManagerImpl class (`es.bsc.compss.checkpoint.CheckpointManagerImpl`) and implement the `assignTaskToGroup` method. The `assignTaskToGroup` method is invoked every time that the runtime instantiates a class and its purpose is to assign a task group to that task. To that end the policy can use any information related to the task; e.g., id of the task, method to execute, accessed data versions on its parameters, etc. Once the group is determined, the policy has to invoke the `addTaskToGroup` method to let the checkpoint manager to which group the task belongs. In addition, if the policy determines that all the tasks within the group have been instantiated, it needs to close the group using the `closeGroup` method.

The following snippet shows an example of a checkpoint policy implementation creating groups of N tasks subsequently instantiated.

Checkpoint polocy implementation

```
public class CheckpointPolicyInstantiatedGroup extends CheckpointManagerImpl {

    private int currentGroup = 0;
    private int groupSize = 3;
    public CheckpointPolicyInstantiatedGroup(HashMap<String, String> config, AccessProcessor␣
↪ap) {
        super(config, 0, 0, ap);
        this.groupsize = config.get("instantiated.group");
    }


    @Override
    protected void assignTaskToGroup(Task t) {
        // Assign the task to the decided group
```

(continues on next page)

```
        CheckpointGroupImpl group = this.addTaskToGroup(t, String.valueOf(countingGroup));
        // If the group reaches its size of closure it closes (in this case is 1)
        if (group.getSize() == groupSize) {
            this.closeGroup(String.valueOf(countingGroup));
            countingGroup += 1;
        }

}
```

COMPSs release contains three pre-defined policies, each leveraging on only one of these mechanisms:

Table 17: Checkpointing

Policy name	Class name	Params	Description
Periodic Time (PT)	es.bsc.compss.checkpoint.policies.CheckpointPolicyPeriodicTime	period.time	Checkpoints every n time
Finished Tasks (FT)	es.bsc.compss.checkpoint.policies.CheckpointPolicyFinishedTasks	finished.tasks	Checkpoints every n finished tasks
Instantiated Tasks Group (ITG)	es.bsc.compss.checkpoint.policies.CheckpointPolicyInstantiatedGroup	instantiated.group	Checkpoints every n instantiated tasks

In order to use checkpointing it is needed to specify three flags in the `enqueue_compss` and `runcompss`. These are:
* `--checkpointer`: This parameter lets you choose the checkpointing policy, and assign one of the class names. *
`--checkpointer_params`: This parameter lets you choose the checkpointing span, depending on the policy the user
has to choose the corresponding param from the table (in the time case the user has to define the time in either s
(seconds), m (minutes) or h (hours), and other options that will be explained later on. * `--checkpointer_folder`:
This parameter defines the folder where the checkpoints will be saved.

As an additional feature the user can avoid checkpointing a specific task, that may have a big overhead on the
filesystem by passing the list of signature names in the `checkpointer_params` flag using the following parameter
`avoid.checkpoint`

An example of usage would be the following:

```
--checkpointer_params=period.time:s,avoid.checkpoint:[checkpoint_file_test.increment] \
--checkpointer=es.bsc.compss.checkpointer.policies.CheckpointPolicyPeriodicTime \
--checkpointer_folder=/tmp/checkpointing/ \
```

5.3 Deployments

This section is intended to show how to execute the COMPSs applications deploying COMPSs.

5.3.1 Master-Worker Deployments

This section is intended to show how to execute the COMPSs applications deploying COMPSs as a master-worker
structure.

5.3.1.1 Local

This section is intended to walk you through the COMPSs usage in local machines.

Executing COMPSs applications

Prerequisites

Prerequisites vary depending on the application's code language: for Java applications the users need to have a **jar archive** containing all the application classes, for Python applications there are no requirements and for C/C++ applications the code must have been previously compiled by using the *compss_build_app* command.

For further information about how to develop COMPSs applications please refer to *Application development*.

Runcompss command

COMPSs applications are executed using the **runcompss** command:

```
compss@bsc:~$ runcompss [options] application_name [application_arguments]
```

The application name must be the fully qualified name of the application in Java, the path to the *.py* file containing the main program in Python and the path to the master binary in C/C++.

The application arguments are the ones passed as command line to main application. This parameter can be empty.

The **runcompss** command allows the users to customize a COMPSs execution by specifying different options. For clarity purposes, parameters are grouped in *Runtime configuration*, *Tools enablers* and *Advanced options*.

```
compss@bsc:~$ runcompss -h

Usage: /opt/COMPSs/Runtime/scripts/user/runcompss [options] application_name application_
↪arguments

* Options:
  General:
    --help, -h                          Print this help message

    --opts                              Show available options

    --version, -v                       Print COMPSs version

  Tools enablers:
    --graph=<bool>, --graph, -g         Generation of the complete graph (true/false)
                                        When no value is provided it is set to true
                                        Default: false
    --tracing=<bool>, --tracing, -t     Set generation of traces.
                                        Default: false
    --monitoring=<int>, --monitoring, -m  Period between monitoring samples (milliseconds)
                                        When no value is provided it is set to 2000
                                        Default: 0
    --external_debugger=<int>,
    --external_debugger                 Enables external debugger connection on the␣
↪specified port (or 9999 if empty)
                                        Default: false
    --jmx_port=<int>                    Enable JVM profiling on specified port
```

(continues on next page)

```
 Runtime configuration options:
    --task_execution=<compss|storage>        Task execution under COMPSs or Storage.
                                             Default: compss
    --storage_impl=<string>                  Path to an storage implementation. Shortcut to␣
↪setting pypath and classpath. See Runtime/storage in your installation folder.
    --storage_conf=<path>                    Path to the storage configuration file
                                             Default: null
    --project=<path>                         Path to the project XML file
                                             Default: /opt/COMPSs//Runtime/configuration/xml/
↪projects/default_project.xml
    --resources=<path>                       Path to the resources XML file
                                             Default: /opt/COMPSs//Runtime/configuration/xml/
↪resources/default_resources.xml
    --lang=<name>                            Language of the application (java/c/python)
                                             Default: Inferred is possible. Otherwise: java
    --summary                                Displays a task execution summary at the end of␣
↪the application execution
                                             Default: false
    --log_level=<level>, --debug, -d         Set the debug level: off | info | api | debug |␣
↪trace
                                             Warning: Off level compiles with -O2 option␣
↪disabling asserts and __debug__
                                             Default: off


 Advanced options:
    --extrae_config_file=<path>              Sets a custom extrae config file. Must be in a␣
↪shared disk between all COMPSs workers.
                                             Default: /opt/COMPSs//Runtime/configuration/xml/
↪tracing/extrae_basic.xml
    --extrae_config_file_python=<path>       Sets a custom extrae config file for python. Must␣
↪be in a shared disk between all COMPSs workers.
                                             Default: null
    --trace_label=<string>                   Add a label in the generated trace file. Only␣
↪used in the case of tracing is activated.
                                             Default: Applicacion name
    --tracing_task_dependencies=<bool>       Adds communication lines for the task␣
↪dependencies (true/false)
                                             Default: false
    --generate_trace=<bool>                  Converts the events register into a trace file.␣
↪Only used in the case of activated tracing.
                                             Default: true
    --delete_trace_packages=<bool>           If true, deletes the tracing packages created by␣
↪the run.
                                             Default: true. Automatically, disabled if the␣
↪trace is not generated.
    --custom_threads=<bool>                  Threads in the trace file are re-ordered and␣
↪customized to indicate the function of the thread.
                                             Only used when the tracing is activated and a␣
↪trace file generated.
                                             Default: true
    --comm=<ClassName>                       Class that implements the adaptor for␣
↪communications

                                             Supported adaptors:
                                                     ├─ es.bsc.compss.nio.master.NIOAdaptor
                                                     └─ es.bsc.compss.gat.master.GATAdaptor
                                             Default: es.bsc.compss.nio.master.NIOAdaptor
```

```
    --conn=<className>                      Class that implements the runtime connector for␣
↪the cloud

                                            Supported connectors:
                                                   ├── es.bsc.compss.connectors.
↪DefaultSSHConnector

                                                   └── es.bsc.compss.connectors.
↪DefaultNoSSHConnector

                                            Default: es.bsc.compss.connectors.
↪DefaultSSHConnector
    --streaming=<type>                      Enable the streaming mode for the given type.
                                            Supported types: FILES, OBJECTS, PSCOS, ALL, NONE
                                            Default: NONE
    --streaming_master_name=<str>           Use an specific streaming master node name.
                                            Default: Empty
    --streaming_master_port=<int>           Use an specific port for the streaming master.
                                            Default: Empty
    --scheduler=<className>                 Class that implements the Scheduler for COMPSs
                                            Supported schedulers:
                                                   ├── es.bsc.compss.components.impl.
↪TaskScheduler

                                                   ├── es.bsc.compss.scheduler.orderstrict.
↪fifo.FifoTS

                                                   ├── es.bsc.compss.scheduler.lookahead.fifo.
↪FifoTS

                                                   ├── es.bsc.compss.scheduler.lookahead.lifo.
↪LifoTS

                                                   ├── es.bsc.compss.scheduler.lookahead.
↪locality.LocalityTS

                                                   ├── es.bsc.compss.scheduler.lookahead.
↪successors.constraintsfifo.ConstraintsFifoTS

                                                   ├── es.bsc.compss.scheduler.lookahead.mt.
↪successors.constraintsfifo.ConstraintsFifoTS

                                                   ├── es.bsc.compss.scheduler.lookahead.
↪successors.fifo.FifoTS

                                                   ├── es.bsc.compss.scheduler.lookahead.mt.
↪successors.fifo.FifoTS

                                                   ├── es.bsc.compss.scheduler.lookahead.
↪successors.lifo.LifoTS

                                                   ├── es.bsc.compss.scheduler.lookahead.mt.
↪successors.lifo.LifoTS

                                                   ├── es.bsc.compss.scheduler.lookahead.
↪successors.locality.LocalityTS

                                                   └── es.bsc.compss.scheduler.lookahead.mt.
↪successors.locality.LocalityTS

                                            Default: es.bsc.compss.scheduler.lookahead.
↪locality.LocalityTS
    --scheduler_config_file=<path>          Path to the file which contains the scheduler␣
↪configuration.

                                            Default: Empty
    --checkpoint=<className>                Class that implements the Checkpoint Management␣
↪policy

                                            Supported checkpoint policies:
                                                   ├── es.bsc.compss.checkpoint.policies.
↪CheckpointPolicyInstantiatedGroup

                                                   ├── es.bsc.compss.checkpoint.policies.
↪CheckpointPolicyPeriodicTime
```

```
                                                    ├── es.bsc.compss.checkpoint.policies.
↪CheckpointPolicyFinishedTasks

                                                    └── es.bsc.compss.checkpoint.policies.
↪NoCheckpoint

                                            Default: es.bsc.compss.checkpoint.policies.
↪NoCheckpoint
   --checkpoint_params=<string>            Checkpoint configuration parameter.
                                            Default: Empty
   --checkpoint_folder=<path>              Checkpoint folder.
                                            Default: Mandatory parameter
   --library_path=<path>                   Non-standard directories to search for libraries␣
↪(e.g. Java JVM library, Python library, C binding library)
                                            Default: Working Directory
   --classpath=<path>                      Path for the application classes / modules
                                            Default: Working Directory
   --appdir=<path>                         Path for the application class folder.
                                            Default: /home/user
   --pythonpath=<path>                     Additional folders or paths to add to the␣
↪PYTHONPATH
                                            Default: /home/user
   --env_script=<path>                     Path to the script file where the application␣
↪environment variables are defined.
                                            COMPSs sources this script before running the␣
↪application.
                                            Default: Empty
   --log_dir=<path>                        Directory to store COMPSs log files (a .COMPSs/␣
↪folder will be created inside this location)
                                            Default: User home
   --master_working_dir=<path>             Use a specific directory to store COMPSs␣
↪temporary files in master
                                            Default: <log_dir>/.COMPSs/<app_name>/tmpFiles
   --uuid=<int>                            Preset an application UUID
                                            Default: Automatic random generation
   --master_name=<string>                  Hostname of the node to run the COMPSs master
                                            Default: Empty
   --master_port=<int>                     Port to run the COMPSs master communications.
                                            Only for NIO adaptor
                                            Default: [43000,44000]
   --jvm_master_opts="<string>"            Extra options for the COMPSs Master JVM. Each␣
↪option separed by "," and without blank spaces (Notice the quotes)
                                            Default: Empty
   --jvm_workers_opts="<string>"           Extra options for the COMPSs Workers JVMs. Each␣
↪option separed by "," and without blank spaces (Notice the quotes)
                                            Default: -Xms256m,-Xmx1024m,-Xmn100m
   --cpu_affinity="<string>"               Sets the CPU affinity for the workers
                                            Supported options: disabled, automatic, dlb or␣
↪user defined map of the form "0-8/9,10,11/12-14,15,16"
                                            Default: automatic
   --gpu_affinity="<string>"               Sets the GPU affinity for the workers
                                            Supported options: disabled, automatic, user␣
↪defined map of the form "0-8/9,10,11/12-14,15,16"
                                            Default: automatic
   --fpga_affinity="<string>"              Sets the FPGA affinity for the workers
                                            Supported options: disabled, automatic, user␣
↪defined map of the form "0-8/9,10,11/12-14,15,16"
                                            Default: automatic
```

```
   --fpga_reprogram="<string>"              Specify the full command that needs to be␣
↪executed to reprogram the FPGA with the desired bitstream. The location must be an absolute␣
↪path.
                                           Default: Empty
   --io_executors=<int>                    IO Executors per worker
                                           Default: 0
   --task_count=<int>                      Only for C/Python Bindings. Maximum number of␣
↪different functions/methods, invoked from the application, that have been selected as tasks
                                           Default: 50
   --input_profile=<path>                  Path to the file which stores the input␣
↪application profile
                                           Default: Empty
   --output_profile=<path>                 Path to the file to store the application profile␣
↪at the end of the execution
                                           Default: Empty
   --PyObject_serialize=<bool>             Only for Python Binding. Enable the object␣
↪serialization to string when possible (true/false).
                                           Default: false
   --persistent_worker_c=<bool>            Only for C Binding. Enable the persistent worker␣
↪in c (true/false).
                                           Default: false
   --enable_external_adaptation=<bool>     Enable external adaptation. This option will␣
↪disable the Resource Optimizer.
                                           Default: false
   --gen_coredump                          Enable master coredump generation
                                           Default: false
   --keep_workingdir                       Do not remove the worker working directory after␣
↪the execution
                                           Default: false
   --python_interpreter=<string>           Python interpreter to use (python/python3).
                                           Default: python3 Version:
   --python_propagate_virtual_environment=<bool>  Propagate the master virtual environment␣
↪to the workers (true/false).
                                              Default: true
   --python_mpi_worker=<bool>              Use MPI to run the python worker instead of␣
↪multiprocessing. (true/false).
                                           Default: false
   --python_memory_profile                 Generate a memory profile of the master.
                                           Default: false
   --python_worker_cache=<string>          Python worker cache (true/size/false).
                                           Only for NIO without mpi worker and python >= 3.8.
                                           Default: false
   --python_cache_profiler=<bool>          Python cache profiler (true/false).
                                           Only for NIO without mpi worker and python >= 3.8.
                                           Default: false
   --wall_clock_limit=<int>                Maximum duration of the application (in seconds).
                                           Default: 0
   --shutdown_in_node_failure=<bool>       Stop the whole execution in case of Node Failure.
                                           Default: false
   --provenance, -p                        Generate COMPSs workflow provenance data in RO-
↪Crate format from YAML file. Automatically activates -graph and -output_profile.
                                           Default: false

* Application name:
   For Java applications:   Fully qualified name of the application
   For C applications:      Path to the master binary
```

```
    For Python applications: Path to the .py file containing the main program

* Application arguments:
    Command line arguments to pass to the application. Can be empty.
```

> **Warning:** The `cpu_affinity` feature is not available in macOS distributions. Then, for all macOS executions the flag `--cpu_affinity=disabled` must be specified, no matter if they are Java, Python or C/C++.

Running a COMPSs application

Before running COMPSs applications the application files **must** be in the **CLASSPATH**. Thus, when launching a COMPSs application, users can manually pre-set the **CLASSPATH** environment variable or can add the `--classpath` option to the `runcompss` command.

The next three sections provide specific information for launching COMPSs applications developed in different code languages (Java, Python and C/C++). For clarity purposes, we will use the *Simple* application (developed in Java, Python and C++) available in the COMPSs Virtual Machine or at https://compss.bsc.es/projects/bar webpage. This application takes an integer as input parameter and increases it by one unit using a task. For further details about the codes please refer to *Sample Applications*.

Tip: For further information about applications scheduling refer to *Schedulers*.

Running Java applications

A Java COMPSs application can be launched through the following command:

```
compss@bsc:~$ cd tutorial_apps/java/simple/jar/
compss@bsc:~/tutorial_apps/java/simple/jar$ runcompss simple.Simple <initial_number>
```

```
compss@bsc:~/tutorial_apps/java/simple/jar$ runcompss simple.Simple 1
[  INFO] Using default execution type: compss
[  INFO] Using default location for project file: /opt/COMPSs/Runtime/configuration/xml/
↪projects/default_project.xml
[  INFO] Using default location for resources file: /opt/COMPSs/Runtime/configuration/xml/
↪resources/default_resources.xml
[  INFO] Using default language: java

----------------- Executing simple.Simple -------------------------

WARNING: COMPSs Properties file is null. Setting default values
[(1066)    API]  -  Starting COMPSs Runtime v<version>
Initial counter value is 1
Final counter value is 2
[(4740)    API]  -  Execution Finished

------------------------------------------------------------
```

In this first execution we use the default value of the `--classpath` option to automatically add the jar file to the classpath (by executing runcompss in the directory which contains the jar file). However, we can explicitly do this by exporting the **CLASSPATH** variable or by providing the `--classpath` value. Next, we provide two more ways to perform the same execution:

```
compss@bsc:~$ export CLASSPATH=$CLASSPATH:/home/compss/tutorial_apps/java/simple/jar/simple.
↪jar
compss@bsc:~$ runcompss simple.Simple <initial_number>
```

```
compss@bsc:~$ runcompss --classpath=/home/compss/tutorial_apps/java/simple/jar/simple.jar \
                       simple.Simple <initial_number>
```

Running Python applications

To launch a COMPSs Python application users have to provide the `--lang=python` option to the runcompss command. If the extension of the main file is a regular Python extension (`.py` or `.pyc`) the *runcompss* command can also infer the application language without specifying the *lang* flag.

```
compss@bsc:~$ cd tutorial_apps/python/simple/
compss@bsc:~/tutorial_apps/python/simple$ runcompss --lang=python ./simple.py <initial_number>
```

```
compss@bsc:~/tutorial_apps/python/simple$ runcompss simple.py 1
[  INFO] Using default execution type: compss
[  INFO] Using default location for project file: /opt/COMPSs/Runtime/configuration/xml/
↪projects/default_project.xml
[  INFO] Using default location for resources file: /opt/COMPSs/Runtime/configuration/xml/
↪resources/default_resources.xml
[  INFO] Inferred PYTHON language

----------------- Executing simple.py -------------------------

WARNING: COMPSs Properties file is null. Setting default values
[(616)    API]  -  Starting COMPSs Runtime v<version>
Initial counter value is 1
Final counter value is 2
[(4297)    API]  -  Execution Finished


------------------------------------------------------------
```

> **Attention:** Executing without debug (e.g. default log level or `--log_level=off`) uses -O2 compiled sources, disabling `asserts` and `__debug__`.

Alternatively, it is possible to execute the a COMPSs Python application using `pycompss` as module:

```
compss@bsc:~$ python -m pycompss <runcompss_flags> <application> <application_parameters>
```

Consequently, the previous example could also be run as follows:

```
compss@bsc:~$ cd tutorial_apps/python/simple/
compss@bsc:~/tutorial_apps/python/simple$ python -m pycompss simple.py <initial_number>
```

If the `-m pycompss` is not set, the application will be run ignoring all PyCOMPSs imports, decorators and API calls, that is, sequentially.

In order to run a COMPSs Python application with a different interpreter, the *runcompss* command provides a specific flag:

```
compss@bsc:~$ cd tutorial_apps/python/simple/
compss@bsc:~/tutorial_apps/python/simple$ runcompss --python_interpreter=python3 ./simple.py
↪<initial_number>
```

However, when using the *pycompss* module, it is inferred from the python used in the call:

```
compss@bsc:~$ cd tutorial_apps/python/simple/
compss@bsc:~/tutorial_apps/python/simple$ python3 -m pycompss simple.py <initial_number>
```

Finally, both *runcompss* and *pycompss* module provide a particular flag for virtual environment propagation (`--python_propagate_virtual_environment=<bool>`). This, flag is intended to activate the current virtual environment in the worker nodes when set to true.

Specific flags

Some of the **runcompss** flags are only for PyCOMPSs application execution:

--pythonpath=**<path>** Additional folders or paths to add to the PYTHONPATH Default: /home/user

--PyObject_serialize=**<bool>** Only for Python Binding. Enable the object serialization to string when possible (true/false). Default: false

--python_interpreter=**<string>** Python interpreter to use (python/python2/python3). Default: "python" version

--python_propagate_virtual_environment=**<true>** Propagate the master virtual environment to the workers (true/false). Default: true

--python_mpi_worker=**<false>** Use MPI to run the python worker instead of multiprocessing. (true/false). Default: false

--python_memory_profile Generate a memory profile of the master. Default: false

> See: *Memory Profiling*

--python_worker_cache=**<string>** Python worker cache (true/true:size/false). Only for NIO without mpi worker and python >= 3.8. Default: false

> See: *Worker cache*

--python_cache_profiler=**<bool>** Python cache profiler (true/false). Only for NIO without mpi worker and python >= 3.8. Default: false

> See: *Worker cache profiling*

> **Warning:** For macOS systems, the flag `--python_interpreter=/path_to/python` must be passed to ensure the same Python version is used both in master and worker parts of the application (the application will crash otherwise). We recommend to use pyenv to manage the macOS installed Python versions. An example using pyenv would be: `--python_interpreter=/Users/username/.pyenv/shims/python3` In addition, be careful with `Xcode` updates, since they can modify the Python system version.

Worker cache

The `--python_worker_cache` is used to enable a cache between processes on each worker node. More specifically, this flag enables a shared memory space between the worker processes, so that they can share objects between processess in order to leverage the deserialization overhead.

The possible values are:

`--python_worker_cache=false` Disable the cache. This is the default value.
`--python_worker_cache=true` Enable the cache. The default cache size is 25% of the worker node memory.
`--python_worker_cache=true:<SIZE>` Enable the cache with specific cache size (in bytes).

During execution, each worker will try to store automatically the parameters and return objects, so that next tasks can make use of them without needing to deserialize from file.

Important: The supported objects to be stored in the cache is **limited** to: **python primitives** (int, float, bool, str (less than 10 Mb), bytes (less than 10 Mb) and None), **lists** (composed by python primitives), **tuples** (composed by python primitives) and **Numpy ndarrays**.

It is important to take into account that storing the objects in cache has some non negligible overhead that can be representative, while getting objects from cache shows to be more efficient than deserialization. Consequently, the applications that most benefit from the cache are the ones that reuse many times the same objects.

Avoiding to store an object into the cache is possible by setting `Cache` to `False` into the `@task` decorator for the parameter. For example, Code 134 shows how to avoid caching the `value` parameter.

Code 134: Avoid parameter caching

```python
from pycompss.api.task import task
from pycompss.api.parameter import *

@task(value={Cache: False})
def mytask(value):
    ....
```

Task return objects are also automatically stored into cache. To avoid caching return objects it is necessary to set `cache_returns=False` into the `@task` decorator, as Code 135 shows.

Code 135: Avoid return caching

```python
from pycompss.api.task import task

@task(returns=1, cache_returns=False)
def mytask():
    return list(range(10))
```

Worker cache profiling

In order to use the cache profiler, you need to add the following flag:

`--python_cache_profiler=true` Additionally, you also need to activate the cache with `--python_worker_-cache=true`.

When using the cache profiler, the cache parameter in `@task` decorator is going to be ignored and all elements that can be stored in the cache will be stored.

The cache profiling file will be located in the workers' folder within the log folder. In this file, you will find a summary showing for each function and parameter (including the return of the function), how many times has been the parameter been added to the cache (*PUT*), and how many times has been this parameter been deserialized from the cache (*GET*). Furthermore, there is also a list (*USED IN*), that shows in which parameter of which function the added parameter has been used.

Additional features

Concurrent serialization

It is possible to perform concurrent serialization of the objects in the master when using Python 3. To this end, just export the `COMPSS_THREADED_SERIALIZATION` environment variable with any value:

```
compss@bsc:~$ export COMPSS_THREADED_SERIALIZATION=1
```

> **Caution:** Please, make sure that the `COMPSS_THREADED_SERIALIZATION` environment variable is not in the environment (`env`) to avoid the concurrent serialization of the objects in the master.

Tip: This feature can also be used within supercomputers in the same way.

Running C/C++ applications

To launch a COMPSs C/C++ application users have to compile the C/C++ application by means of the `compss_-build_app` command. For further information please refer to *C/C++ Binding*. Once complied, the `--lang=c` option must be provided to the runcompss command. If the main file is a C/C++ binary the *runcompss* command can also infer the application language without specifying the *lang* flag.

```
compss@bsc:~$ cd tutorial_apps/c/simple/
compss@bsc:~/tutorial_apps/c/simple$ runcompss --lang=c simple <initial_number>
```

```
compss@bsc:~/tutorial_apps/c/simple$ runcompss ~/tutorial_apps/c/simple/master/simple 1
[  INFO] Using default execution type: compss
[  INFO] Using default location for project file: /opt/COMPSs/Runtime/configuration/xml/
↪projects/default_project.xml
[  INFO] Using default location for resources file: /opt/COMPSs/Runtime/configuration/xml/
↪resources/default_resources.xml
[  INFO] Inferred C/C++ language

----------------- Executing simple -------------------------

JVM_OPTIONS_FILE: /tmp/tmp.ItT1tQfKgP
COMPSS_HOME: /opt/COMPSs
Args: 1

WARNING: COMPSs Properties file is null. Setting default values
[(650)    API] -  Starting COMPSs Runtime v<version>
Initial counter value is 1
[   BINDING] -  @compss_wait_on  -  Entry.filename: counter
[   BINDING] -  @compss_wait_on  -  Runtime filename: d1v2_1497432831496.IT
Final counter value is 2
[(4222)    API] -  Execution Finished

------------------------------------------------------------
```

Walltime

The `runcompss` command provides the `--wall_clock_limit` for the users to specify the maximum execution time for the application (in seconds). If the time is reached, the execution is stopped.

Tip: This flag enables to stop the execution of an application in a contolled way if the execution is taking more than expected.

Additional configurations

The COMPSs runtime has two configuration files: `resources.xml` and `project.xml` . These files contain information about the execution environment and are completely independent from the application.

For each execution users can load the default configuration files or specify their custom configurations by using, respectively, the `--resources=<absolute_path_to_resources.xml>` and the `--project=<absolute_path_-to_project.xml>` in the `runcompss` command. The default files are located in the `/opt/COMPSs/Runtime/configuration/xml/` path. Users can manually edit these files or can use the *Eclipse IDE* tool developed for COMPSs.

For further details please check the *Configuration Files*.
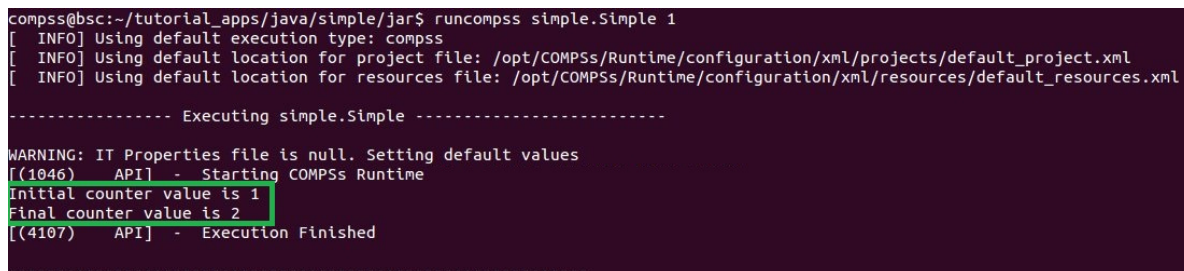
Results and logs

Results

When executing a COMPSs application we consider different type of results:

- **Application Output:** Output generated by the application.
- **Application Files:** Files used or generated by the application.
- **Tasks Output:** Output generated by the tasks invoked from the application.

Regarding the application output, COMPSs will preserve the application output but will add some pre and post output to indicate the COMPSs Runtime state. Figure 8 shows the standard output generated by the execution of the Simple Java application. The green box highlights the application `stdout` while the rest of the output is produced by COMPSs.

Figure 8: Output generated by the execution of the *Simple* Java application with COMPSs

Regarding the application files, COMPSs **does not modify** any of them and thus, the results obtained by executing the application with COMPSs are the same than the ones generated by the sequential execution of the application.

Regarding the tasks output, COMPSs introduces some modifications due to the fact that tasks can be executed in remote machines. After the execution, COMPSs stores the *stdout* and the *stderr* of each job (a task execution) inside the `` /**home/$USER/.COMPSs/$APPNAME/$EXEC_NUMBER/jobs/** `` directory of the main application node.

Figure 9 and Figure 10 show an example of the results obtained from the execution of the *Hello* Java application. While Figure 9 provides the output of the sequential execution of the application (without COMPSs), Figure 10

---

provides the output of the equivalent COMPSs execution. Please note that the sequential execution produces the `Hello World! (from a task)` message in the `stdout` while the COMPSs execution stores the message inside the `job1_NEW.out` file.

Figure 9: Sequential execution of the *Hello* java application

Figure 10: COMPSs execution of the *Hello* java application

Logs

COMPSs includes three log levels for running applications but users can modify them or add more levels by editing the logger files under the `/opt/COMPSs/Runtime/configuration /log/` folder. Any of these log levels can be selected by adding the `--log_level=<debug | info | off>` flag to the `runcompss` command. The default value is `off`.

The logs generated by the `NUM_EXEC` execution of the application APP by the user USER are stored under `/home/$USER/.COMPSs/$APP/$EXEC_NUMBER/` folder (from this point on: **base log folder**). The `EXEC_NUMBER` execution number is automatically used by COMPSs to prevent mixing the logs of data of different executions.

When running COMPSs with **log level off** only the errors are reported. This means that the *base log folder* will contain two empty files (`runtime.log` and `resources.log`) and one empty folder (`jobs`). If somehow the application has failed, the `runtime.log` and/or the `resources.log` will not be empty and a new file per failed job will appear inside the `jobs` folder to store the `stdout` and the `stderr`. Figure 11 shows the logs generated by the execution of the Simple java application (without errors) in **off** mode.

Figure 11: Structure of the logs folder for the Simple java application in **off** mode

When running COMPSs with **log level info** the *base log folder* will contain two files (`runtime.log` and `resources.log`) and one folder (`jobs`). The `runtime.log` file contains the execution information retrieved from the master

resource, including the file transfers and the job submission details. The `resources.log` file contains information about the available resources such as the number of processors of each resource (slots), the information about running or pending tasks in the resource queue and the created and destroyed resources. The jobs folder will be empty unless there has been a failed job. In this case it will store, for each failed job, one file for the `stdout` and another for the `stderr`. As an example, Figure 12 shows the logs generated by the same execution than the previous case but with **info** mode.
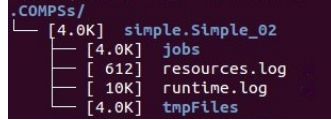
Figure 12: Structure of the logs folder for the Simple java application in **info** mode

The `runtime.log` and `resources.log` are quite large files, thus they should be only checked by advanced users. For an easier interpretation of these files the COMPSs Framework includes a monitor tool. For further information about the COMPSs Monitor please check *Monitor*.

Figure 13 and Figure 14 provide the content of these two files generated by the execution of the *Simple* java application.

Figure 13: runtime.log generated by the execution of the *Simple* java application

Running COMPSs with **log level debug** generates the same files as the info log level but with more detailed information. Additionally, the `jobs` folder contains two files per **submitted** job; one for the `stdout` and another for the `stderr`. In the other hand, the COMPSs Runtime state is printed out on the `stdout`. Figure 15 shows the logs generated by the same execution than the previous cases but with **debug** mode.

The runtime.log and the resources.log files generated in this mode can be **extremely large**. Consequently, the users should take care of their quota and manually erase these files if needed.

When running Python applications a `pycompss.log` file is written inside the *base log folder* containing debug information about the specific calls to PyCOMPSs.

Furthermore, when running `runcompss` with additional flags (such as monitoring or tracing) additional folders will appear inside the *base log folder*. The meaning of the files inside these folders is explained in *Tools*.

Figure 14: resources.log generated by the execution of the *Simple* java application

Figure 15: Structure of the logs folder for the Simple java application in **debug** mode

5.3.1.2 Supercomputers

This section is intended to walk you through the COMPSs usage in Supercomputers.

Executing COMPSs applications

Loading the COMPSs Environment

Depending on the supercomputer installation, COMPSs can be loaded by an environment script, or an Environment Module. The following paragraphs provide the details about how to load the COMPSs environment in the different situations.

COMPSs Environment Script

After a successful installation from the supercomputers package, users can find the *compssenv* script in the folder where COMPSs was installed. This script can be used to load the COMPSs environment in the system as indicated below.

```
$ source <COMPSS_INSTALLATION_DIR>/compssenv
```

COMPSs Environment Module

In BSC supercomputers, COMPSs is configured as an Environment Module. As shown in next Figure, users can type the `module available COMPSs` command to list the supported COMPSs modules in the supercomputer. The users can also execute the `module load COMPSs/<version>` command to load an specific COMPSs module.

```
$ module available COMPSs
---------- /apps/modules/modulefiles/tools ----------
COMPSs/1.3
COMPSs/1.4
COMPSs/2.0
COMPSs/2.1
COMPSs/2.2
COMPSs/2.3
COMPSs/2.4
COMPSs/2.5
COMPSs/2.6
COMPSs/2.7
COMPSs/2.8
COMPSs/2.9
COMPSs/2.10
COMPSs/3.0
COMPSs/3.1
COMPSs/release(default)
COMPSs/trunk


$ module load COMPSs/release
load java/8u131 (PATH, MANPATH, JAVA_HOME, JAVA_ROOT, JAVA_BINDIR, SDK_HOME, JDK_HOME, JRE_
↪HOME)
load papi/5.5.1 (PATH, LD_LIBRARY_PATH, C_INCLUDE_PATH)
load PYTHON/3.7.4 (PATH, MANPATH, LD_LIBRARY_PATH, LIBRARY_PATH, PKG_CONFIG_PATH, C_INCLUDE_
↪PATH, CPLUS_INCLUDE_PATH, PYTHONHOME, PYTHONPATH)
load COMPSs/release (PATH, CLASSPATH, MANPATH, GAT_LOCATION, COMPSS_HOME, JAVA_TOOL_OPTIONS,␣
↪LDFLAGS, CPPFLAGS)
```

(continues on next page)

The following command can be run to check if the correct COMPSs version has been loaded:

```
$ enqueue_compss --version
COMPSs version <version>
```

Configuration Notes

The COMPSs module contains **all** the COMPSs dependencies, including Java, Python and MKL. Modifying any of these dependencies can cause execution failures and thus, we **do not** recomend to change them. Before running any COMPSs job please check your environment and, if needed, comment out any line inside the `.bashrc` file that loads custom COMPSs, Java, Python and/or MKL modules.

The COMPSs environment needs to be loaded in all the nodes that will run COMPSs jobs. Some queue system (such as Slurm) already forward the environment in the allocated nodes. If it is not the case, the `module load` or the `compssenv` script **must** be included in your `.bashrc` file. To do so, please run the following command with the corresponding COMPSs version:

```
$ cat "module load COMPSs/release" >> ~/.bashrc
```

Log out and back in again to check that the file has been correctly edited. The next listing shows an example of the output generated by well loaded COMPSs installation.

```
$ exit
$ ssh USER@SC
load java/8u131 (PATH, MANPATH, JAVA_HOME, JAVA_ROOT, JAVA_BINDIR, SDK_HOME, JDK_HOME, JRE_
↪HOME)
load papi/5.5.1 (PATH, LD_LIBRARY_PATH, C_INCLUDE_PATH)
load PYTHON/3.7.4 (PATH, MANPATH, LD_LIBRARY_PATH, LIBRARY_PATH, PKG_CONFIG_PATH, C_INCLUDE_
↪PATH, CPLUS_INCLUDE_PATH, PYTHONHOME, PYTHONPATH)
load COMPSs/release (PATH, CLASSPATH, MANPATH, GAT_LOCATION, COMPSS_HOME, JAVA_TOOL_OPTIONS,␣
↪LDFLAGS, CPPFLAGS)

USER@SC$ enqueue_compss --version
COMPSs version <version>
```

Important: Please remember that PyCOMPSs uses Python 3.7.4 by default. In order to use another Python version, the requested Python version must be loaded before loading COMPSs, or the environment variable `COMPSS_-PYTHON_VERSION` exported with the requested Python version (available to be loaded from a module).

COMPSs Job submission

COMPSs jobs can be easily submited by running the `enqueue_compss` command. This command allows to configure any **runcompss** (*Runcompss command*) option and some particular queue options such as the queue system, the number of nodes, the wallclock time, the master working directory, the workers working directory and number of tasks per node.

Next, we provide detailed information about the `enqueue_compss` command:

```
$ enqueue_compss -h

Usage: /apps/COMPSs/3.1/Runtime/scripts/user/enqueue_compss [queue_system_options] [COMPSs_
↪options] application_name application_arguments
```

```
* Options:
  General:
    --help, -h                          Print this help message
    --heterogeneous                     Indicates submission is going to be heterogeneous
                                        Default: Disabled

  Queue system configuration:
    --sc_cfg=<name>                     SuperComputer configuration file to use. Must␣
→exist inside queues/cfgs/
                                        Default: default


  Submission configuration:
  General submision arguments:
    --exec_time=<minutes>               Expected execution time of the application (in␣
→minutes)
                                        Default: 10
    --job_name=<name>                   Job name
                                        Default: COMPSs
    --queue=<name>                      Queue/partition name to submit the job. Depends␣
→on the queue system.
                                        Default: default
    --reservation=<name>                Reservation to use when submitting the job.
                                        Default: disabled
    --env_script=<path/to/script>       Script to source the required environment for the␣
→application.
                                        Default: Empty
    --extra_submit_flag=<flag>          Flag to pass queue system flags not supported by␣
→default command flags.
                                        Spaces must be added as '#'
                                        Default: Empty
    --constraints=<constraints>         Constraints to pass to queue system.
                                        Default: disabled
    --qos=<qos>                         Quality of Service to pass to the queue system.
                                        Default: default
    --forward_cpus_per_node=<true|false> Flag to indicate if number to cpus per node must␣
→be forwarded to the worker process.
                                        The number of forwarded cpus will be equal to the␣
→cpus_per_node in a worker node and
                                        equal to the worker_in_master_cpus in a master␣
→node.
                                        Default: false
    --job_dependency=<jobID>            Postpone job execution until the job dependency␣
→has ended.
                                        Default: None
    --forward_time_limit=<true|false>   Forward the queue system time limit to the␣
→runtime.
                                        It will stop the application in a controlled way.
                                        Default: true
    --storage_home=<string>             Root installation dir of the storage␣
→implementation.
                                        Can be defined with the STORAGE_HOME environment␣
→variable.
                                        Default: null
    --storage_props=<string>            Absolute path of the storage properties file
                                        Mandatory if storage_home is defined
  Agents deployment arguments:
```

```
   --agents=<string>                          Hierarchy of agents for the deployment. Accepted␣
→values: plain|tree
                                              Default: tree
   --agents                                   Deploys the runtime as agents instead of the␣
→classic Master-Worker deployment.
                                              Default: disabled


 Homogeneous submission arguments:
   --num_nodes=<int>                          Number of nodes to use
                                              Default: 2
   --num_switches=<int>                       Maximum number of different switches. Select 0␣
→for no restrictions.

                                              Maximum nodes per switch: 18
                                              Only available for at least 4 nodes.
                                              Default: 0
 Heterogeneous submission arguments:
   --type_cfg=<file_location>                 Location of the file with the descriptions of␣
→node type requests

                                              File should follow the following format:
                                              type_X(){
                                                cpus_per_node=24
                                                node_memory=96
                                                ...
                                              }
                                              type_Y(){
                                                ...
                                              }
   --master=<master_node_type>                Node type for the master
                                              (Node type descriptions are provided in the --
→type_cfg flag)
   --workers=type_X:nodes,type_Y:nodes        Node type and number of nodes per type for the␣
→workers

                                              (Node type descriptions are provided in the --
→type_cfg flag)
 Launch configuration:
   --cpus_per_node=<int>                      Available CPU computing units on each node
                                              Default: 48
   --gpus_per_node=<int>                      Available GPU computing units on each node
                                              Default: 0
   --fpgas_per_node=<int>                     Available FPGA computing units on each node
                                              Default: 0
   --io_executors=<int>                       Number of IO executors on each node
                                              Default: 0
   --fpga_reprogram="<string>                 Specify the full command that needs to be␣
→executed to reprogram the FPGA with

                                              the desired bitstream. The location must be an␣
→absolute path.
                                              Default:
   --max_tasks_per_node=<int>                 Maximum number of simultaneous tasks running on a␣
→node

                                              Default: -1
   --node_memory=<MB>                         Maximum node memory: disabled | <int> (MB)
                                              Default: disabled
   --node_storage_bandwidth=<MB>              Maximum node storage bandwidth: <int> (MB)
                                              Default: 450
```

```
    --network=<name>                       Communication network for transfers: default |␣
↪ethernet | infiniband | data.

                                           Default: infiniband


    --prolog="<string>"                    Task to execute before launching COMPSs (Notice␣
↪the quotes)

                                           If the task has arguments split them by ","␣
↪rather than spaces.

                                           This argument can appear multiple times for more␣
↪than one prolog action

                                           Default: Empty
    --epilog="<string>"                    Task to execute after executing the COMPSs␣
↪application (Notice the quotes)

                                           If the task has arguments split them by ","␣
↪rather than spaces.

                                           This argument can appear multiple times for more␣
↪than one epilog action

                                           Default: Empty


    --master_working_dir=<name | path>     Working directory of the application local_disk |␣
↪shared_disk | <path>

                                           Default:
    --worker_working_dir=<name | path>     Worker directory. Use: local_disk | shared_disk |
↪<path>

                                           Default: local_disk


    --worker_in_master_cpus=<int>          Maximum number of CPU computing units that the␣
↪master node can run as worker. Cannot exceed cpus_per_node.
                                           Default: 24
    --worker_in_master_memory=<int> MB     Maximum memory in master node assigned to the␣
↪worker. Cannot exceed the node_memory.

                                           Mandatory if worker_in_master_cpus is specified.
                                           Default: 50000
    --worker_port_range=<min>,<max>        Port range used by the NIO adaptor at the worker␣
↪side

                                           Default: 43001,43005
    --jvm_worker_in_master_opts="<string>" Extra options for the JVM of the COMPSs Worker in␣
↪the Master Node.

                                           Each option separed by "," and without blank␣
↪spaces (Notice the quotes)

                                           Default:
    --container_image=<path>               Runs the application by means of a container␣
↪engine image

                                           Default: Empty
    --container_compss_path=<path>         Path where compss is installed in the container␣
↪image

                                           Default: /opt/COMPSs
    --container_opts="<string>"            Options to pass to the container engine
                                           Default: empty
    --elasticity=<max_extra_nodes>         Activate elasticity specifiying the maximum extra␣
↪nodes (ONLY AVAILABLE FORM SLURM CLUSTERS WITH NIO ADAPTOR)
                                           Default: 0
    --automatic_scaling=<bool>             Enable or disable the runtime automatic scaling␣
↪(for elasticity)

                                           Default: true
    --jupyter_notebook=<path>,             Swap the COMPSs master initialization with␣
↪jupyter notebook from the specified path.
```

```
   --jupyter_notebook                      Default: false
   --ipython                               Swap the COMPSs master initialization with␣
→ipython.

                                           Default: empty


 Runcompss configuration:


 Tools enablers:
   --graph=<bool>, --graph, -g             Generation of the complete graph (true/false)
                                           When no value is provided it is set to true
                                           Default: false
   --tracing=<bool>, --tracing, -t         Set generation of traces.
                                           Default: false
   --monitoring=<int>, --monitoring, -m    Period between monitoring samples (milliseconds)
                                           When no value is provided it is set to 2000
                                           Default: 0
   --external_debugger=<int>,
   --external_debugger                     Enables external debugger connection on the␣
→specified port (or 9999 if empty)
                                           Default: false
   --jmx_port=<int>                        Enable JVM profiling on specified port

 Runtime configuration options:
   --task_execution=<compss|storage>       Task execution under COMPSs or Storage.
                                           Default: compss
   --storage_impl=<string>                 Path to an storage implementation. Shortcut to␣
→setting pypath and classpath. See Runtime/storage in your installation folder.
   --storage_conf=<path>                   Path to the storage configuration file
                                           Default: null
   --project=<path>                        Path to the project XML file
                                           Default: /apps/COMPSs/3.1//Runtime/configuration/
→xml/projects/default_project.xml
   --resources=<path>                      Path to the resources XML file
                                           Default: /apps/COMPSs/3.1//Runtime/configuration/
→xml/resources/default_resources.xml
   --lang=<name>                           Language of the application (java/c/python)
                                           Default: Inferred is possible. Otherwise: java
   --summary                               Displays a task execution summary at the end of␣
→the application execution
                                           Default: false
   --log_level=<level>, --debug, -d        Set the debug level: off | info | api | debug |␣
→trace
                                           Warning: Off level compiles with -O2 option␣
→disabling asserts and __debug__
                                           Default: off


 Advanced options:
   --extrae_config_file=<path>             Sets a custom extrae config file. Must be in a␣
→shared disk between all COMPSs workers.
                                           Default: /apps/COMPSs/3.1//Runtime/configuration/
→xml/tracing/extrae_basic.xml
   --extrae_config_file_python=<path>      Sets a custom extrae config file for python. Must␣
→be in a shared disk between all COMPSs workers.
                                           Default: null
```

```
   --trace_label=<string>                Add a label in the generated trace file. Only␣
↪used in the case of tracing is activated.
                                         Default: None
   --tracing_task_dependencies=<bool>    Adds communication lines for the task␣
↪dependencies (true/false)
                                         Default: false
   --generate_trace=<bool>               Converts the events register into a trace file.␣
↪Only used in the case of activated tracing.
                                         Default: false
   --delete_trace_packages=<bool>        If true, deletes the tracing packages created by␣
↪the run.
                                         Default: false. Automatically, disabled if the␣
↪trace is not generated.
   --custom_threads=<bool>               Threads in the trace file are re-ordered and␣
↪customized to indicate the function of the thread.
                                         Only used when the tracing is activated and a␣
↪trace file generated.
                                         Default: true
   --comm=<ClassName>                    Class that implements the adaptor for␣
↪communications

                                         Supported adaptors:
                                                 ├── es.bsc.compss.nio.master.NIOAdaptor
                                                 └── es.bsc.compss.gat.master.GATAdaptor
                                         Default: es.bsc.compss.nio.master.NIOAdaptor
   --conn=<className>                    Class that implements the runtime connector for␣
↪the cloud

                                         Supported connectors:
                                                 ├── es.bsc.compss.connectors.
↪DefaultSSHConnector

                                                 └── es.bsc.compss.connectors.
↪DefaultNoSSHConnector

                                         Default: es.bsc.compss.connectors.
↪DefaultSSHConnector
   --streaming=<type>                    Enable the streaming mode for the given type.
                                         Supported types: FILES, OBJECTS, PSCOS, ALL, NONE
                                         Default: NONE
   --streaming_master_name=<str>         Use an specific streaming master node name.
                                         Default: null
   --streaming_master_port=<int>         Use an specific port for the streaming master.
                                         Default: null
   --scheduler=<className>               Class that implements the Scheduler for COMPSs
                                         Supported schedulers:
                                                 ├── es.bsc.compss.components.impl.
↪TaskScheduler

                                                 ├── es.bsc.compss.scheduler.orderstrict.
↪fifo.FifoTS

                                                 ├── es.bsc.compss.scheduler.lookahead.fifo.
↪FifoTS

                                                 ├── es.bsc.compss.scheduler.lookahead.lifo.
↪LifoTS

                                                 ├── es.bsc.compss.scheduler.lookahead.
↪locality.LocalityTS

                                                 ├── es.bsc.compss.scheduler.lookahead.
↪successors.constraintsfifo.ConstraintsFifoTS

                                                 ├── es.bsc.compss.scheduler.lookahead.mt.
↪successors.constraintsfifo.ConstraintsFifoTS
```

```
                                                    ├── es.bsc.compss.scheduler.lookahead.
↪successors.fifo.FifoTS
                                                    ├── es.bsc.compss.scheduler.lookahead.mt.
↪successors.fifo.FifoTS
                                                    ├── es.bsc.compss.scheduler.lookahead.
↪successors.lifo.LifoTS
                                                    ├── es.bsc.compss.scheduler.lookahead.mt.
↪successors.lifo.LifoTS
                                                    ├── es.bsc.compss.scheduler.lookahead.
↪successors.locality.LocalityTS
                                                    └── es.bsc.compss.scheduler.lookahead.mt.
↪successors.locality.LocalityTS
                                                Default: es.bsc.compss.scheduler.lookahead.
↪locality.LocalityTS
    --scheduler_config_file=<path>              Path to the file which contains the scheduler␣
↪configuration.
                                                Default: Empty
    --checkpoint=<className>                    Class that implements the Checkpoint Management␣
↪policy

                                                Supported checkpoint policies:
                                                    ├── es.bsc.compss.checkpoint.policies.
↪CheckpointPolicyInstantiatedGroup
                                                    ├── es.bsc.compss.checkpoint.policies.
↪CheckpointPolicyPeriodicTime
                                                    ├── es.bsc.compss.checkpoint.policies.
↪CheckpointPolicyFinishedTasks
                                                    └── es.bsc.compss.checkpoint.policies.
↪NoCheckpoint
                                                Default: es.bsc.compss.checkpoint.policies.
↪NoCheckpoint
    --checkpoint_params=<string>                Checkpoint configuration parameter.
                                                Default: Empty
    --checkpoint_folder=<path>                  Checkpoint folder.
                                                Default: Mandatory parameter
    --library_path=<path>                       Non-standard directories to search for libraries␣
↪(e.g. Java JVM library, Python library, C binding library)
                                                Default: Working Directory
    --classpath=<path>                          Path for the application classes / modules
                                                Default: Working Directory
    --appdir=<path>                             Path for the application class folder.
                                                Default: /home/bscXX/bscXXYYY
    --pythonpath=<path>                         Additional folders or paths to add to the␣
↪PYTHONPATH
                                                Default: /home/bscXX/bscXXYYY
    --env_script=<path>                         Path to the script file where the application␣
↪environment variables are defined.

                                                COMPSs sources this script before running the␣
↪application.
                                                Default: Empty
    --log_dir=<path>                            Directory to store COMPSs log files (a .COMPSs/␣
↪folder will be created inside this location)
                                                Default: User home
    --master_working_dir=<path>                 Use a specific directory to store COMPSs␣
↪temporary files in master
                                                Default: <log_dir>/.COMPSs/<app_name>/tmpFiles
    --uuid=<int>                                Preset an application UUID
```

```
                                            Default: Automatic random generation
    --master_name=<string>                  Hostname of the node to run the COMPSs master
                                            Default:
    --master_port=<int>                     Port to run the COMPSs master communications.
                                            Only for NIO adaptor
                                            Default: [43000,44000]
    --jvm_master_opts="<string>"            Extra options for the COMPSs Master JVM. Each␣
→option separed by "," and without blank spaces (Notice the quotes)
                                            Default:
    --jvm_workers_opts="<string>"           Extra options for the COMPSs Workers JVMs. Each␣
→option separed by "," and without blank spaces (Notice the quotes)
                                            Default: -Xms256m,-Xmx1024m,-Xmn100m
    --cpu_affinity="<string>"               Sets the CPU affinity for the workers
                                            Supported options: disabled, automatic, dlb or␣
→user defined map of the form "0-8/9,10,11/12-14,15,16"
                                            Default: automatic
    --gpu_affinity="<string>"               Sets the GPU affinity for the workers
                                            Supported options: disabled, automatic, user␣
→defined map of the form "0-8/9,10,11/12-14,15,16"
                                            Default: automatic
    --fpga_affinity="<string>"              Sets the FPGA affinity for the workers
                                            Supported options: disabled, automatic, user␣
→defined map of the form "0-8/9,10,11/12-14,15,16"
                                            Default: automatic
    --fpga_reprogram="<string>"             Specify the full command that needs to be␣
→executed to reprogram the FPGA with the desired bitstream. The location must be an absolute␣
→path.
                                            Default:
    --io_executors=<int>                    IO Executors per worker
                                            Default: 0
    --task_count=<int>                      Only for C/Python Bindings. Maximum number of␣
→different functions/methods, invoked from the application, that have been selected as tasks
                                            Default: 50
    --input_profile=<path>                  Path to the file which stores the input␣
→application profile
                                            Default: Empty
    --output_profile=<path>                 Path to the file to store the application profile␣
→at the end of the execution
                                            Default: Empty
    --PyObject_serialize=<bool>             Only for Python Binding. Enable the object␣
→serialization to string when possible (true/false).
                                            Default: false
    --persistent_worker_c=<bool>            Only for C Binding. Enable the persistent worker␣
→in c (true/false).
                                            Default: false
    --enable_external_adaptation=<bool>     Enable external adaptation. This option will␣
→disable the Resource Optimizer.
                                            Default: false
    --gen_coredump                          Enable master coredump generation
                                            Default: false
    --keep_workingdir                       Do not remove the worker working directory after␣
→the execution
                                            Default: false
    --python_interpreter=<string>           Python interpreter to use (python/python3).
                                            Default: python3 Version:
    --python_propagate_virtual_environment=<bool>  Propagate the master virtual environment␣
→to the workers (true/false).
```

```
                                            Default: true
   --python_mpi_worker=<bool>               Use MPI to run the python worker instead of␣
→multiprocessing. (true/false).
                                            Default: false
   --python_memory_profile                 Generate a memory profile of the master.
                                            Default: false
   --python_worker_cache=<string>          Python worker cache (true/size/false).
                                            Only for NIO without mpi worker and python >= 3.8.
                                            Default: false
   --python_cache_profiler=<bool>          Python cache profiler (true/false).
                                            Only for NIO without mpi worker and python >= 3.8.
                                            Default:
   --wall_clock_limit=<int>                Maximum duration of the application (in seconds).
                                            Default: 0
   --shutdown_in_node_failure=<bool>       Stop the whole execution in case of Node Failure.
                                            Default: false
   --provenance, -p                        Generate COMPSs workflow provenance data in RO-
→Crate format from YAML file. Automatically activates -graph and -output_profile.
                                            Default: false

* Application name:
   For Java applications:   Fully qualified name of the application
   For C applications:      Path to the master binary
   For Python applications: Path to the .py file containing the main program

* Application arguments:
   Command line arguments to pass to the application. Can be empty.
```

Tip: For further information about applications scheduling refer to *Schedulers*.

Attention: From COMPSs 2.8 version, the `worker_working_dir` has changed its built-in values to be more generic. The current values are: `local_disk` which substitutes the former `scratch` value; and `shared_disk` which replaces the `gpfs` value.

Attention: From COMPSs 3.1 version:

- the `base_log_dir` has been renamed to `log_dir`.
- the `specific_log_dir` has been removed. Instead, please use the `master_working_dir` in order to define the master temporary files directory.

Caution: Supercomputers may have different partitions in shared disks (e.g. `/gpfs/scratch`, `/gpfs/projects` and `/gpfs/home`).

Consequently, it is **recommended** to set the `log_dir` and `master_working_dir` flags in the same partition as the `worker_working_dir` to avoid performance drop.

Walltime

As with the `runcompss` command, the `enqueue_compss` command also provides the `--wall_clock_limit` for the users to specify the maximum execution time for the application (in seconds). If the time is reached, the execution is stopped.

Do not confuse with `--exec_time`, since `exec_time` indicates the walltime for the queuing system, whilst `wall_-clock_limit` is for COMPSs. Consequently, if the `exec_time` is reached, the queuing system will arise an exception and the execution will be stopped suddenly (potentially causing loose of data). However, if the `wall_clock_limit` is reached, the COMPSs runtime stops and grabs all data safely.

Tip: It is a good practice to define the `--wall_clock_limit` with less time than defined for `--exec_time`, so that the COMPSs runtime can stop the execution safely and ensure that no data is lost.

PyCOMPSs within interactive jobs

PyCOMPSs can be used in interactive jobs through the use of ipython. To this end, the first thing is to request an interactive job. For example, an interactive job with Slurm for one node with 48 cores (as in MareNostrum 4) can be requested as follows:

```
$ salloc --qos=debug -N1 -n48

salloc: Pending job allocation 12189081
salloc: job 12189081 queued and waiting for resources
salloc: job 12189081 has been allocated resources
salloc: Granted job allocation 12189081
salloc: Waiting for resource configuration
salloc: Nodes s02r2b27 are ready for job
```

When the job starts running, the terminal directly opens within the given node.

Then, it is necessary to start the COMPSs infrastructure in the given nodes. To this end, the following command will start one worker with 24 cores (default worker in master), and then launch the *ipython* interpreter:

```
$ launch_compss \
  --sc_cfg=mn.cfg \
  --master_node="$SLURMD_NODENAME" \
  --worker_nodes="" \
  --ipython \
  --pythonpath=$(pwd) \
  "dummy"
```

Note that the *launch_ compss* command requires the supercomputing configuration file, which in the MareNostrum 4 case is *mn.cfg* (more information about the supercomputer configuration can be found in *Configuration Files*). In addition, requires to define which node is going to be the master, and which ones the workers (if none, takes the default worker in master). Finally, the *–ipython* flag indicates that use ipython is expected.

When ipython is started, the COMPSs infrastructure is ready, and the user can start running interactive commands considering the PyCOMPSs API for jupyter notebook (see Jupyter *API calls*).

MareNostrum 4

Basic queue commands

The MareNostrum supercomputer uses the SLURM (Simple Linux Utility for Resource Management) workload manager. The basic commands to manage jobs are listed below:

- **sbatch** Submit a batch job to the SLURM system
- **scancel** Kill a running job
- **squeue -u <username>** See the status of jobs in the SLURM queue

For more extended information please check the *SLURM: Quick start user guide* at https://slurm.schedmd.com/quickstart.html .

Tracking COMPSs jobs

When submitting a COMPSs job a temporal file will be created storing the job information. For example:

```
$ enqueue_compss \
  --exec_time=15 \
  --num_nodes=3 \
  --cpus_per_node=16 \
  --master_working_dir=$(pwd) \
  --worker_working_dir=shared_disk \
  --lang=python \
  --log_level=debug \
  <APP> <APP_PARAMETERS>


SC Configuration:        default.cfg
Queue:                   default
Reservation:             disabled
Num Nodes:               3
Num Switches:            0
GPUs per node:           0
Job dependency:          None
Exec-Time:               00:15
Storage Home:            null
Storage Properties:      null
Other:
        --sc_cfg=default.cfg
        --cpus_per_node=48
        --master_working_dir=/path/to/app_dir
        --worker_working_dir=shared_disk
        --lang=python
        --classpath=.
        --library_path=.
        --comm=es.bsc.compss.nio.master.NIOAdaptor
        --tracing=false
        --graph=false
        --pythonpath=.
        <APP> <APP_PARAMETERS>
Temp submit script is: /scratch/tmp/tmp.pBG5yfFxEo

$ cat /scratch/tmp/tmp.pBG5yfFxEo
#!/bin/bash
#
```

(continues on next page)

```
#SBATCH --job-name=COMPSs
#SBATCH --workdir=.
#SBATCH -o compss-%J.out
#SBATCH -e compss-%J.err
#SBATCH -N 3
#SBATCH -n 144
#SBATCH --exclusive
#SBATCH -t00:15:00
...
```

> **Caution:** Since MN4 has different partitions in shared disk (gpfs): `/gpfs/scratch`, `/gpfs/projects` and `/gpfs/home`, it is **recommended** to set the `log_dir` and `master_working_dir` flags in the same partition as the `worker_working_dir` to avoid performance drop.

In order to track the jobs state users can run the following command:

```
$ squeue
JOBID    PARTITION  NAME    USER  TIME_LEFT  TIME_LIMIT  START_TIME  ST NODES  CPUS  NODELIST
474130    main     COMPSs   XX    0:15:00    0:15:00       N/A       PD   3    144   -
```

The specific COMPSs logs are stored under the `~/.COMPSs/` folder; saved as a local *runcompss* execution. For further details please check the *Executing COMPSs applications* Section.

MinoTauro

Basic queue commands

The MinoTauro supercomputer uses the SLURM (Simple Linux Utility for Resource Management) workload manager. The basic commands to manage jobs are listed below:

- **sbatch** Submit a batch job to the SLURM system
- **scancel** Kill a running job
- **squeue -u <username>** See the status of jobs in the SLURM queue

For more extended information please check the *SLURM: Quick start user guide* at https://slurm.schedmd.com/quickstart.html .

Tracking COMPSs jobs

When submitting a COMPSs job a temporal file will be created storing the job information. For example:

```
$ enqueue_compss \
  --exec_time=15 \
  --num_nodes=3 \
  --cpus_per_node=16 \
  --master_working_dir=. \
  --worker_working_dir=shared_disk \
  --lang=python \
  --log_level=debug \
  <APP> <APP_PARAMETERS>


SC Configuration:          default.cfg
```

```
Queue:                  default
Reservation:            disabled
Num Nodes:              3
Num Switches:           0
GPUs per node:          0
Job dependency:         None
Exec-Time:              00:15
Storage Home:           null
Storage Properties:     null
Other:
        --sc_cfg=default.cfg
        --cpus_per_node=16
        --master_working_dir=.
        --worker_working_dir=shared_disk
        --lang=python
        --classpath=.
        --library_path=.
        --comm=es.bsc.compss.nio.master.NIOAdaptor
        --tracing=false
        --graph=false
        --pythonpath=.
        <APP> <APP_PARAMETERS>
Temp submit script is: /scratch/tmp/tmp.pBG5yfFxEo


$ cat /scratch/tmp/tmp.pBG5yfFxEo
#!/bin/bash
#
#SBATCH --job-name=COMPSs
#SBATCH --workdir=.
#SBATCH -o compss-%J.out
#SBATCH -e compss-%J.err
#SBATCH -N 3
#SBATCH -n 48
#SBATCH --exclusive
#SBATCH -t00:15:00
...
```

In order to trac the jobs state users can run the following command:

```
$ squeue
JOBID  PARTITION   NAME    USER  ST  TIME    NODES  NODELIST (REASON)
XXXX   projects    COMPSs  XX    R   00:02       3  nvb[6-8]
```

The specific COMPSs logs are stored under the `~/.COMPSs/` folder; saved as a local *runcompss* execution. For further details please check the *Executing COMPSs applications* Section.

Nord 3

Basic queue commands

The Nord3 supercomputer uses the LSF (Load Sharing Facility) workload manager. The basic commands to manage jobs are listed below:

- **bsub** Submit a batch job to the LSF system
- **bkill** Kill a running job
- **bjobs** See the status of jobs in the LSF queue
- **bqueues** Information about LSF batch queues

For more extended information please check the *IBM Platform LSF Command Reference* at https://www.ibm.com/support/knowledgecenter/en/SSETD4_9.1.2/lsf_kc_cmd_ref.html .

Tracking COMPSs jobs

When submitting a COMPSs job a temporal file will be created storing the job information. For example:

```
$ enqueue_compss \
  --exec_time=15 \
  --num_nodes=3 \
  --cpus_per_node=16 \
  --master_working_dir=. \
  --worker_working_dir=shared_disk \
  --lang=python \
  --log_level=debug \
  <APP> <APP_PARAMETERS>


SC Configuration:        default.cfg
Queue:                   default
Reservation:             disabled
Num Nodes:               3
Num Switches:            0
GPUs per node:           0
Job dependency:          None
Exec-Time:               00:15
Storage Home:            null
Storage Properties:      null
Other:
        --sc_cfg=default.cfg
        --cpus_per_node=16
        --master_working_dir=.
        --worker_working_dir=shared_disk
        --lang=python
        --classpath=.
        --library_path=.
        --comm=es.bsc.compss.nio.master.NIOAdaptor
        --tracing=false
        --graph=false
        --pythonpath=.
        <APP> <APP_PARAMETERS>
Temp submit script is: /scratch/tmp/tmp.pBG5yfFxEo

$ cat /scratch/tmp/tmp.pBG5yfFxEo
#!/bin/bash
```

(continues on next page)

```
#
#BSUB -J COMPSs
#BSUB -cwd .
#BSUB -oo compss-%J.out
#BSUB -eo compss-%J.err
#BSUB -n 3
#BSUB -R "span[ptile=1]"
#BSUB -W 00:15
...
```

In order to trac the jobs state users can run the following command:

```
$ bjobs
JOBID   USER    STAT   QUEUE   FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
XXXX    bscXX   PEND   XX      login1      XX          COMPSs     Month Day Hour
```

The specific COMPSs logs are stored under the `~/.COMPSs/` folder; saved as a local *runcompss* execution. For further details please check the *Executing COMPSs applications* Section.

Enabling COMPSs Monitor

Configuration

As supercomputer nodes are connection restricted, the better way to enable the *COMPSs Monitor* is from the users local machine. To do so please install the following packages:

- COMPSs Runtime
- COMPSs Monitor
- sshfs

For further details about the COMPSs packages installation and configuration please refer to *Installation and Administration* Section. If you are not willing to install COMPSs in your local machine please consider to download our Virtual Machine available at our webpage.

Once the packages have been installed and configured, users need to mount the sshfs directory as follows. The `SC_USER` stands for your supercomputer's user, the `SC_ENDPOINT` to the supercomputer's public endpoint and the `TARGET_LOCAL_FOLDER` to the local folder where you wish to deploy the supercomputer files):

```
compss@bsc:~$ scp $HOME/.ssh/id_rsa.pub ${SC_USER}@mn1.bsc.es:~/id_rsa_local.pub
compss@bsc:~$ ssh SC_USER@SC_ENDPOINT \
                "cat ~/id_rsa_local.pub >> ~/.ssh/authorized_keys; \
                rm ~/id_rsa_local.pub"
compss@bsc:~$ mkdir -p TARGET_LOCAL_FOLDER/.COMPSs
compss@bsc:~$ sshfs -o IdentityFile=$HOME/.ssh/id_rsa -o allow_other \
                SC_USER@SC_ENDPOINT:~/.COMPSs \
                TARGET_LOCAL_FOLDER/.COMPSs
```

Whenever you wish to unmount the sshfs directory please run:

```
compss@bsc:~$ sudo umount TARGET_LOCAL_FOLDER/.COMPSs
```

Execution

Access the COMPSs Monitor through its webpage (http://localhost:8080/compss-monitor by default) and log in with the `TARGET_LOCAL_FOLDER` to enable the COMPSs Monitor for MareNostrum.

Please remember that to enable **all** the COMPSs Monitor features applications must be ran with the *-m* flag. For further details please check the *Executing COMPSs applications* Section.

Figure 16 illustrates how to login and Figure 17 shows the COMPSs Monitor main page for an application run inside a Supercomputer.
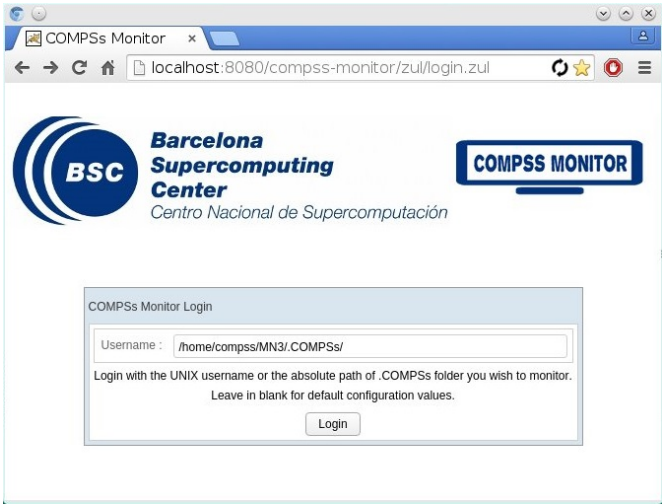
Figure 16: COMPSs Monitor login for Supercomputers
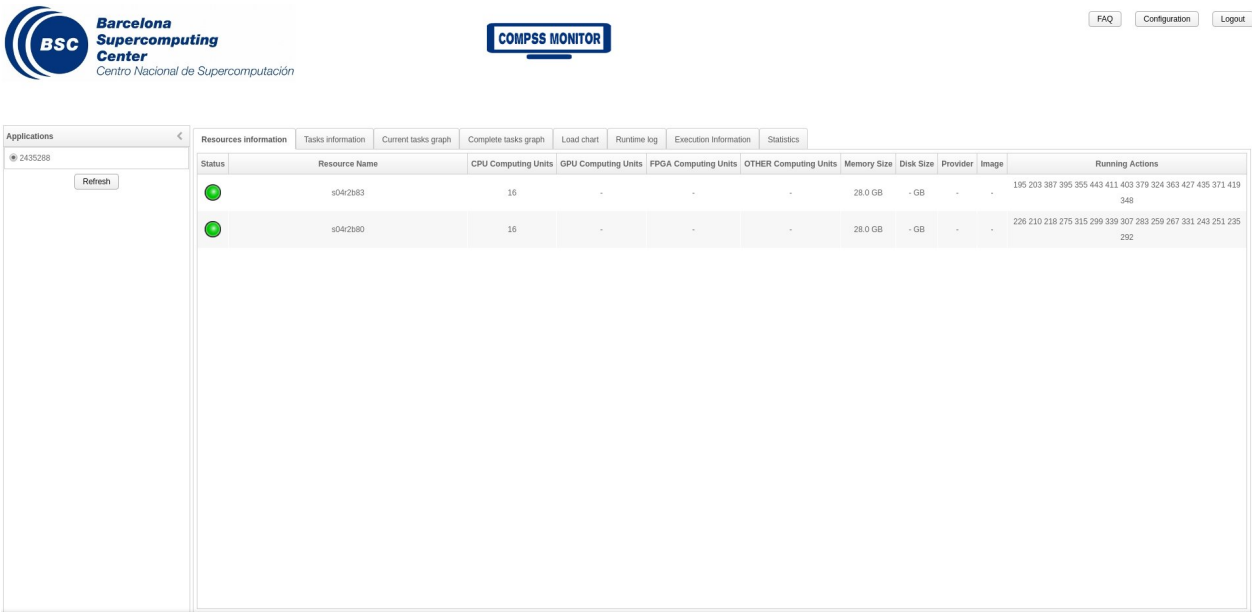
Figure 17: COMPSs Monitor main page for a test application at Supercomputers

5.3.1.3 Docker

What is Docker?

Docker is an open-source project that automates the deployment of applications inside software containers, by providing an additional layer of abstraction and automation of operating-system-level virtualization on Linux. In addition to the Docker container engine, there are other Docker tools that allow users to create complex applications (Docker-Compose) or to manage a cluster of Docker containers (Docker Swarm).

COMPSs supports running a distributed application in a Docker Swarm cluster.

Requirements

In order to use COMPSs with Docker, some requirements must be fulfilled:

- Have **Docker** and **Docker-Compose** installed in your local machine.
- Have an available **Docker Swarm cluster** and its Swarm manager ip and port to access it remotely.
- A **Dockerhub account**. Dockerhub is an online repository for Docker images. We don't currently support another sharing method besides uploading to Dockerhub, so you will need to create a personal account. This has the advantage that it takes very little time either upload or download the needed images, since it will reuse the existing layers of previous images (for example the COMPSs base image).

Execution in Docker

The runcompss-docker execution workflow uses Docker-Compose, which is in charge of spawning the different application containers into the Docker Swarm manager. Then the Docker Swarm manager schedules the containers to the nodes and the application starts running. The COMPSs master and workers will run in the nodes Docker Swarm decides. To see where the masters and workers are located in runtime, you can use:

```
$ docker -H '<swarm_manager_ip:swarm_port>' ps -a
```

The execution of an application using Docker containers with COMPSs **consists of 2 steps**:

Execution step 1: Creation of the application image

The very first step to execute a COMPSs application in Docker is creating your application Docker image.

This must be done **only once** for every new application, and then you can run it as many times as needed. If the application is updated for whatever reason, this step must be done again to create and share the updated image.

In order to do this, you must use the **compss_docker_gen_image** tool, which is available in the standard COMPSs application. This tool is the responsible of taking your application, create the needed image, and upload it to Dockerhub to share it.

The image is created injecting your application into a COMPSs base image. This base image is available in Dockerhub. In case you need it, you can pull it using the following command:

```
$ docker pull compss/compss
```

The **compss_docker_gen_image** script receives 2 parameters:

> **--c, --context-dir** Specifies the **context directory** path of the application. This path **MUST BE ABSOLUTE**, not relative. The context directory is a local directory that **must contain the needed binaries and input files of the app (if any)**. In its simplest case, it will contain the executable file (a .jar for example). Keep the context-directory as lightest as possible.
>
> For example: –**context-dir**='/**home**/**compss-user**/**my-app-dir**' (where 'my-app-dir' contains 'app.jar', 'data1.dat' and 'data2.csv'). For more details,

this context directory will be recursively copied into a COMPSs base image. Specifically, it will create all the path down to the context directory inside the image.

--image-name   Specifies a name for the created image. It **MUST** have this format: 'DOCKERHUB-USERNAME/image-name'. The *DOCKERHUB_USERNAME* must be the username of your personal Dockerhub account. The *image_name* can be whatever you want, and will be used as the identifier for the image in Dockerhub. This name will be the one you will use to execute the application in Docker. For example, if my Dockerhub username is john123 and I want my image to be named "my-image-app": `--image-name="john123/my-image-app"`.

As stated before, this is needed to share your container application image with the nodes that need it. Image tags are also supported (for example "john123/my-image-app:1.23).

Important: After creating the image, be sure to write down the absolute context-directory and the absolute classpath (the absolute path to the executable jar). You will need it to run the application using `runcompss-docker`. In addition, if you plan on distributing the application, you can use the Dockerhub image's information tab to write them, so the application users can retrieve them.

Execution step 2: Run the application

To execute COMPSs in a Docker Swarm cluster, you must use the `runcompss-docker` command, instead of `runcompss`.

The command `runcompss-docker` has some **additional arguments** that will be needed by COMPSs to run your application in a distributed Docker Swarm cluster environment. The rest of typical arguments (classpath for example) will be delegated to runcompss command.

These additional arguments must go before the typical runcompss arguments. The runcompss-docker additional arguments are:

--w, --worker-containers   Specifies the number of **worker containers** the app will execute on. One more container will be created to host the **master**. If you have enough nodes in the Swarm cluster, each container will be executed by one node. This is the default schedule strategy used by Swarm. For example: `--worker-containers=3`

--s, --swarm-manager   Specifies the Swarm manager ip and port (format: ip:port). For example: `--swarm-manager='129.114.108.8:4000'`

--i, --image-name   Specify the image name of the application image in Dockerhub. Remember you must generate this with compss_docker_gen_image Remember as well that the format must be: 'DOCKERHUB_USERNAME/APP_IMAGE_-NAME:TAG' (the :TAG is optional). For example: `--image-name='john123/my-compss-application:1.9'`

--c, --context-dir   Specifies the **context directory** of the app. It must be specified by the application image provider. For example: `--context-dir='/home/compss-user/my-app-context-dir'`

As **optional** arguments:

--c-cpu-units   Specifies the number of cpu units used by each container (default value is 4). For example: `*--c-cpu-units:=16`

--c-memory   Specifies the physical memory used by each container in GB (default value is 8 GB). For example, in this case, each container would use as maximum 32 GB of physical memory: `--c-memory=32`

Here is the **format** you must use with `runcompss-docker` command:

```
$ runcompss-docker --worker-containers=N \
                   --swarm-manager='<ip>:<port>' \
                   --image-name='DOCKERHUB_USERNAME/image_name' \
                   --context-dir='CTX_DIR' \
                   [rest of classic runcompss args]
```

Or alternatively, in its shortest form:

```
$ runcompss-docker --w=N --s='<ip>:<port>' --i='DOCKERHUB_USERNAME/image_name' --c='CTX_DIR' \
                   [rest of classic runcompss args]
```

Execution with TLS

If your cluster uses **TLS** or has been created using **Docker-Machine**, you will have to **export two environment variables** before using runcompss-docker:

On one hand, **DOCKER_TLS_VERIFY** environment variable will tell Docker that you are using TLS:

```
export DOCKER_TLS_VERIFY="1"
```

On the other hand, **DOCKER_CERT_PATH** variable will tell Docker where to find your TLS certificates. As an example:

```
export DOCKER_CERT_PATH="/home/compss-user/.docker/machine/machines/my-manager-node"
```

In case you have created your cluster using docker-machine, in order to know what your *DOCKER_CERT_PATH* is, you can use this command:

```
$ docker-machine env my-swarm-manager-node-name | grep DOCKER_CERT_PATH
```

In which *swarm-manager-node-name* must be changed by the name docker-machine has assigned to your swarm manager node. With these environment variables set, you are ready to use `runcompss-docker` in a cluster using TLS.

Execution results

The execution results will be retrieved from the master container of your application.

If your context-directory name is **'matmul'**, then your results will be saved in the **'matmul-results'** directory, which will be located in the same directory you executed runcompss-docker on.

Inside the **'matmul-results'** directory you will have:

- A folder named **'matmul'** with all the result files that were in the same directory as the executable when the application execution ended. More precisely, this will contain the context-directory state right after finishing your application execution. Additionally, and for more advanced debug purposes, you will have some intermediate files created by runcompss-docker (Dockerfile, project.xml, resources.xml), in case you want to check for more complex errors or details.
- A folder named **'debug'**, which (in case you used the runcompss debug option (**-d**)), will contain the **'.COMPSs'** directory, which contains another directory in which there are the typical debug files runtime.log, jobs, etc. Remember **.COMPSs** is a **hidden** directory, take this into account if you do **ls** inside the debug directory (add the **-a** option).

To make it simpler, we provide a **tree visualization** of an example of what your directories should look like after the execution. In this case we executed the **Matmul example application** that we provide you:

```
.
├── matmul
│   └── matmul.jar
└── matmul-results
    ├── debug
    │   └── .COMPSs
    │       └── matmul.files.Matmul_01
    │           ├── jobs
    │           │   ├── job1_NEW.err
    │           │   ├── job1_NEW.out
    │           │   ├── job2_NEW.err
    │           │   ├── job2_NEW.out
    │           │   ├── job3_NEW.err
    │           │   └── job3_NEW.out
    │           ├── resources.log
    │           ├── runtime.log
    │           └── tmpFiles
    └── matmul
        ├── A.0.0
        ├── A.0.1
        ├── A.0.2
        ├── B.0.0
        ├── B.0.1
        ├── B.0.2
        ├── Dockerfile
        ├── matmul.jar
        ├── project.xml
        └── resources.xml
```
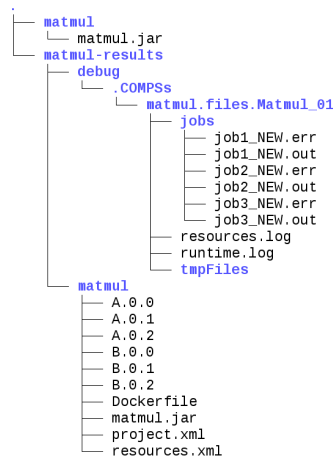
Figure 18: Result and log folders of a *Matmul* execution with COMPSs and Docker

Execution examples

Next we will use the *Matmul* application as an example of a Java application running with COMPSs and Docker.

Imagine we have our Matmul application in `/home/john/matmul` and inside the `matmul` directory we only have the file `matmul.jar`.

We have created a Dockerhub account with username 'john123'.

The first step will be creating the image:

```
$ compss_docker_gen_image --context-dir='/home/john/matmul' \
                          --image-name='john123/matmul-example'
```

Now, we write down the context-dir (`/home/john/matmul`) and the classpath (`/home/john/matmul/matmul.jar`). We do this because they will be needed for future executions. Since the image is created and uploaded, we won't need to do this step anymore.

Now we are going to execute our Matmul application in a Docker cluster.

Take as assumptions:

- We will use **5 worker docker containers**.
- The **swarm-manager ip** will be 129.114.108.8, with the Swarm manager listening to the **port** 4000.
- We will use **debug (-d)**.
- Finally, as we would do with the typical runcompss, we specify the **main class** name and its **parameters** (16 and 4 in this case).

In addition, we know from the former step that the image name is `john123/matmul-example`, the **context directory** is `/home/john/matmul`, and the classpath is `/home/john/matmul/matmul.jar`. And this is how you would run `runcompss-docker`:

```
$ runcompss-docker --worker-containers=5 \
                   --swarm-manager='129.114.108.8:4000' \
                   --context-dir='/home/john/matmul' \
                   --image-name='john123/matmul-example' \
                   --classpath=/home/john/matmul/matmul.jar \
                   -d \
                   matmul.objects.Matmul 16 4
```

Here we show another example using the short arguments form, with the KMeans example application, that is also provided as an example COMPSs application to you:

First step, create the image once:

```
$ compss_docker_gen_image --context-dir='/home/laura/apps/kmeans' \
                          --image-name='laura-67/my-kmeans'
```

And now execute with 30 worker containers, and Swarm located in '110.3.14.159:26535'.

```
$ runcompss-docker --w=30 \
                   --s='110.3.14.159:26535' \
                   --c='/home/laura/apps/kmeans' \
                   --image-name='laura-67/my-kmeans' \
                   --classpath=/home/laura/apps/kmeans/kmeans.jar \
                   kmeans.KMeans
```

5.3.1.4 Chameleon

What is Chameleon?

The Chameleon project is a configurable experimental environment for large-scale cloud research based on a *OpenStack* KVM Cloud. With funding from the *National Science Foundation (NSF)*, it provides a large-scale platform to the open research community allowing them explore transformative concepts in deeply programmable cloud services, design, and core technologies. The Chameleon testbed, is deployed at the *University of Chicago* and the *Texas Advanced Computing Center* and consists of 650 multi-core cloud nodes, 5PB of total disk space, and leverage 100 Gbps connection between the sites.

The project is led by the *Computation Institute* at the *University of Chicago* and partners from the *Texas Advanced Computing Center* at the *University of Texas* at Austin, the *International Center for Advanced Internet Research* at *Northwestern University*, the *Ohio State University*, and *University of Texas* at *San Antoni*, comprising a highly qualified and experienced team. The team includes members from the *NSF* supported *FutureGrid* project and from the *GENI* community, both forerunners of the *NSFCloud* solicitation under which this project is funded. Chameleon will also sets of partnerships with commercial and academic clouds, such as *Rackspace*, *CERN* and *Open Science Data Cloud (OSDC)*.

For more information please check https://www.chameleoncloud.org/ .

Execution in Chameleon

Currently, COMPSs can only handle the Chameleon infrastructure as a cluster (deployed inside a lease). Next, we provide the steps needed to execute COMPSs applications at Chameleon:

- Make a lease reservation with 1 minimum node (for the COMPSs master instance) and a maximum number of nodes equal to the number of COMPSs workers needed plus one
- Instantiate the master image (based on the published image *COMPSs_ _ CC-CentOS7*)
- Attach a public IP and login to the master instance (the instance is correctly contextualized for COMPSs executions if you see a COMPSs login banner)
- Set the instance as COMPSs master by running `/etc/init.d/chameleon_init start`
- Copy your CH file (API credentials) to the Master and source it
- Run the `chameleon_cluster_setup` script and fill the information when prompted (you will be asked for the name of the master instance, the reservation id and number of workers). This scripts may take several minutes since it sets up the all cluster.
- Execute your COMPSs applications normally using the `runcompss` script

As an example you can check this video https://www.youtube.com/watch?v=BrQ6anPHjAU performing a full setup and execution of a COMPSs application at Chameleon.

5.3.1.5 Jupyter Notebook

Notebook execution

The jupyter notebook can be executed as a common Jupyter notebook by steps or the whole application.

Important: A message showing the failed task/s will pop up if an exception within them happens.

This pop up message will also allow you to continue the execution without PyCOMPSs, or to restart the COMPSs runtime. Please, note that in the case of COMPSs restart, the tracking of some objects may be lost (will need to be recomputed).

Notebook example

Sample notebooks can be found in the *PyCOMPSs Notebooks* Section.

Tips and Tricks

Tasks information

It is possible to show task related information with `tasks_info` function.

```python
# Previous user code

import pycompss.interactive as ipycompss
ipycompss.start(graph=True)

# User code that calls tasks

# Check the current tasks info
ipycompss.tasks_info()

ipycompss.stop(sync=True)

# Subsequent code
```

Important: The tasks information will not be displayed if the `monitor` option at `ipycompss.start` is not set (to a refresh value).

The `tasks_info` function provides a widget that can be updated while running other cells from the notebook, and will keep updating every second until stopped. Alternatively, it will show a snapshot of the tasks information status if ipywidgets is not available.

The information displayed is composed by two plots: the left plot shows the average time per task, while the right plot shows the amount of tasks. Then, a table with the specific number of number of executed tasks, maximum execution time, mean execution time and minimum execution time, per task is shown.

Tasks status

It is possible to show task status (running or completed) tasks with the `tasks_status` function.

```python
# Previous user code

import pycompss.interactive as ipycompss
ipycompss.start(graph=True)

# User code that calls tasks

# Check the current tasks info
ipycompss.tasks_status()

ipycompss.stop(sync=True)

# Subsequent code
```

Important: The tasks information will not be displayed if the `monitor` option at `ipycompss.start` is not set (to a refresh value).

The `tasks_status` function provides a widget that can be updated while running other cells from the notebook, and will keep updating every second until stopped. Alternatively, it will show a snapshot of the tasks status if ipywidgets is not available.

The information displayed is composed by a pie chart and a table showing the number of running tasks, and the number of completed tasks.

Resources status

It is possible to show resources status with the `resources_status` function.

```python
# Previous user code

import pycompss.interactive as ipycompss
ipycompss.start(graph=True)

# User code that calls tasks

# Check the current tasks info
ipycompss.resources_status()

ipycompss.stop(sync=True)

# Subsequent code
```

Important: The tasks information will not be displayed if the `monitor` option at `ipycompss.start` is not set (to a refresh value).

The `resources_status` function provides a widget that can be updated while running other cells from the notebook, and will keep updating every second until stopped. Alternatively, it will show a snapshot of the resources status if ipywidgets is not available.

The information displayed is a table showing the number of computing units, gpus, fpgas, other computing units, amount of memory, amount of disk, status and actions.

Current task graph

It is possible to show the current task graph with the `current_task_graph` function.

```
# Previous user code

import pycompss.interactive as ipycompss
ipycompss.start(graph=True)

# User code that calls tasks

# Check the current task graph
ipycompss.current_task_graph()

ipycompss.stop(sync=True)

# Subsequent code
```

Important: The graph will not be displayed if the `graph` option at `ipycompss.start` is not set to `true`.

In addition, the `current_task_graph` has some options. Specifically, its full signature is:

```
current_task_graph(fit=False, refresh_rate=1, timeout=0)
```

Parameters:

- `fit` Adjust the size to the available space in jupyter if set to true. Display full size if set to false (default).
- `refresh_rate` When *timeout* is set to a value different from 0, it defines the number of seconds between graph refresh.
- `timeout` Check the current task graph during the *timeout* value (seconds). During the *timeout* value, it refresh the graph considering the *refresh_rate* value. It can be stopped with the stop button of Jupyter. Does not update the graph if set to 0 (default).

Caution: The graph can be empty if all pending tasks have been completed.

Complete task graph

It is possible to show the complete task graph with the `complete_task_graph` function.

```
# Previous user code

import pycompss.interactive as ipycompss
ipycompss.start(graph=True)

# User code that calls tasks

# Check the current task graph
ipycompss.complete_task_graph()

ipycompss.stop(sync=True)

# Subsequent code
```

Important: The graph will not be displayed if the `graph` option at `ipycompss.start` is not set to `true`.

In addition, the `complete_task_graph` has some options. Specifically, its full signature is:

```
complete_task_graph(fit=False, refresh_rate=1, timeout=0)
```

Parameters:

> **fit** Adjust the size to the available space in jupyter if set to true. Display full size if set to false (default).
>
> **refresh_rate** When *timeout* is set to a value different from 0, it defines the number of seconds between graph refresh.
>
> **timeout** Check the current task graph during the *timeout* value (seconds). During the *timeout* value, it refresh the graph considering the *refresh_rate* value. It can be stopped with the stop button of Jupyter. Does not update the graph if set to 0 (default).

Caution: The graph may be empty or raise an exception if the graph has not been updated by the runtime (may happen if there are too few tasks). In this situation, stop the compss runtime (synchronizing the remaining objects if intended to start the runtime afterwards) and try again.

5.3.2 Agents Deployments

Opposing to well-established deployments with an almost-static set of computing resources and hardly-varying interconnection conditions such as a single-computer, a cluster or a supercomputer; dynamic infrastructures, like Fog environments, require a different kind of deployment able to adapt to rapidly-changing conditions. Such infrastructures are likely to comprise several mobile devices whose connectivity to the infrastructure is temporary. When the device is within the network range, it joins an already existing COMPSs deployment and interacts with the other resources to offload tasks onto them or viceversa. Eventually, the connectivity of that mobile device could be disrupted to never reestablish. If the leaving device was used as a worker node, the COMPSs master needs to react to the departure and reassign the tasks running on that node. If the device was the master node, it should be able to carry on with the computation being isolated from the rest of the infrastructure or with another set of available resources.

COMPSs Agents is a deployment approach especially designed to fit in this kind of environments. Each device is an autonomous individual with processing capabilities hosting the execution of a COMPSs runtime as a background service. Applications - running on that device or on another - can contact this service to request the execution of a function in a serverless, stateless manner (resembling the Function-as-a-Service model). If the requested function follows the COMPSs programming model, the runtime will parallelise its execution as if it were the main function of a regular COMPSs application.

Agents can associate with other agents by offering their embedded computing resources to execute functions to achieve a greater purpose; in exchange, they receive a platform where they can offload their computation in the same manner, and, thus, achieve lower response times. As opossed to the master-worker approach followed by the classic COMPSs deployment, where a single node produces the all the workload, in COMPSs Agents deployments, any of the nodes within the platform becomes a potential source of computation to distribute. Therefore, this master-centric approach where workload producer to orchestrate holistically the execution is no longer valid. Besides, concentrating all the knowledge of several applications and handling the changes of infrastructure represents an important computational burden for the resource assuming the master role, especially if it is a resource-scarce device like a mobile. For this two reasons, COMPSs agents proposes a hierachic approach to organize the nodes. Each node will only be aware of some devices with which it has direct connection and only decides whether the task runs on its embedded computing devices or if the responsability of executing the task is delegated onto one of the other agents. In the latter case, the receiver node will face the same problem and decide whether it should host the execution or forward it to a different node.

The following image illustrates an example of a COMPSs agents hierarchy that could be deployed in any kind of facilities; for instance, a university campus. In this case, students only interact directly with their mobile phones

---

and laptops to run their applications; however, the computing workload produced by them is distributed across the whole system. To do so, the mobile devices need to connect to one of the edge devices devices scattered across the facilities acting as a Wi-Fi Hotspot (in the example, raspberry Pi) which runs a COMPSs agent. To submit the operation execution to the platform, mobile devices can either contact a COMPSs agent running in the device or the application can directly contact the remote agent running on the rPI. All rPi agents are connected to an on-premise server within the campus that also runs a COMPSs Agent. Upon an operation request by a user device, the rPi can host the computation on its own devices or forward the request to one of its neighbouring agents: the on-premise server or another user's device running a COMPSs agent. In the case that the rPi decides to move up the request through the hierarchy, the on-premise server faces a similar problem: hosting the computation on its local devices, delegating the execution onto one of the rPi – which in turn could forward the execution back to another user's device –, or submit the request to a cloud. Internally, the Cloud can also be organized with COMPSs Agents hierarchy; thus, one of its nodes can act as the gateway to receive external requests and share the workload across the whole system.
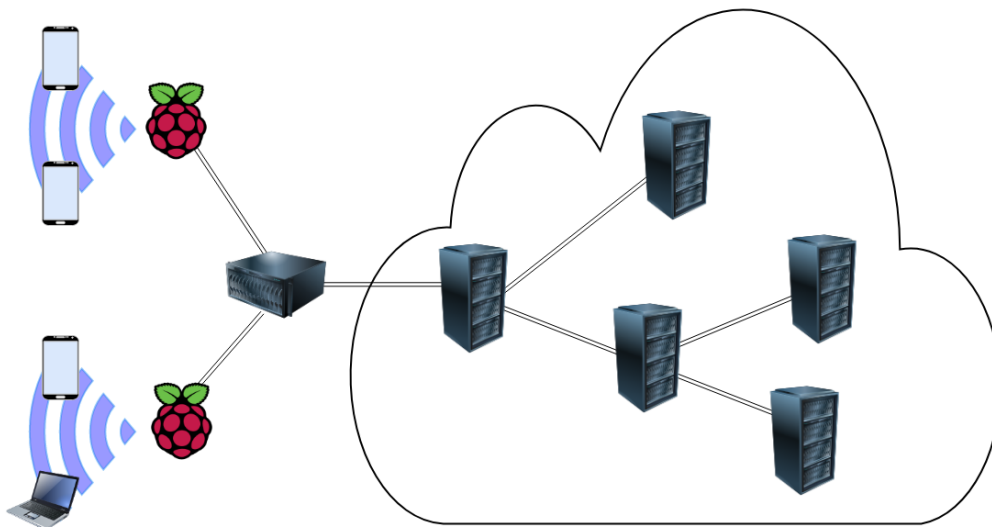
5.3.2.1 Local

This section is intended to show how to execute COMPSs applications deploying the runtime as an agent in local machines.

Deploying a COMPSs Agent

COMPSs Agents are deployed using the **compss_agent_start** command:

```
compss@bsc:~$ compss_agent_start [OPTION]
```

There is one mandatory parameter `--hostname` that indicates the name that other agents and itself use to refer to the agent. Bear in mind that agents are not able to dynamically modify its classpath; therefore, the `--classpath` parameter becomes important to indicate the application available on the agent. Any public method available on the classpath is an execution request candidate.

The following command raises an agent with name 192.168.1.100 and any of the public methods of the classes encapsulated in the jarfile `/app/path.jar` can be executed.

```
compss@bsc:~$ compss_agent_start  --hostname=192.168.1.100 --classpath=/app/path.jar
```

The `compss_agent_start` command allows users to set up the COMPSs runtime by specifying different options in the same way as done for the `runcompss` command. To indicate the available resources, the device administrator can use the `--project` and `--resources` option exactly in the same way as for the `runcompss` command. For

further details on how to dynamically modify the available resources, please, refer to section *Modifying the available resources*.

Currently, COMPSs agents allow interaction through two interfaces: the Comm interface and the REST interface. The Comm interface leverages on a propietary protocol to submit operations and request updates on the current resource configuration of the agent. Although users and applications can use this interface, its design purpose is to enable high-performance interactions among agents rather than supporting user interaction. The REST interface takes the completely opposed approach; Users should interact with COMPSs agents through it rather than submitting tasks with the Comm interface. The COMPSs agent allows to enact both interfaces at a time; thus, users can manually submit operations using the REST interface, while other agents can use the Comm interface. However, the device owner can decide at deploy time which of the interfaces will be available on the agent and through which port the API will be exposed using the `rest_port` and `comm_port` options of the `compss_-agent_start` command. Other agents can be configured to interact with the agent through any of the interfaces. For further details on how to configure the interaction with another agent, please, refer to section *Modifying the available resources*.

```
compss@bsc:~$ compss_agent_start -h

Usage: /opt/COMPSs/Runtime/scripts/user/compss_agent_start [OPTION]...

COMPSs options:

    --appdir=<path>                         Path for the application class folder.
                                            Default: /home/flordan/git/compss/framework/
↪builders

    --classpath=<path>                      Path for the application classes / modules
                                            Default: Working Directory

    --comm=<className>                      Class that implements the adaptor for␣
↪communications with other nodes

                                            Supported adaptors:
                                                  ─── es.bsc.compss.nio.master.NIOAdaptor
                                                  ─── es.bsc.compss.gat.master.GATAdaptor
                                                  ─── es.bsc.compss.agent.rest.Adaptor
                                                  ─── es.bsc.compss.agent.comm.CommAgentAdaptor
                                            Default: es.bsc.compss.agent.comm.CommAgentAdaptor

    --comm_port=<int>                       Port on which the agent sets up a Comm interface.␣
↪(<=0: Disabled)

    -d, --debug                             Enable debug. (Default: disabled)

    --hostname                              Name with which itself and other agents will␣
↪identify the agent.

    --jvm_opts="string"                     Extra options for the COMPSs Runtime JVM. Each␣
↪option separed by "," and without blank spaces (Notice the quotes)

    --library_path=<path>                   Non-standard directories to search for libraries␣
↪(e.g. Java JVM library, Python library, C binding library)
                                            Default: Working Directory

    --log_dir=<path>                        Log directory. (Default: /tmp/)

    --log_level=<level>                     Set the debug level: off | info | api | debug |␣
↪trace

                                            Default: off
```

(continues on next page)

```
    --master_port=<int>                         Port to run the COMPSs master communications.
                                                (Only when es.bsc.compss.nio.master.NIOAdaptor is␣
→used. The value is overriden by the comm_port value.)
                                                Default: [43000,44000]

    --pythonpath=<path>                         Additional folders or paths to add to the␣
→PYTHONPATH
                                                Default: /home/flordan/git/compss/framework/
→builders

    --python_interpreter=<string>              Python interpreter to use (python/python2/
→python3).
                                                Default: python Version:

    --python_propagate_virtual_environment=<true>    Propagate the master virtual environment␣
→to the workers (true/false).
                                                    Default: true

    --python_mpi_worker=<false>                 Use MPI to run the python worker instead of␣
→multiprocessing. (true/false).
                                                Default: false

    --python_memory_profile                     Generate a memory profile of the master.
                                                Default: false
    --python_worker_cache=<string>              Python worker cache (true/size/false).
                                                Only for NIO without mpi worker and python >= 3.8.
                                                Default: false

    --project=<path>                            Path of the project file
                                                (Default: /opt/COMPSs/Runtime/configuration/xml/
→projects/examples/local/project.xml)

    --resources=<path>                          Path of the resources file
                                                (Default: /opt/COMPSs/Runtime/configuration/xml/
→resources/examples/local/resources.xml)

    --rest_port=<int>                           Port on which the agent sets up a REST interface.␣
→(<=0: Disabled)

    --reuse_resources_on_block=<boolean>        Enables/Disables reusing the resources assigned␣
→to a task when its execution stalls.

                                                (Default:true)

    --scheduler=<className>                     Class that implements the Scheduler for COMPSs
                                                Supported schedulers:
                                                        ├── es.bsc.compss.scheduler.fifodatalocation.
→FIFODataLocationScheduler

                                                        ├── es.bsc.compss.scheduler.fifonew.
→FIFOScheduler

                                                        ├── es.bsc.compss.scheduler.fifodatanew.
→FIFODataScheduler

                                                        ├── es.bsc.compss.scheduler.lifonew.
→LIFOScheduler

                                                        ├── es.bsc.compss.components.impl.
→TaskScheduler
```

```
                                          └── es.bsc.compss.scheduler.loadbalancing.
↪LoadBalancingScheduler
                                          Default: es.bsc.compss.scheduler.loadbalancing.
↪LoadBalancingScheduler

    --scheduler_config_file=<path>        Path to the file which contains the scheduler␣
↪configuration.
                                          Default: Empty

    --input_profile=<path>                Path to the file which stores the input␣
↪application profile
                                          Default: Empty

    --output_profile=<path>               Path to the file to store the application profile␣
↪at the end of the execution
                                          Default: Empty

    --summary                             Displays a task execution summary at the end of␣
↪the application execution
                                          Default: false

    --tracing=<level>, --tracing, -t      Set generation of traces and/or tracing level ( [␣
↪true | basic ] | advanced | scorep | arm-map | arm-ddt | false)
                                          True and basic levels will produce the same␣
↪traces.
                                          When no value is provided it is set to 1
                                          Default: 0

    --trace_label=<string>                Add a label in the generated trace file. Only␣
↪used in the case of tracing is activated.
                                          Default: None

    Other options:
    --help                                prints this message
```

Executing an operation

The **compss_agent_call_operation** commands interacts with the REST interface of the COMPSs agent to submit an operation.

```
compss@bsc:~$ compss_agent_call_operation [options] application_name application_arguments
```

The command has two mandatory flags `--master_node` and `--master_port` to indicate the endpoint of the COMPSs Agent. By default, the command submits an execution of the `main` method of the Java class with the name passed in as the `application_name` and gathering all the application arguments in a single String[] instance. To execute Python methods, the user can use the `--lang=PYTHON` option and the Agent will execute the python script with the name passed in as `application_name`. Operation invocations can be customized by using other options of the command. The `--method_name` option allow to execute a specific method; in the case of specifying a method, each of the parameters will be passed in as a different parameter to the function and it is necessary to indicate the `--array` flag to encapsulate all the parameters as an array.

Additionally, the command offers two options to shutdown a whole agents deployment upon the operation completion. The flag `--stop` indicates that, at the end of the operation, the agent receiving the operation request will stop. For shutting down the rest of the deployment, the command offers the option `--forward_to` to indicate a list of IP:port pairs. Upon the completion of the operation, the agent receiving the request will forward the stop command to all the nodes specified in such option.

```
compss@bsc.es:~$ compss_agent_call_operation -h

Usage: compss_agent_call_operation [options] application_name application_arguments

* Options:
General:
    --help, -h                              Print this help message

    --opts                                  Show available options

    --version, -v                           Print COMPSs version

    --master_node=<string>                  Node where to run the COMPSs Master
                                            Mandatory

    --master_port=<string>                  Node where to run the COMPSs Master
                                            Mandatory

    --stop                                  Stops the agent after the execution
                                            of the task.

    --forward_to=<list>                     Forwards the stop action to other
                                            agents, the list shoud follow the
                                            format:
                                            <ip1>:<port1>;<ip2>:<port2>...
Launch configuration:
    --cei=<string>                          Canonical name of the interface declaring the␣
↪methods
                                            Default: No interface declared

    --lang=<string>                         Language implementing the operation
                                            Default: JAVA

    --method_name=<string>                  Name of the method to invoke
                                            Default: main and enables array parameter

    --parameters_array, --array            Parameters are encapsulated as an array
                                            Default: disabled
```

For example, to submit the execution of the `demoFunction` method from the `es.bsc.compss.tests.DemoClass` class passing in a single parameter with value 1 on the agent 127.0.0.1 with a REST interface listening on port 46101, the user should execute the following example command:

```
compss@bsc.es:~$ compss_agent_call_operation --master_node="127.0.0.1" --master_port="46101" -
↪-method_name="demoFunction" es.bsc.compss.test.DemoClass 1
```

For the agent to detect inner tasks within the operation execution, the COMPSs Programming model requires an interface selecting the methods to be replaced by asynchronous task creations. An invoker should use the `--cei` option to specify the name of the interface selecting the tasks.

Modifying the available resources

Finally, the COMPSs framework offers tree commands to control dynamically the pool of resources available for the runtime un one agent. These commands are `compss_agent_add_resources`, `compss_agent_reduce_resources` and `compss_agent_lost_resources`.

The **compss_agent_add_resources** commands interacts with the REST interface of the COMPSs agent to attach new resources to the Agent.

```
compss@bsc.es:~$ compss_agent_add_resources [options] resource_name [<adaptor_property_
→name=adaptor_property_value>]
```

By default, the command modifies the resource pool of the agent deployed on the node running the command listening on port 46101; however, this can be modified by using the options `--agent_node` and `--agent_-port` to indicate the endpoint of the COMPSs Agent. The other options passed in to the command modify the characteristics of the resources to attach; by default, it adds one single CPU core. However, it also allows to modify the amount of GPU cores, FPGAs, memory type and size and OS details.

```
compss@bsc.es:~$ compss_agent_add_resources -h

Usage: compss_agent_add_resources [options] resource_name [<adaptor_property_name=adaptor_
→property_value>]

* Options:
General:
    --help, -h                          Print this help message

    --opts                              Show available options

    --version, -v                       Print COMPSs version

    --agent_node=<string>               Name of the node where to add the resource
                                        Default:

    --agent_port=<string>               Port of the node where to add the resource
                                        Default:
Resource description:
    --comm=<string>                     Canonical class name of the adaptor to interact␣
→with the resource

                                        Default: es.bsc.compss.agent.comm.CommAgentAdaptor

    --cpu=<integer>                     Number of cpu cores available on the resource
                                        Default: 1

    --gpu=<integer>                     Number of gpus devices available on the resource
                                        Default: 0

    --fpga=<integer>                    Number of fpga devices available on the resource
                                        Default: 0

    --mem_type=<string>                 Type of memory used by the resource
                                        Default: [unassigned]

    --mem_size=<string>                 Size of the memory available on the resource
                                        Default: -1

    --os_type=<string>                  Type of operating system managing the resource
                                        Default: [unassigned]
```

(continues on next page)

```
    --os_distr=<string>                          Distribution of the operating system managing the␣
↪resource

                                                 Default: [unassigned]


    --os_version=<string>                        Version of the operating system managing the␣
↪resource

                                                 Default: [unassigned]
```

If `resource_name` matches the name of the Agent, the capabilities of the device are increased according to the description; otherwise, the runtime adds a remote worker to the resource pool with the specified characteristics. Notice that, if there is another resource within the pool with the same name, the agent will increase the resources of such node instead of adding it as a new one. The `--comm` option is used for selecting which adaptor is used for interacting with the remote node; the default adaptor (CommAgent) interacts with the remote node through the Comm interface of the COMPSs agent.

The following command adds a new Agent onto the pool of resources of the Agent deployed at IP 192.168.1.70 with a REST Interface on port 46101. The new agent, which has 4 CPU cores, is deployed on IP 192.168.1.72 and has a Comm interface endpoint on port 46102.

```
compss@bsc.es:~$ compss_agent_add_resources --agent_node=192.168.1.70 --agent_port=46101 --
↪cpu=4 192.168.1.72 Port=46102
```

Conversely, the `compss_agent_reduce_resources` command allows to reduce the number of resources configured in an agent. Executing the command causes the target agent to reduce the specified amount of resources from one of its configured neighbors. At the moment of the reception of the resource removal request, the agent might be actively using those remote resources by executing some tasks. If that is the case, the agent will register the resource reduction request, stop submitting more workload to the corresponding node, and, when the idle resources of the node match the request, the agent removes them from the pool. If upon the completion of the `compss_agent_-reduce_resources` command no resources are associated to the reduced node, the node is completely removed from the resource pool of the agent. The options and default values are the same than for the `compss_agent_-add_resources` command. Notice that `--comm` option is not available because only one resource can be associated to that name regardless the selected adaptor.

```
compss@bsc.es:~$ compss_agent_reduce_resources -h

Usage: compss_agent_reduce_resources [options] resource_name

* Options:
General:
    --help, -h                           Print this help message

    --opts                               Show available options

    --version, -v                        Print COMPSs version

    --agent_node=<string>                Name of the node where to add the resource
                                         Default:

    --agent_port=<string>                Port of the node where to add the resource
                                         Default:
Resource description:
    --cpu=<integer>                      Number of cpu cores available on the resource
                                         Default: 1

    --gpu=<integer>                      Number of gpus devices available on the resource
                                         Default: 0
```

```
   --fpga=<integer>                         Number of fpga devices available on the resource
                                            Default: 0

   --mem_type=<string>                      Type of memory used by the resource
                                            Default: [unassigned]

   --mem_size=<string>                      Size of the memory available on the resource
                                            Default: -1

   --os_type=<string>                       Type of operating system managing the resource
                                            Default: [unassigned]

   --os_distr=<string>                      Distribution of the operating system managing the␣
↪resource

                                            Default: [unassigned]

   --os_version=<string>                    Version of the operating system managing the␣
↪resource

                                            Default: [unassigned]
```

Finally, the last command to control the pool of resources configured, `compss_agent_lost_resources`, immediately removes from an agent's pool all the resources corresponding to the remote node associated to that name.

```
compss@bsc.es:~$ compss_agent_lost_resources [options] resource_name
```

In this case, the only available options are those used for identifying the endpoint of the agent:`--agent_node` and `--agent_port`. As with the previous commands, by default, the request is submitted to the agent deployed on the IP address 127.0.0.1 and listening on port 46101.

5.3.2.2 Supercomputers

Similar to Section *Supercomputers* for Master-Worker deployments, this section is intended to walk you through the COMPSs usage with agents in Supercomputers. All the configuration and commands to install COMPSs on the Supercomputer, load the environment and submitting a job remain exactly the same as described in Sections *Supercomputers*.

The only difference to submit jobs with regards the COMPSs Master-Worker approach is to enact the `agents` option of the **enqueue_compss** command. When this option is enabled, the whole COMPSs deployment changes and, instead of deploying the COMPSs master in one node and workers in the remaining ones, it deploys an agent in each node provided by the queue system. When all the agents have been deployed, COMPSs' internal scripts handling the job execution will submit the operation using the REST API of the one of the agent. Although COMPSs agents allow any method of the application to be the starting point of the execution, to mantain the similarities between the scripts when deploying COMPSs following the Master-Worker or the Agents approaches, the execution will start with the main method of the class/module passed in as a parameter to the script.

The main advantage of using the Agents approach in Supercomputers is the ability to define different topologies. For that purpose, the `--agents` option of the **enqueue_compss** script allows to choose two different options `--agents=plain` and `--agents=tree`.

The `Plain` topology configures the deployment resembling the Master-worker approach. One of the agents is selected as the master an has all the other agents as workers where to offload tasks; the agents acting as workers also host a COMPSs runtime and, therefore, they can detect nested tasks on the tasks offloaded onto them. However, nested tasks will always be executed on the worker agent detecting them.

The `Tree` topology is the default topology when using agent deployments on Supercomputers. These option tries to create a three-layer topology that aims to exploit data locality and reduce the workload of the scheduling problem. Such topology consists in deploying an agent on each node managing only the resources available within

---

the node. Then, the script groups all the nodes by rack and selects a representative node for each group that will orchestrate all the resources within it and offload tasks onto the other agents. Finally, the script picks one of these representative agents as the main agent of the hierarchy; this main agent is configured to be able to offload tasks onto the representative agents for all other racks; it will be onto this node that the script will call the main method of the execution. The following image depicts an example of such topology on Marenostrum.
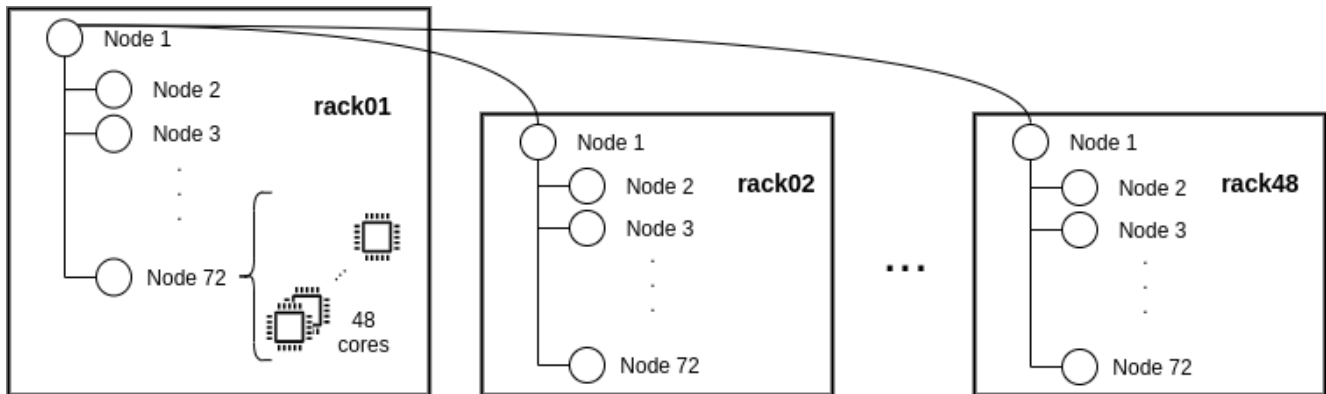
To ensure that no resources are wasted waiting from the execution end until the wall clock limit, the **enqueue_-compss** script submits the invocation enabling the `--stop` and `--forward` options to stop all the deployed agents for the execution.

Chapter 6

Tools

COMPSs has a rich echosystem of tools that help on monitoring and measuring the perfornamce of COMPSs applications.

This section is intended to walk you through the COMPSs' tools.

6.1 Application graph

At the end of the application execution a dependency graph can be generated representing the order of execution of each type of task and their dependencies. To allow the final graph generation the `-g` flag has to be passed to the `runcompss` command (alternative flags to the `-g` are the `--graph` or the `--graph=true`); the graph file is written in the `<BASE_LOG_DIR>/monitor/complete_graph.dot` at the end of the execution (`<BASE_LOG_DIR>` is usually `$HOME/.COMPSs` unless the `--base_log_dir=<BASE_LOG_DIR>` flag is specified).

Warning: Application graph generation is **not supported using agents**.

Figure 19 shows a dependency graph example of a *SparseLU* Java application. The graph can be converted from *dot* format to *pdf* format by running the following command:

```
compss@bsc:~$ compss_gengraph ~/.COMPSs/sparseLU.arrays.SparseLU_01/monitor/complete_graph.dot
```

When only the `dot` file is passed as a parameter, `compss_gengraph` creates a PDF image file by default, named `complete_graph.pdf`. However, a different image format can be selected by specifying any other format supported by GraphViz. Check the list of supported formats here, and verify also that they are supported in your system, since not all formats are available for all systems. An example on how to specify the format would be:

```
compss@bsc:~$ compss_gengraph svg ~/.COMPSs/sparseLU.arrays.SparseLU_01/monitor/complete_
↪graph.dot
```

This would generate a `complete_graph.svg` output file that would contain the application's workflow image in Scalable Vector Graphics (SVG) format.
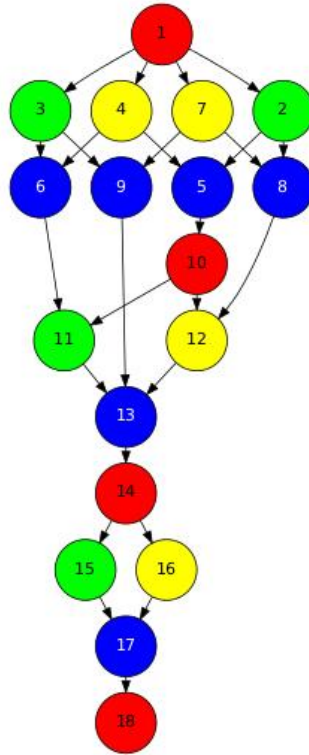
Figure 19: The dependency graph of the SparseLU application

6.2 Monitor

The COMPSs Framework includes a Web graphical interface that can be used to monitor the execution of COMPSs applications. COMPSs Monitor is installed as a service and can be easily managed by running any of the following commands:

```
compss@bsc:~$ /etc/init.d/compss-monitor usage
Usage: compss-monitor {start | stop | reload | restart | try-restart | force-reload | status}
```

> **Warning:**   The monitor is **not supported using agents**.

6.2.1 Service configuration

The COMPSs Monitor service can be configured by editing the **/opt/COMPSs/Tools/monitor/apache-tomcat/ conf/compss-monitor.conf** file which contains one line per property:

COMPSS_MONITOR Default directory to retrieve monitored applications (defaults to the `.COMPSs` folder inside the `root` user).
COMPSs_MONITOR_PORT Port where to run the compss-monitor web service (defaults to 8080).
COMPSs_MONITOR_TIMEOUT Web page timeout between browser and server (defaults to 20s).

6.2.2 Usage

In order to use the COMPSs Monitor users need to start the service as shown in Figure 20.

Figure 20: COMPSs Monitor start command

Tip: The monitor can be started and stopped in multiple environments (local, docker and supercomputer) automatically using the CLI. Please check: *Running the COMPSs monitor*

And use a web browser to open the specific URL:

```
compss@bsc:~$ firefox http://localhost:8080/compss-monitor &
```

The COMPSs Monitor allows to monitor applications from different users and thus, users need to first login to access their applications. As shown in Figure 21, the users can select any of their executed or running COMPSs applications and display it.

To enable **all** the COMPSs Monitor features, applications must run the `runcompss` command with the `-m` flag. This flag allows the COMPSs Runtime to store special information inside inside the `log_base_folder` under the `monitor` folder (see Figure 21 and Figure 22). Only advanced users should modify or delete any of these files. If the application that a user is trying to monitor has not been executed with this flag, some of the COMPSs Monitor features will be disabled.

```
compss@bsc:~/tutorial_apps/java/simple/jar$ runcompss -dm simple.Simple 1
[  INFO] Using default execution type: compss
[  INFO] Using default location for project file: /opt/COMPSs/Runtime/configuration/xml/
↪projects/default_project.xml
[  INFO] Using default location for resources file: /opt/COMPSs/Runtime/configuration/xml/
↪resources/default_resources.xml
[  INFO] Using default language: java


----------------- Executing simple.Simple --------------------------

WARNING: COMPSs Properties file is null. Setting default values
[(799)    API]  -  Deploying COMPSs Runtime v<version>
[(801)    API]  -  Starting COMPSs Runtime v<version>
[(801)    API]  -  Initializing components
[(1290)    API]  -  Ready to process tasks
```

(continues on next page)
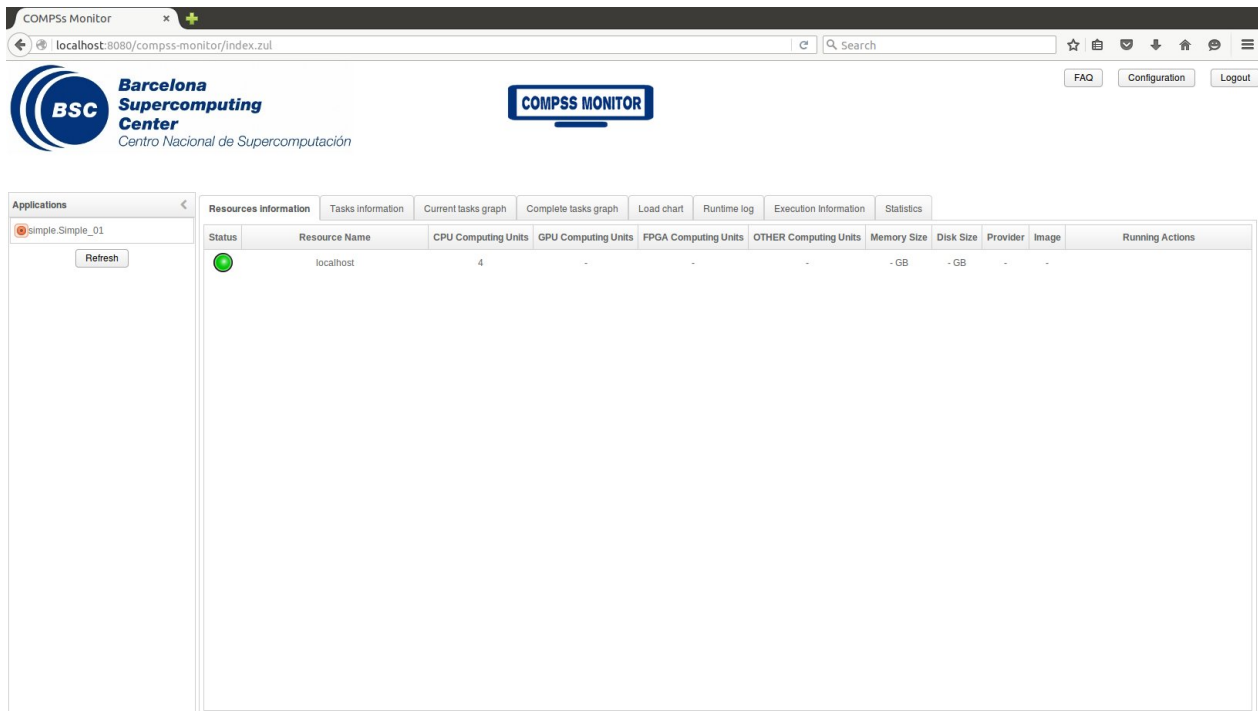
Figure 21: COMPSs monitoring interface

(continued from previous page)

```
[(1293)    API]  -  Opening /home/compss/tutorial_apps/java/simple/jar/counter in mode OUT
[(1338)    API]  -  File target Location: /home/compss/tutorial_apps/java/simple/jar/counter
Initial counter value is 1
[(1340)    API]  -  Creating task from method increment in simple.SimpleImpl
[(1340)    API]  -  There is 1 parameter
[(1341)    API]  -   Parameter 1 has type FILE_T
Final counter value is 2
[(4307)    API]  -  No more tasks for app 1
[(4311)    API]  -  Getting Result Files 1
[(4340)    API]  -  Stop IT reached
[(4344)    API]  -  Stopping Graph generation...
[(4344)    API]  -  Stopping Monitor...
[(6347)    API]  -  Stopping AP...
[(6348)    API]  -  Stopping TD...
[(6509)    API]  -  Stopping Comm...
[(6510)    API]  -  Runtime stopped
[(6510)    API]  -  Execution Finished


------------------------------------------------------------
```


Figure 22: Logs generated by the Simple java application with the monitoring flag enabled

6.2.3 Graphical Interface features

In this section we provide a summary of the COMPSs Monitor supported features available through the graphical interface:

- **Resources information** Provides information about the resources used by the application
- **Tasks information** Provides information about the tasks definition used by the application
- **Current tasks graph** Shows the tasks dependency graph currently stored into the COMPSs Runtime
- **Complete tasks graph** Shows the complete tasks dependecy graph of the application
- **Load chart** Shows different dynamic charts representing the evolution over time of the resources load and the tasks load
- **Runtime log** Shows the runtime log
- **Execution Information** Shows specific job information allowing users to easily select failed or uncompleted jobs
- **Statistics** Shows application statistics such as the accumulated cloud cost.

Important: To enable all the COMPSs Monitor features applications must run with the `-m` flag.

The webpage also allows users to configure some performance parameters of the monitoring service by accessing the *Configuration* button at the top-right corner of the web page.

For specific COMPSs Monitor feature configuration please check our *FAQ* section at the top-right corner of the web page.

6.3 Tracing

COMPSs is instrumented with EXTRAE, which enables to produce PARAVER traces for performance profiling.

This section is intended to walk you through the tracing of your COMPSs applications in order to analyse the performance with great detail.

6.3.1 COMPSs applications tracing

COMPSs Runtime has a built-in instrumentation system to generate post-execution tracefiles of the applications' execution. The tracefiles contain different events representing the COMPSs master state, the tasks' execution state, and the data transfers (transfers' information is only available when using NIO adaptor), and are useful for both visual and numerical performance analysis and diagnosis. The instrumentation process essentially intercepts and logs different events, so it adds overhead to the execution time of the application.

The tracing system uses Extrae[1] to generate tracefiles of the execution that, in turn, can be visualized with Paraver[2]. Both tools are developed and maintained by the Performance Tools team of the BSC and are available on its web page http://www.bsc.es/computer-sciences/performance-tools.

Extrae keeps track of the events in an intermediate format file (with *.mpit* extension). At the end of the execution, all these files can bee gathered and merged with Extrae's `mpi2prv` command to create the final tracefile, a Paraver format file (.prv). See the *Visualization* Section for further information about the Paraver tool.

For further information about Extrae, please visit the following site: http://www.bsc.es/computer-science/extrae

When tracing is enabled, Extrae instruments computing threads and some resources management operations to provide information about tasks' executions, data transfers, and, if PAPI is available (see *PAPI: Hardware Counters* for more info), hardware counters.

---

[1] For more information: https://www.bsc.es/computer-sciences/extrae
[2] For more information: https://www.bsc.es/computer-sciences/performance-tools/paraver

6.3.1.1 Activate Tracing

By default, the tracing is disabled for any COMPSs execution. However, all the scripts that start a COMPSs excution (`runcompss`, `enqueue_compss` and `compss_agent_start`) have an option to activate the tracing for that the execution. The user activates it by providing one of the following arguments to the corresponding script.

- `-t`
- `--tracing`
- `--tracing=true`

Example:

```
$ runcompss --tracing application_name application_args
```

When tracing is activated, Extrae generates additional output to help the user ensure that instrumentation is turned on and working without issues. This output contains diverse information about the tracing system as shown in the following example and its running. Extrae version used (`VERSION` will be replaced by the actual number during executions), the XML configuration file used (`/opt/COMPSs/Runtime/configuration/xml/tracing/extrae_basic.xml` – if using python, the `extrae_python_worker.xml` located in the same folder will be used in the workers), the amount of threads instrumented (objects through 1.1.1 to 1.2.7), available hardware counters (`PAPI_TOT_INS (0x80000032) ... PAPI_L3_TCM (0x80000008)`) or the name of the generated tracefile (`./trace/kmeans.py_compss.prv`). When debug is activated, the log of each worker also contains the Extrae initialization information.

Tip: The application used for this example is *Kmeans*. The trace generated by this execution is depicted in Figure 23.

```
$ runcompss --tracing --generate_trace=false kmeans.py -n 102400000 -f 8 -d 3 -c 8 -i 10
[ INFO ] Inferred PYTHON language
[ INFO ] Using default location for project file: /opt/COMPSs//Runtime/configuration/xml/
↪projects/default_project.xml
[ INFO ] Using default location for resources file: /opt/COMPSs//Runtime/configuration/xml/
↪resources/default_resources.xml
[ INFO ] Using default execution type: compss

---------------- Executing kmeans.py -------------------------

Welcome to Extrae 3.8.3
Extrae: Parsing the configuration file (/home/user/.COMPSs/kmeans.py_01/cfgfiles/extrae.xml)␣
↪begins
Extrae: Warning! <trace> tag has no <home> property defined.
Extrae: Generating intermediate files for Paraver traces.
Extrae: PAPI domain set to ALL for HWC set 1
Extrae: HWC set 1 contains following counters < PAPI_TOT_INS (0x80000032) PAPI_TOT_CYC␣
↪(0x8000003b) PAPI_L1_DCM (0x80000000) PAPI_L2_DCM (0x80000002) PAPI_L3_TCM (0x80000008)␣
↪PAPI_BR_INS (0x80000037) PAPI_BR_MSP (0x8000002e) RESOURCE_STALLS (0x4000002e) > - never␣
↪changes
Extrae: Tracing buffer can hold 100000 events
Extrae: Circular buffer disabled.
Extrae: Warning! <input-output> tag will be ignored. This library does not support␣
↪instrumenting I/O calls.
Extrae: Dynamic memory instrumentation is disabled.
Extrae: Basic I/O memory instrumentation is disabled.
Extrae: System calls instrumentation is disabled.
Extrae: Parsing the configuration file (/home/user/.COMPSs/kmeans.py_01/cfgfiles/extrae.xml)␣
↪has ended
Extrae: Intermediate traces will be stored in /home/user/.COMPSs/kmeans.py_01/trace
```

<div align="right">(continues on next page)</div>

```
Extrae: Tracing mode is set to: Detail.
Extrae: Error! Hardware counter PAPI_TOT_INS (0x80000032) cannot be added in set 1 (task 0,␣
↪thread 0)
Extrae: Error! Hardware counter PAPI_TOT_CYC (0x8000003b) cannot be added in set 1 (task 0,␣
↪thread 0)
Extrae: Error! Hardware counter PAPI_L1_DCM (0x80000000) cannot be added in set 1 (task 0,␣
↪thread 0)
Extrae: Error! Hardware counter PAPI_L2_DCM (0x80000002) cannot be added in set 1 (task 0,␣
↪thread 0)
Extrae: Error! Hardware counter PAPI_L3_TCM (0x80000008) cannot be added in set 1 (task 0,␣
↪thread 0)
Extrae: Error! Hardware counter PAPI_BR_INS (0x80000037) cannot be added in set 1 (task 0,␣
↪thread 0)
Extrae: Error! Hardware counter PAPI_BR_MSP (0x8000002e) cannot be added in set 1 (task 0,␣
↪thread 0)
Extrae: Error! Hardware counter RESOURCE_STALLS (0x4000002e) cannot be added in set 1 (task 0,
↪ thread 0)
Extrae: Error when setting domain for eventset 1
Extrae: PAPI_start failed to start eventset 1 on thread 0! (error = -1)
Extrae: Successfully initiated with 1 tasks and 1 threads

WARNING: COMPSs Properties file is null. Setting default values
[(732)    API] - Starting COMPSs Runtime v2.10.rc2205 (build 20220525-1503.
↪re74c11cbc6c248a6c5745edaf3a4a47c2c9d0c7e)
Generation/Load done
Starting kmeans
Doing iteration #1/10
Doing iteration #2/10
Doing iteration #3/10
Doing iteration #4/10
Doing iteration #5/10
Doing iteration #6/10
Doing iteration #7/10
Doing iteration #8/10
Doing iteration #9/10
Doing iteration #10/10
Ending kmeans
-----------------------------------------
-------------- RESULTS ------------------
-----------------------------------------
Initialization time: 114.582741
Kmeans time: 140.148499
Total time: 254.731240
-----------------------------------------
CENTRES:
[[0.69757475 0.74511351 0.48157611]
 [0.54683653 0.20274669 0.2117475 ]
 [0.24194863 0.74448094 0.75633981]
 [0.21854362 0.67072938 0.23273541]
 [0.77272546 0.68522249 0.16245965]
 [0.22683962 0.23359743 0.67203863]
 [0.75351606 0.73746265 0.83339847]
 [0.75838884 0.23805883 0.71538748]]
-----------------------------------------
Extrae: Intermediate raw trace file created : /home/user/.COMPSs/kmeans.py_01/trace/set-0/
↪TRACE@bsccs189.0000082523000000000002.mpit
```

```
Extrae: Intermediate raw trace file created : /home/user/.COMPSs/kmeans.py_01/trace/set-0/
↪TRACE@bsccs189.0000082523000000000001.mpit
Extrae: Intermediate raw trace file created : /home/user/.COMPSs/kmeans.py_01/trace/set-0/
↪TRACE@bsccs189.0000082523000000000003.mpit
Extrae: Intermediate raw trace file created : /home/user/.COMPSs/kmeans.py_01/trace/set-0/
↪TRACE@bsccs189.0000082523000000000004.mpit
Extrae: Intermediate raw trace file created : /home/user/.COMPSs/kmeans.py_01/trace/set-0/
↪TRACE@bsccs189.0000082523000000000000.mpit
Extrae: Intermediate raw sym file created : /home/user/.COMPSs/kmeans.py_01/trace/set-0/
↪TRACE@bsccs189.0000082523000000000000.sym
Extrae: Deallocating memory.
Extrae: Application has ended. Tracing has been terminated.
[(259804)    API]  -  Execution Finished
Dismissing tracing package removal. Traces were requested but not generated.


------------------------------------------------------------
```

6.3.1.2 Trace Generation

At the end of a COMPSs application execution, each node involved in the execution generates a package file containing all the output generated by Extrae; the master node gathers all these files in the trace subfolder of the log directory of the execution.

After that, an additional step to merge the Extrae output in these packages and merge them into as single trace that can be opened with Paraver. This step can be done by the scripts launching COMPSs right after the execution of the application. To enable/disable this procedure, COMPSs scripts have three additional options to control the trace generation. The `generate_trace` option allows to enable disable this process. By default, it is enabled in `runcompss` and `compss_agent_start` and disabled in `enqueue_compss` executions. Through the `--trace_label` option, the user set the name of the resulting trace; and , with the `--delete_trace_packages`, the user specifies to the script whether the packages generated by the runtime should be kept after completing the trace generation or if they must be deleted.

For those executions where the trace was not generated by the execution script, COMPSs provides the `compss_-gentrace` and `enqueue_compss_gentrace` scripts. As with `runcompss` and `enqueue_compss`, the `compss_-gentrace` script merges the trace while `enqueue_compss_gentrace` enqueues a job on a queue system that will do the same.

```
~/.COMPSs/kmeans.py_01/trace$ compss_gentrace --trace_name=trace
[ INFO ] COMPSs Paraver trace generation.
Traces:
  Input folder: /home/user/.COMPSs/kmeans.py_01/trace
  Output folder: /home/user/.COMPSs/kmeans.py_01/trace
  Trace name: trace

Options:
  Custom threads: true
  Keep packages: false

Logging:
  Level: off
  Folder: /home/user/.COMPSs/kmeans.py_01/trace
merger: Output trace format is: Paraver
merger: Extrae 3.8.3
mpi2prv: Assigned nodes < bsccs189 >
mpi2prv: Assigned size per processor < <1 Mbyte >
mpi2prv: File /tmp/tmp.b9P6UYmIJ5/python/set-0/TRACE@bsccs189.0000082745000000000000.mpit is␣
↪object 1.1.1 on node bsccs189 assigned to processor 0
```

```
mpi2prv: File /tmp/tmp.b9P6UYmIJ5/python/set-0/TRACE@bsccs189.0000082747000000000000.mpit is␣
↪object 2.1.1 on node bsccs189 assigned to processor 0
mpi2prv: File /tmp/tmp.b9P6UYmIJ5/python/set-0/TRACE@bsccs189.0000082748000000000000.mpit is␣
↪object 3.1.1 on node bsccs189 assigned to processor 0
mpi2prv: File /tmp/tmp.b9P6UYmIJ5/python/set-0/TRACE@bsccs189.0000082749000000000000.mpit is␣
↪object 4.1.1 on node bsccs189 assigned to processor 0
mpi2prv: File /tmp/tmp.b9P6UYmIJ5/python/set-0/TRACE@bsccs189.0000082750000000000000.mpit is␣
↪object 5.1.1 on node bsccs189 assigned to processor 0
mpi2prv: A total of 8 symbols were imported from /tmp/tmp.b9P6UYmIJ5/python/TRACE.sym file
mpi2prv: 0 function symbols imported
mpi2prv: 8 HWC counter descriptions imported
mpi2prv: Checking for target directory existence... exists, ok!
mpi2prv: Warning: Couldn't open /tmp/COMPSsWorker/f83c9da7-74c1-4703-b0d5-c980823b6422/
↪localhost/python/.libseqtrace-subprocess.so for reading, addresses may not be translated.
mpi2prv: Warning: Couldn't open /tmp/COMPSsWorker/f83c9da7-74c1-4703-b0d5-c980823b6422/
↪localhost/python/.libseqtrace-subprocess.so for reading, addresses may not be translated.
mpi2prv: Warning: Couldn't open /tmp/COMPSsWorker/f83c9da7-74c1-4703-b0d5-c980823b6422/
↪localhost/python/.libseqtrace-subprocess.so for reading, addresses may not be translated.
mpi2prv: Warning: Couldn't open /tmp/COMPSsWorker/f83c9da7-74c1-4703-b0d5-c980823b6422/
↪localhost/python/.libseqtrace-subprocess.so for reading, addresses may not be translated.
mpi2prv: Selected output trace format is Paraver
mpi2prv: Stored trace format is Paraver
mpi2prv: Searching synchronization points... done
mpi2prv: Time Synchronization disabled.
mpi2prv: Circular buffer enabled at tracing time? NO
mpi2prv: Parsing intermediate files
mpi2prv: Progress 1 of 2 ... 5% 10% 15% 20% 25% 30% 35% 40% 45% 50% 55% 60% 65% 70% 75% 80% 85
↪% 90% 95% done
mpi2prv: Processor 0 succeeded to translate its assigned files
mpi2prv: Elapsed time translating files: 0 hours 0 minutes 0 seconds
mpi2prv: Elapsed time sorting addresses: 0 hours 0 minutes 0 seconds
mpi2prv: Generating tracefile (intermediate buffers of 1342156 events)
        This process can take a while. Please, be patient.
mpi2prv: Progress 2 of 2 ... 5% 10% 15% 20% 25% 30% 35% 40% 45% 50% 55% 60% 65% 70% 75% 80% 85
↪% 90% 95% done
mpi2prv: Warning! Clock accuracy seems to be in microseconds instead of nanoseconds.
mpi2prv: Elapsed time merge step: 0 hours 0 minutes 0 seconds
mpi2prv: Resulting tracefile occupies 144040 bytes
mpi2prv: Removing temporal files... done
mpi2prv: Elapsed time removing temporal files: 0 hours 0 minutes 0 seconds
mpi2prv: Congratulations! /home/user/.COMPSs/kmeans.py_01/trace/python//1_python_trace.prv␣
↪has been generated.
merger: Output trace format is: Paraver
merger: Extrae 3.8.3
mpi2prv: Assigned nodes < bsccs189 >
mpi2prv: Assigned size per processor < 1 Mbytes >
mpi2prv: File /home/user/.COMPSs/kmeans.py_01/trace/set-0/TRACE@bsccs189.
↪0000082523000000000000.mpit is object 1.1.1 on node bsccs189 assigned to processor 0
mpi2prv: File /home/user/.COMPSs/kmeans.py_01/trace/set-0/TRACE@bsccs189.
↪0000082523000000000001.mpit is object 1.1.2 on node bsccs189 assigned to processor 0
mpi2prv: File /home/user/.COMPSs/kmeans.py_01/trace/set-0/TRACE@bsccs189.
↪0000082523000000000002.mpit is object 1.1.3 on node bsccs189 assigned to processor 0
mpi2prv: File /home/user/.COMPSs/kmeans.py_01/trace/set-0/TRACE@bsccs189.
↪0000082523000000000003.mpit is object 1.1.4 on node bsccs189 assigned to processor 0
mpi2prv: File /home/user/.COMPSs/kmeans.py_01/trace/set-0/TRACE@bsccs189.
↪0000082523000000000004.mpit is object 1.1.5 on node bsccs189 assigned to processor 0
```

```
mpi2prv: File set-0/TRACE@bsccs189.0000082653000001000000.mpit is object 1.2.1 on node␣
→bsccs189 assigned to processor 0
mpi2prv: File set-0/TRACE@bsccs189.0000082653000001000001.mpit is object 1.2.2 on node␣
→bsccs189 assigned to processor 0
mpi2prv: File set-0/TRACE@bsccs189.0000082653000001000002.mpit is object 1.2.3 on node␣
→bsccs189 assigned to processor 0
mpi2prv: File set-0/TRACE@bsccs189.0000082653000001000003.mpit is object 1.2.4 on node␣
→bsccs189 assigned to processor 0
mpi2prv: File set-0/TRACE@bsccs189.0000082653000001000004.mpit is object 1.2.5 on node␣
→bsccs189 assigned to processor 0
mpi2prv: File set-0/TRACE@bsccs189.0000082653000001000005.mpit is object 1.2.6 on node␣
→bsccs189 assigned to processor 0
mpi2prv: A total of 8 symbols were imported from /home/user/.COMPSs/kmeans.py_01/trace/TRACE.
→sym file
mpi2prv: 0 function symbols imported
mpi2prv: 8 HWC counter descriptions imported
mpi2prv: Checking for target directory existence... exists, ok!
mpi2prv: Selected output trace format is Paraver
mpi2prv: Stored trace format is Paraver
mpi2prv: Searching synchronization points... done
mpi2prv: Time Synchronization disabled.
mpi2prv: Circular buffer enabled at tracing time? NO
mpi2prv: Parsing intermediate files
mpi2prv: Progress 1 of 2 ... 5% 10% 15% 20% 25% 30% 35% 40% 45% 50% 55% 60% 65% 70% 75% 80% 85
→% 90% 95% done
mpi2prv: Processor 0 succeeded to translate its assigned files
mpi2prv: Elapsed time translating files: 0 hours 0 minutes 0 seconds
mpi2prv: Elapsed time sorting addresses: 0 hours 0 minutes 0 seconds
mpi2prv: Generating tracefile (intermediate buffers of 610071 events)
        This process can take a while. Please, be patient.
mpi2prv: Progress 2 of 2 ... 5% 10% 15% 20% 25% 30% 35% 40% 45% 50% 55% 60% 65% 70% 75% 80% 85
→% 90% 95% done
mpi2prv: Warning! Clock accuracy seems to be in microseconds instead of nanoseconds.
mpi2prv: Elapsed time merge step: 0 hours 0 minutes 0 seconds
mpi2prv: Resulting tracefile occupies 327879 bytes
mpi2prv: Removing temporal files... done
mpi2prv: Elapsed time removing temporal files: 0 hours 0 minutes 0 seconds
mpi2prv: Congratulations! /home/user/.COMPSs/kmeans.py_01/trace//trace.prv has been generated.
```

6.3.1.3 Information Available

Tracefiles contain three kinds of information:

Events Marking diverse situations such as the runtime start, tasks' execution or synchronization points.
Communications Showing the transfers and requests of the parameters needed by COMPSs tasks.
Hardware counters Of the execution obtained with Performance API (see *PAPI: Hardware Counters*)

6.3.1.4 Custom Threads

Although Paraver traces illustrate the events, communications and HW counters for each Thread and processor in the system, it is hard to identify what thread is performing each operation.

Currently, traces can show these threads:

- Master node / Agent
  - Application's main thread
  - Access Processor
  - Task Dispatcher
  - File System (High priority)
  - File System (Low priority)
  - Timer
  - Wall_Clock
  - Threads available for computing (executors)
- Worker node
  - Worker main thread
  - Worker File System (High priority)
  - Worker File System (Low priority)
  - Worker timer
  - Threads available for computing (executors)

To ease the identification of each thread, all trace-generating scripts allow an option (`custom_threads`) that triggers a post-processing of the resulting trace to identify which thread corresponds to each runtime component and sort them as runtime thread or threads available to run tasks (executors). By default, this additional step is enabled in all trace-generating scripts.

6.3.1.5 Trace Example

Figure 23 is a tracefile generated by the execution of a k-means clustering algorithm. Each timeline contains information of a different resource, and each event's name is on the legend. Depending on the number of computing threads specified for each worker, the number of timelines varies. However the following threads are always shown:

Master - Thread 1.1.1 This timeline shows the actions performed by the main thread of the COMPSs application

Access Processor - Thread 1.1.2 All the events related to the tasks' parameters management, such as dependencies or transfers are shown in this thread.

Task Dispatcher - Thread 1.1.3 Shows information about the state and scheduling of the tasks to be executed.

Worker X Master - Thread X.1.1 This thread is the master of each worker and handles the computing resources and transfers. It is repeated for each available resource. All data events of the worker, such as requests, transfers and receives are marked on this timeline (when using the appropriate configurations).

Worker X File system - Thread X.1.2 This thread manages the synchronous file system operations (e.g. copy file) performed by the worker.

Worker X Timer - Thread X.1.3 This thread manages the cancellation of the tasks when the wall-clock limit is reached.

Worker X Executor Y - Thread X.2.Y Shows the actual tasks execution information and is repeated as many times as computing threads has the worker X
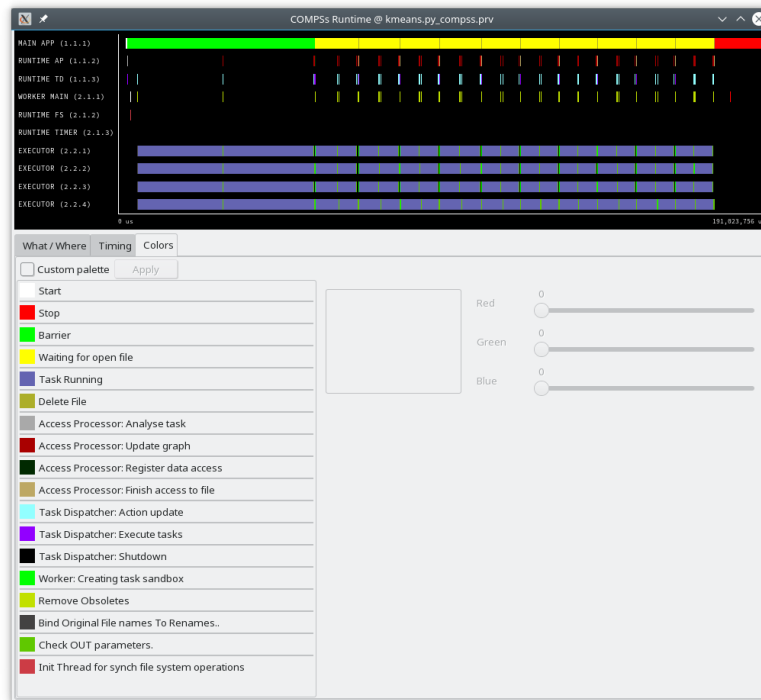
Figure 23: tracefile for a k-means algorithm visualized with compss_runtime.cfg

6.3.1.6 Trace for Agents

Applications deployed as COMPSs Agents can also be traced. Unlike master-worker COMPSs applications, where the trace contains the events for all the nodes within the infrastructure, with the Agents approach, each Agent generates its own trace.

To activate the tracing the `compss_agent_start` command allows the `-t`, `--tracing` and `--tracing=<level>` options with the same meaning as with the master-worker approach. For example:

```
$ compss_agent_start \
    --hostname="COMPSsWorker01" \
    --pythonpath="~/python/path" \
    --log_dir="~/agent1/log" \
    --rest_port="46101" \
    --comm_port="46102" \
    -d -t \
    --project="~/project.xml" \
    --resources="~/resources.xml"&
```

Upon the completion of an operation submitted with the `--stop` flag, the agent stops and generates a trace folder within his log folder, containing the prv, pcf and row files.

```
$ compss_agent_call_operation" \
  --lang="PYTHON" \
  --master_node="127.0.0.1" \
  --master_port="46101" \
  --method_name="kmeans" \
  --stop \
  "kmeans"
```

When multiple agents are involved in an application's execution, the stop command must be forwarded to all the other agents with the `--forward` parameter.
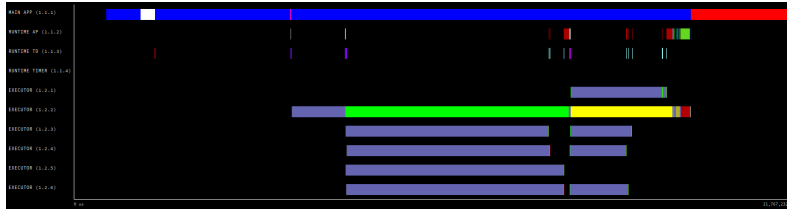
```
$ compss_agent_call_operation" \
    --lang="PYTHON" \
    --master_node="127.0.0.1" \
    --master_port="46101" \
    --method_name="kmeans" \
    --stop \
    --forward_to="COMPSsWorker02:46201;COMPSsWorker03:46301" \
    "kmeans"
```

Upon the completion of the last operation submitted and the shutdown of all involved agents, all agent will have generated their own individual trace.
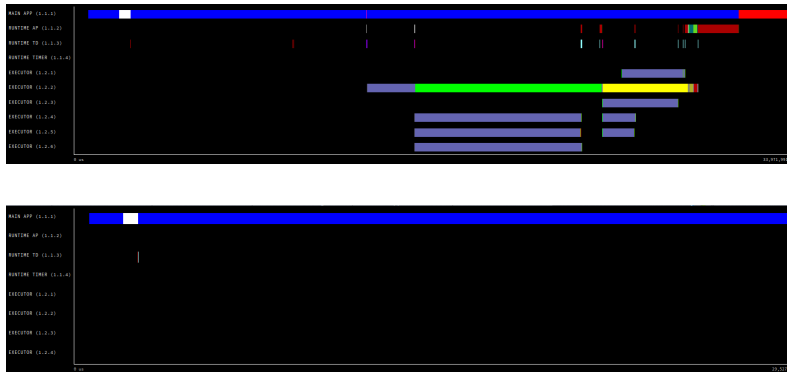
In order to merge this traces the script `compss_agent_merge_traces` can be used. The script takes as parameters the folders of the log dirs of the agents with the traces to merge.

```
$ compss_agent_merge_traces -h
/opt/COMPSs/Runtime/scripts/user/compss_agent_merge_traces <options> <log_dir1> <log_dir2>
↪<log_dir3> ...

Merges the traces of the specified agents into a new trace created at the directory <output_
↪dir>

options:
        -h/--help                                       shows this message

        --output_dir=<output_dir>                       the directory where to store the␣
↪merged traces

        -f/--force_override                             overrides output_dir if it already␣
↪exists without asking

        --result_trace_name=<result_trace_name>         the name of the generated trace
```

Usage example:

```
$ compss_agent_merge_traces \
    --result_trace_name=merged_kmeans \
```
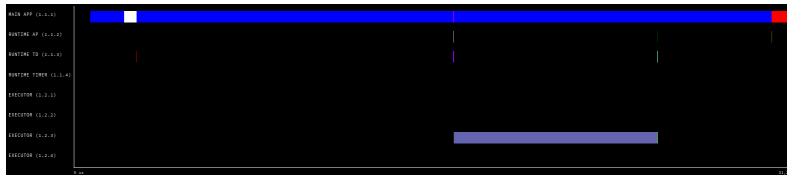
```
~/.COMPSs/1agent_python3_01/agent1 \
~/.COMPSs/1agent_python3_01/agent2 \
~/.COMPSs/1agent_python3_01/agent3
```

The script will put the merged trace in the specified `output_dir` or in the current directory inside a folder named `compss_agent_merge_traces` by default
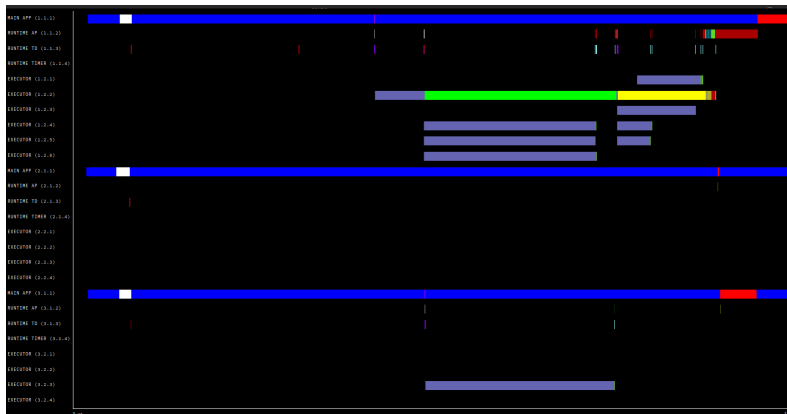
6.3.1.7 Custom Installation and Configuration

Custom Extrae

COMPSs uses the environment variable `EXTRAE_HOME` to get the reference to its installation directory (by default: `/opt/COMPSs/Dependencies/extrae`). However, if the variable is already defined once the runtime is started, COMPSs will not override it. User can take advantage of this fact in order to use custom extrae installations. Just set the `EXTRAE_HOME` environment variable to the directory where your custom package is, and make sure that it is also set for the worker's environment. Be aware that using different Extrae packages can break the runtime and executions so you may change it at your own risk.

Custom Configuration file

COMPSs offers the possibility to specify an extrae custom configuration file in order to harness all the tracing capabilities further tailoring which information about the execution is displayed (except for Python workers). To do so just indicate the file as an execution parameter as follows:

`--extrae_config_file=/path/to/config/file.xml`

In addition, there is also the possibility to specify an extrae custom configuration file for the Python workers as follows:

`--extrae_config_file_python=/path/to/config/file_python.xml`

The configuration files must be in a shared disk between all COMPSs workers because a file's copy is not distributed among them, just the path to that file.

Tip: The default configuration files are in:

- `${COMPSS_HOME}/Runtime/configuration/xml/tracing/extrae_basic.xml`
- `${COMPSS_HOME}/Runtime/configuration/xml/tracing/extrae_python_worker.xml` (when using Python)

The can be taken as base for customization.

Two aspects that configuration files allow to customize are the directories that Extrae will use as working directory and where it leaves the final mpit files. By default, COMPSs configures extrae to leave the traces within the trace sub-directory within the execution log directory. to replicate this behaviour, custom configuration files can use the `{{TRACE_OUTPUT_DIR}}` term on the temporal-directory and final-directory attributes of the configuration. At runtime, this term will be replaced by the actual log dir.

6.3.2 Visualization

Paraver is the BSC tool for trace visualization. Trace events are encoded in Paraver format (`.prv`) by the Extrae tool. Paraver is a powerful tool and allows users to show many views of the trace data using different configuration files. Users can manually load, edit or create configuration files to obtain different tracing views.

The following subsections explain how to load a trace file into Paraver, open the task events view using an already predefined configuration file, and how to adjust the view to display the data properly.

For further information about Paraver, please visit the following site:

http://www.bsc.es/computer-sciences/performance-tools/paraver

6.3.2.1 Trace Loading

The final trace file in Paraver format (.prv) is at the base log folder of the application execution inside the trace folder. The fastest way to open it is calling the Paraver binary directly using the tracefile name as the argument.

```
$ wxparaver /path/to/trace/trace.prv
```

Tip: The path where the traces are usually located is `${HOME}/.COMPSs/<APPLICATION_NAME_INFO>/trace/`.

Where <APPLICATION_NAME_INFO> represents the executed application name and some information, such as the execution number or deployment information (e.g. number of nodes) and the generation time.

6.3.2.2 Configurations

To see the different events, counters and communications that the runtime generates, diverse configurations are available with the COMPSs installation. To open one of them, go to the "Load Configuration" option in the main window and select "File". The configuration files are under the following path for the default installation `/opt/COMPSs/Dependencies/paraver/cfgs/`. A detailed list of all the available configurations can be found in *Paraver: configurations*.

The following guide uses a kmeans trace (result from executing the *Kmeans* sample code with the `--tracing` flag.) with the *compss_tasks.cfg* configuration file as an example to illustrate the basic usage of Paraver. After accepting the load of the configuration file, another window appears showing the view. Figure 24 and Figure 25 show an example of this process.

Caution: In a Paraver view, a red exclamation sign may appear in the bottom-left corner. This means that some event values are not being shown (because they are out of the current view scope), so little adjustments must be made to view the trace correctly:

- Fit window: modifies the view scope to fit and display all the events in the current window.
  - Right click on the trace window
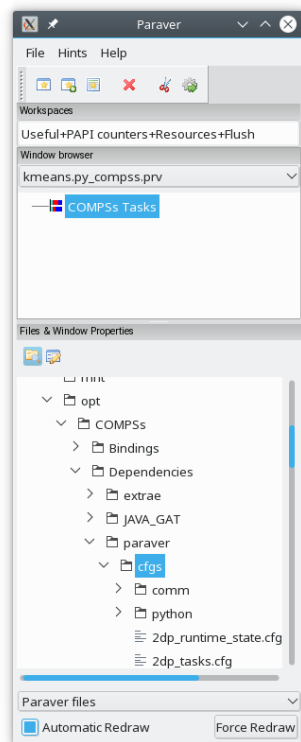
Figure 24: Paraver menu
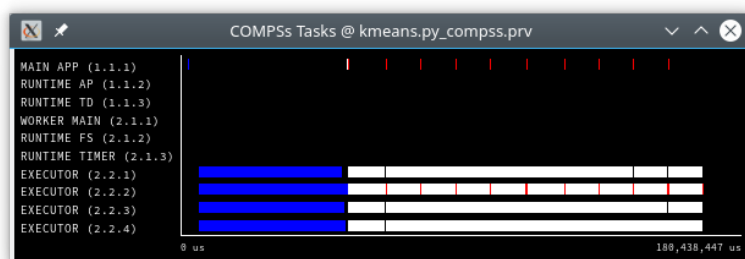
Figure 25: Kmeans Trace file

> – Choose the option Fit Semantic Scale / Fit Both

6.3.2.3 View Adjustment

- View Event Flags: marks with a green flag all the emitted events.
  - Right click on the trace window
  - Chose the option View / Event Flags
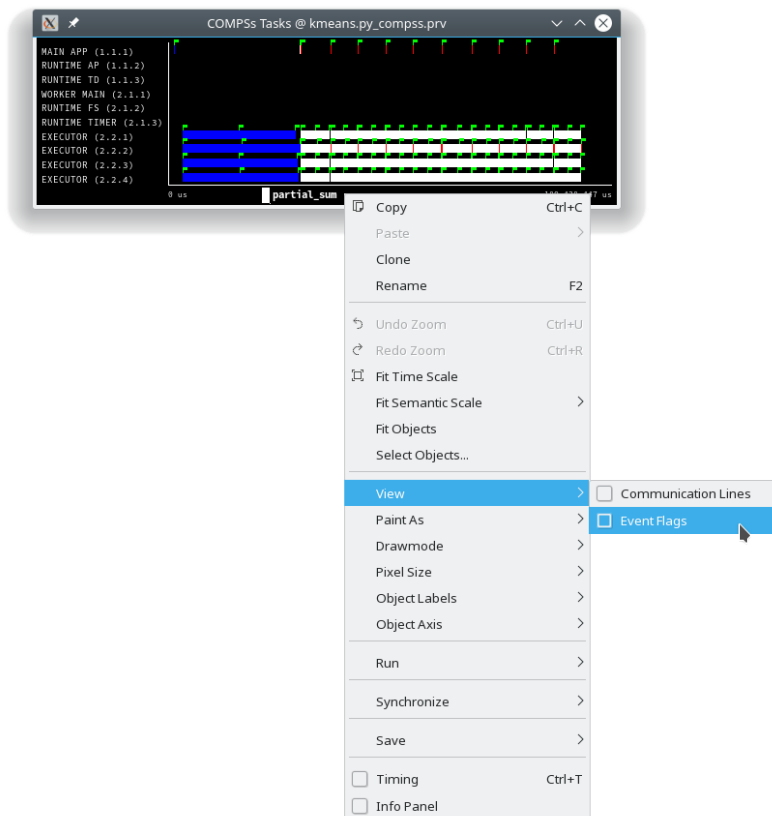
Figure 26: Paraver view adjustment: View Event Flags

- Show Info Panel: display the information panel. In the tab "Colors" we can see the legend of the colors shown in the view.
  - Right click on the trace window
  - Check the Info Panel option
  - Select the Colors tab in the panel

- Zoom: explore the tracefile more in-depth by zooming into the most relevant sections.
  - Select a region in the trace window to see that region in detail
  - Repeat the previous step as many times as needed
  - The undo-zoom option is in the right click panel
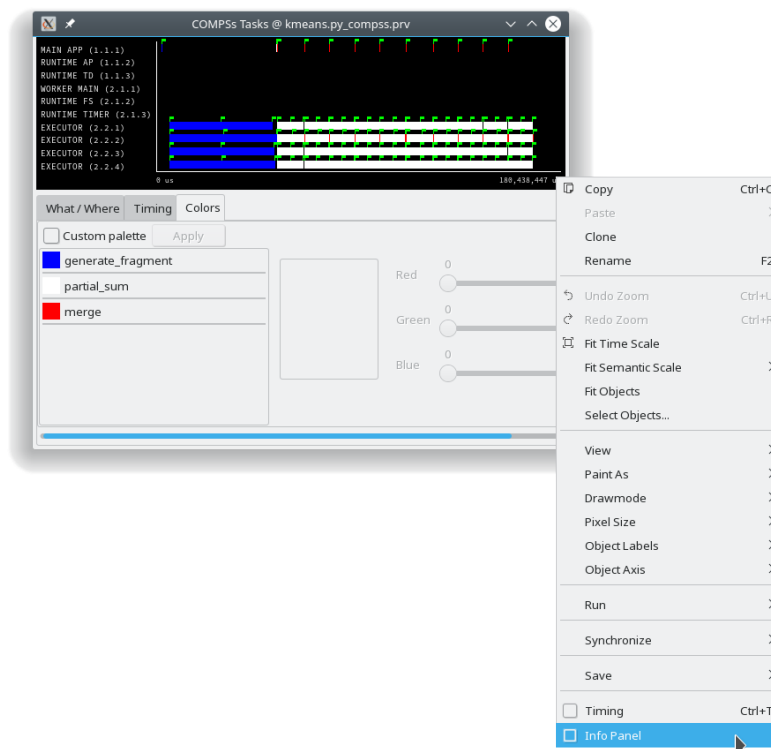
Figure 27: Paraver view adjustment: Show info panel
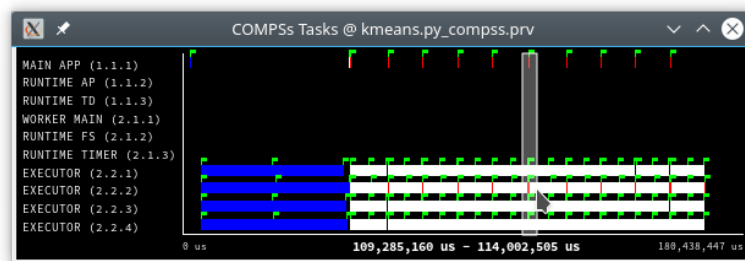
Figure 28: Paraver view adjustment: Zoom configuration
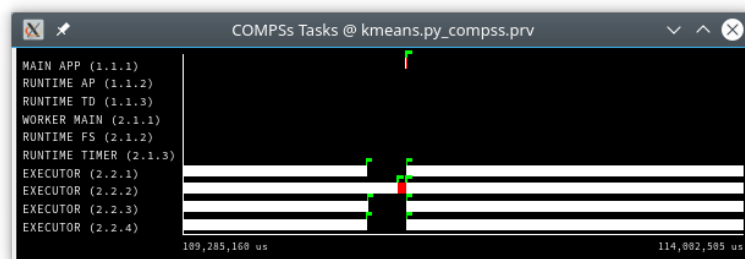
Figure 29: Paraver view adjustment: Zoom result

6.3.3 Interpretation

This section explains how to interpret a trace view once it has been adjusted as described in the previous section.

- The trace view has on its horizontal axis the execution time and on the vertical axis one line for the master at the top, and below it, one line for each of the workers.
- In a line, the black color is associated with an idle state, i.e. there is no event at that time.
- Whenever an event starts or ends a flag is shown.
- In the middle of an event, the line shows a different color. Colors are assigned depending on the event type.
- The info panel contains the legend of the assigned colors to each event type.
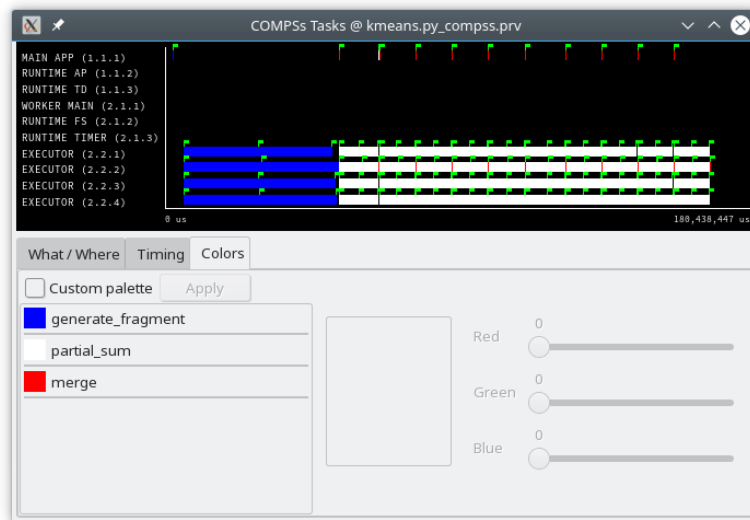
Figure 30: Trace interpretation

6.3.4 Analysis

This section gives some tips to analyze a COMPSs trace from two different points of view: graphically and numerically.

6.3.4.1 Graphical Analysis

The main concept is that computational events, the task events in this case, must be well distributed among all workers to have a good parallelism, and the duration of task events should be also balanced, this means, the duration of computational bursts.

In the previous trace view, all the tasks of type "generate_fragment" in dark blue appear to be well distributed among the four workers, each worker executor executes two "generate_fragment" tasks.

Next, a set of "partial_sum" tasks, coloured in white, are distributed across the four workers. In particular, eight "partial_sum" tasks are executed per kmeans iteration, so each worker executor executes two "partial_sum" tasks per iteration. This trace shows the execution of ten iterations. Note that all "partial_sum" tasks are very similar in time. This means that there is not much variability among them, and consequently not imbalance.

Finally, there is a "merge" task at the end of each iteration (coloured in red). This task is executed by one of the worker executors, and gathers the result from the previous eight "partial_sum" tasks. This task can be better displayed thanks to zoom.
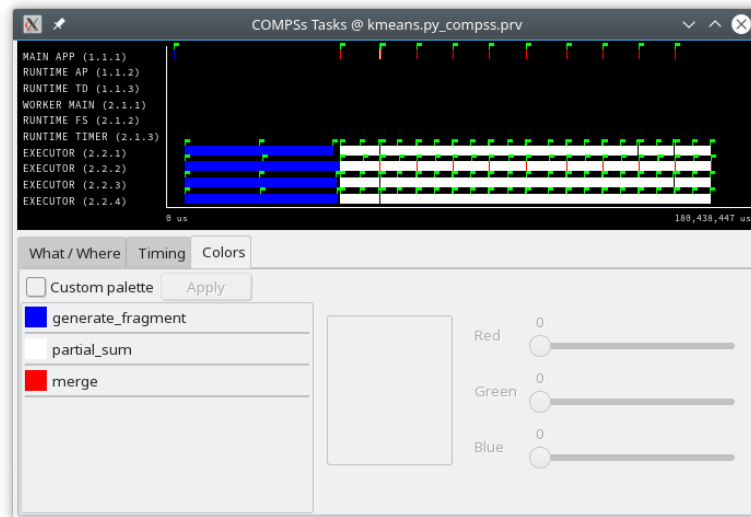
Figure 31: Basic trace view of a Kmeans execution.
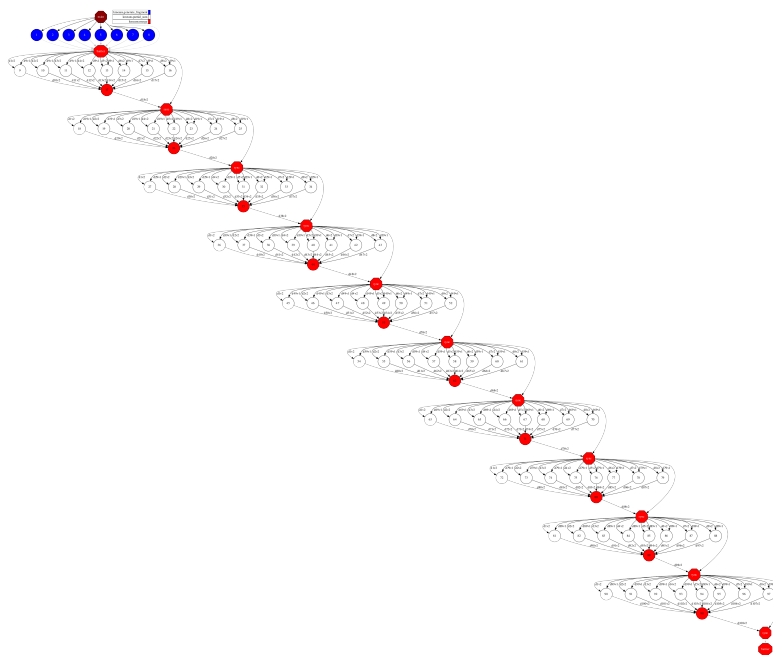
Figure 32: Data dependencies graph of a Kmeans execution.
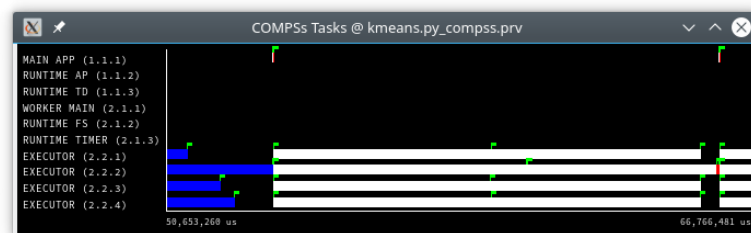
Figure 33: Zoomed in view of a Kmeans execution (first iteration).

6.3.4.2 Numerical Analysis

Here we analize the Kmeans trace numerically.
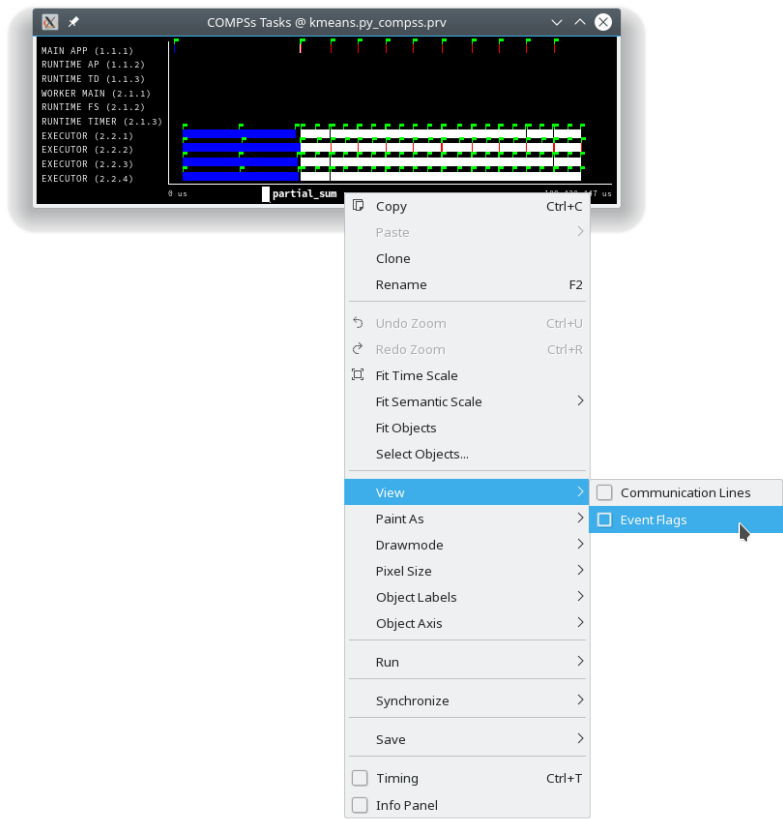
Figure 34: Original sample trace of a Kmeans execution to be analyzed

Paraver offers the possibility of having different histograms of the trace events. Click the "New Histogram" button in the main window and accept the default options in the "New Histogram" window that will appear.

Figure 35: Paraver Menu - New Histogram

After that, the following table is shown. In this case for each worker, the time spent executing each type of task is shown in gradient from light green for lower values to dark-blue for higher ones. The values coresponding to the colours and task names can be shown by clicking in the gray magnifying glass button. And the task corresponding to each task column can also be shown by clicking in the colur bars button.

The time spent executing each type of task is shown, and task names appear in the same color than in the trace view. The color of the cells in a row is kept, conforming a color based histogram.

The previous table also gives, at the end of each column, some extra statistical information for each type of tasks (as the total, average, maximum or minimum values, etc.).

In the window properties of the main window (Button Figure 39), it is possible to change the semantic of the statistics to see other factors rather than the time, for example, the number of bursts (Figure 40).

In the same way as before, the following table shows for each worker the number of bursts for each type of task, this is, the number or tasks executed of each type. Notice the gradient scale from light-green to dark-blue changes with the new values.
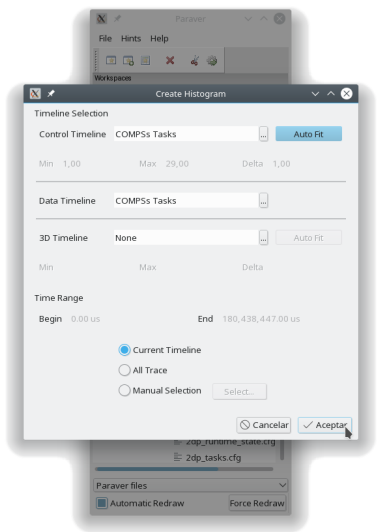
Figure 36: Histogram configuration (Accept default values)
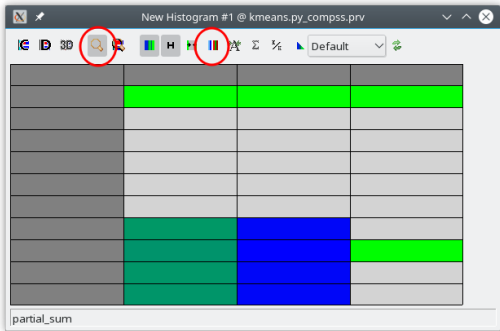
Figure 37: Kmeans histogram corresponding to previous trace

Figure 38: Kmeans numerical histogram corresponding to previous trace
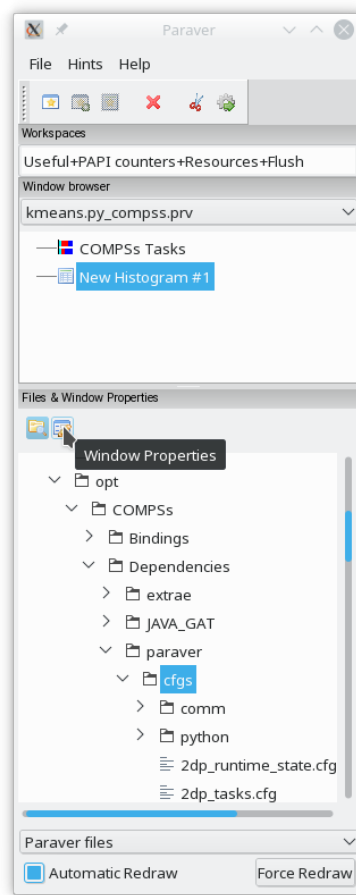
Figure 39: Paraver window properties button
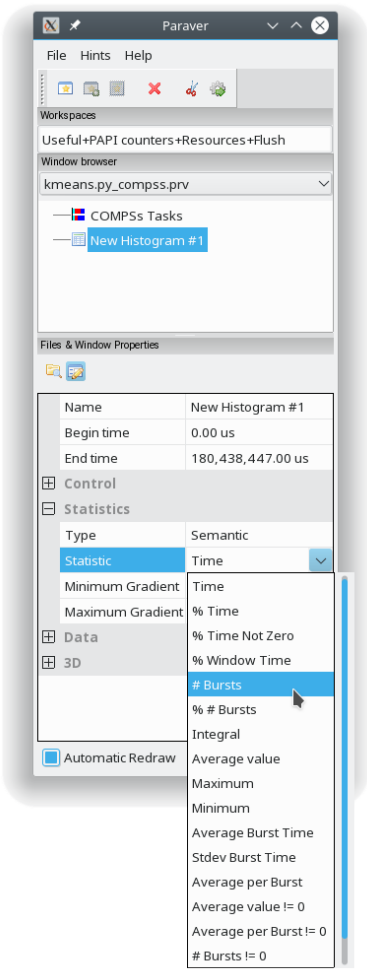
Figure 40: Paraver histogram options menu
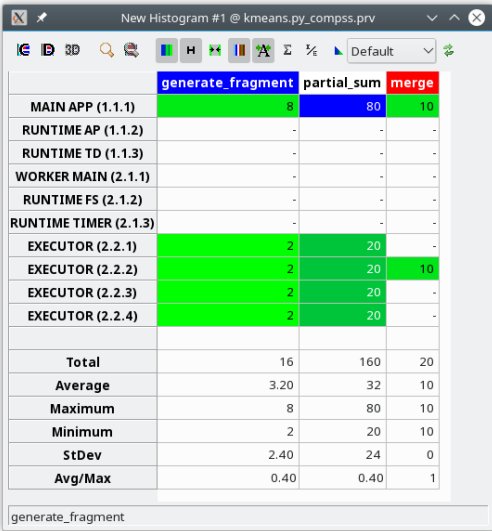
Figure 41: Kmeans histogram with the number of bursts

6.3.5 PAPI: Hardware Counters

The applications instrumentation supports hardware counters through the performance API (PAPI). In order to use it, PAPI needs to be present on the machine before installing COMPSs.

During COMPSs installation it is possible to check if PAPI has been detected in the Extrae config report:

```
Package configuration for Extrae VERSION based on extrae/trunk rev. XXXX:
-----------------------
Installation prefix: /opt/COMPSs/Dependencies/extrae
Cross compilation: no
...
...
...


Performance counters: yes
  Performance API: PAPI
  PAPI home: /usr
  Sampling support: yes
```

> **Caution:** PAPI detection is only performed in the machine where COMPSs is installed. User is responsible of providing a valid PAPI installation to the worker machines to be used (if they are different from the master), otherwise workers will crash because of the missing *libpapi.so*.

PAPI installation and requirements depend on the OS. On Ubuntu 14.04 it is available under *papi-tools* package; on OpenSuse *libpapi*, *papi* and *papi-devel* packages. For more information check https://icl.cs.utk.edu/projects/papi/wiki/Installing_PAPI.

Extrae only supports 8 active hardware counters at the same time. Both basic and advanced mode have the same default counters list:

PAPI_TOT_INS Instructions completed
PAPI_TOT_CYC Total cycles
PAPI_LD_INS Load instructions
PAPI_SR_INS Store instructions
PAPI_BR_UCN Unconditional branch instructions
PAPI_BR_CN Conditional branch instructions
PAPI_VEC_SP Single precision vector/SIMD instructions
RESOURCE_STALLS Cycles Allocation is stalled due to Resource Related reason

The XML config file contains a secondary set of counters. In order to activate it just change the *starting-set-distribution* from 2 to 1 under the *cpu* tag. The second set provides the following information:

PAPI_TOT_INS Instructions completed
PAPI_TOT_CYC Total cycles
PAPI_L1_DCM Level 1 data cache misses
PAPI_L2_DCM Level 2 data cache misses
PAPI_L3_TCM Level 3 cache misses
PAPI_FP_INS Floating point instructions

Tip: To find the available PAPI counters on a given computer issue the command:

```
$ papi_avail -a
```

And for more hardware counters:

```
$ papi_native_avail
```

To further customize the tracked counters, modify the XML to suit your needs. For more information about Extrae's XML configuration refer to https://www.bsc.es/computer-sciences/performance-tools/trace-generation/extrae/extrae-user-guide.

6.3.6 Paraver: configurations

Table 18, Table 19 and Table 20 provide information about the different pre-build configurations that are distributed with COMPSs and that can be found under the `/opt/COMPSs/Dependencies/ paraver/cfgs/` folder. The *cfgs* folder contains all the basic views, the *python* folder contains the configurations for Python events, and finally the *comm* folder contains the configurations related to communications.

Additionally, it can be shown the data transfers and the task dependencies. To see them it is needed to show communication lines in the paraver windows, to only see the task dependencies are needed to put in Filter > Communications > Comm size, the size equal to 0. Some of the dependencies between tasks may be lost.

Table 18: General paraver configurations for COMPSs Applications

Configuration File Name	Description	Targ
2dp_runtime_state.cfg	2D plot of runtime state	Run
2dp_tasks.cfg	2D plot of tasks duration	App
3dh_duration_runtime.cfg	3D Histogram of runtime execution	Run
3dh_duration_tasks.cfg	3D Histogram of tasks duration	App
compss_cpu_constraints.cfg	Shows tasks cpu constraints	Run
compss_executors.cfg	Shows the number of executor threads in each node	Run
compss_runtime.cfg	Shows COMPSs Runtime events (master and workers)	Run
compss_runtime_master.cfg	Shows COMPSs Runtime master events	Run
compss_storage.cfg	Shows COMPSs persistent storage events	Run
compss_tasks_and_runtime.cfg	Shows COMPSs Runtime events (master and workers) and tasks execution	App
compss_tasks.cfg	Shows tasks execution and tasks instantiation in master nodes	App
compss_tasks_communications.cfg	Shows tasks and communications	App
compss_tasks_cpu_affinity.cfg	Shows tasks CPU affinity	App
compss_tasks_dependencies.cfg	Shows tasks and dependencies (only for the master node)	App
compss_tasks_gpu_affinity.cfg	Shows tasks GPU affinity	App
compss_tasks_id.cfg	Shows tasks execution by task id	App
compss_tasks_runtime_&_agents.cfg	Shows COMPSs Agent and Runtime events and tasks execution	App
compss_waiting_tasks.cfg	Shows waiting tasks	Run
histograms_HW_counters.cfg	Shows hardware counters histograms	Both
instantiation_time.cfg	Shows the instantiation time	Run
Interval_between_runtime.cfg	Interval between runtime events	Run
nb_executing_tasks.cfg	Number of executing tasks	App
nb_requested_cpus.cfg	Number of requested CPUs	Run
nb_requested_disk_bw.cfg	Number of requested disk bandwidth	Run
nb_requested_gpus.cfg	Number of requested GPUs	Run
nb_executing_mem.cfg	Number of executing memory	Run
number_executors.cfg	Number of executors	Run
task_duration.cfg	Shows tasks duration	App
thread_cpu.cfg	Shows the initial executing CPU	Run
thread_identifiers.cfg	Shows the type of each thread	Run
time_btw_tasks.cfg	Shows the time between tasks	Run
user_events.cfg	Shows the user events (type `9100000`)	App

Table 19: Available paraver configurations for Python events of
COMPSs Applications

Configuration File Name	Description	Target
3dh_duration_runtime_-master_binding.cfg	3D Histogram of runtime events of python in master node	Python Binding
3dh_events_inside_-task.cfg	3D Histogram of python events	Python Binding
3dh_tasks_phase.cfg	3D Histogram of execution functions	Python Binding
compss_runtime_master_-binding.cfg	Shows runtime events of python in master node	Python Binding
deserialization_object_-number.cfg	Shows the numbers of the objects that are being deserialized	Python Binding
deserialization_size.cfg	Shows the size of the objects that are being deserialized (Bytes)	Python Binding
events_inside_tasks.cfg	Events showing python information such as user function execution time, modules imports, or serializations	Python Binding
events_in_workers.cfg	Events showing python binding information in worker	Python Binding
nb_user_code_execut-ing.cfg	Number of user code executing	Python Binding
serdes_bw.cfg	Serialization and deserializations bandwidth (MB/s)	Python Binding
serdes_cache_bw.cfg	Serialization and deserializations to cache bandwidth (MB/s)	Python Binding
serialization_object_num-ber.cfg	Shows the numbers of the objects that are being serialized	Python Binding
serialization_size.cfg	Shows the size of the objects that are being serialized (Bytes)	Python Binding
tasks_cpu_affinity.cfg	Events showing the CPU affinity of the tasks (shows only the first core if multiple assigned)	Python Binding
tasks_gpu_affinity.cfg	Events showing the GPU affinity of the tasks (shows only the first GPU if multiple assigned)	Python Binding
Time_between_events_in-side_tasks.cfg	Shows the time between events inside tasks	Python Binding

Table 20: Available paraver configurations for COMPSs Applica-
tions

Configuration File Name	Description	Target
communication_ma-trix.cfg	Table view of communications between each node	Runtime Communications
compss_data_trans-fers.cfg	Shows data transfers for each task's parameter	Runtime Communications
compss_tasksID_trans-fers.cfg	Task's transfers request for each task (task with its IDs are also shown)	Runtime Communications
process_bandwith.cfg	Send/Receive bandwith table for each node	Runtime Communications
receive_bandwith.cfg	Receive bandwith view for each node	Runtime Communications
send_bandwith.cfg	Send bandwith view for each node	Runtime Communications
sr_bandwith.cfg	Send/Receive bandwith view for each node	Runtime Communications

6.3.7 User Events in Python

Users can emit custom events inside their python **tasks**. Thanks to the fact that python is not a compiled language, users can emit events inside their own tasks using the available EXTRAE instrumentation object because it is already loaded and available in the PYTHONPATH when running with tracing enabled.

To emit an event first import pyextrae:

- `import pyextrae.sequential as pyextrae` to emit events from the main code.
- `import pyextrae.multiprocessing as pyextrae` to emit events within tasks code.

And then just use the call `pyextrae.event(type, id)` (or `pyextrae.eventandcounters (type, id)` if you also want to emit PAPI hardware counters).

Tip: It must be used a type number higher than `8000050` in order to avoid type conflicts.

We suggest to use `9100000` since we provide the `user_events.cfg` configuration file to visualize the user events of this type in PARAVER.

6.3.7.1 Events in main code

The following code snippet shows how to emit an event from the main code (or any other code which is not within a task). In this case it is necessary to import `pyextrae.sequential`.

```python
from pycompss.api.api import compss_wait_on
from pycompss.api.task import task
import pyextrae.sequential as pyextrae


@task(returns=1)
def increment(value):
    return value + 1

def main():
    value = 1
    pyextrae.eventandcounters(9100000, 2)
    result = increment(value)
    result = compss_wait_on(result)
    pyextrae.eventandcounters(9100000, 0)
    print("result: " + str(result))

if __name__ == "__main__":
    main()
```

6.3.7.2 Events in task code

The following code snippet shows how to emit an event from the task code. In this case it is necessary to import `pyextrae.multiprocessing`.

```python
from pycompss.api.task import task


@task()
def compute():
    import pyextrae.multiprocessing as pyextrae
    pyextrae.eventandcounters(9100000, 2)
    ...
    # Code to wrap within event 2
```

```
    ...
    pyextrae.eventandcounters(9100000, 0)
```

> **Caution:**   Please, note that the `import pyextrae.multiprocessing as pyextrae` is performed within the
> task. If the user needs to add more events to tasks within the same module (excluding the applicatin main
> module) and wants to put this import in the top of the module making `pyextrae` available for all of them, it
> is necessary to enable the tracing hook on the tasks that emit events:
>
> ```python
> from pycompss.api.task import task
> import pyextrae.multiprocessing as pyextrae
>
> @task(tracing_hook=True)
> def compute():
>     pyextrae.eventandcounters(9100000, 2)
>     ...
>     # Code to wrap within event 2
>     ...
>     pyextrae.eventandcounters(9100000, 0)
> ```
>
> The `tracing_hook` is disabled by default in order to reduce the overhead introduced by tracing avoiding to
> intercept all function calls within the task code.

6.3.7.3 Result trace

The events will appear automatically on the generated trace. In order to visualize them, just load the `user_-events.cfg` configuration file in PARAVER.

If a different type value is choosen, take the same `user_events.cfg` and go to `Window Properties -> Filter -> Events -> Event Type` and change the value labeled *Types* for your custom events type.

Tip: If you want to name the events, you will need to manually add them to the `.pcf` file with the corresponding
name for each `value`.

6.3.7.4 Practical example

Consider the following application where we define an event in the main code (1) and another within the task (2).
The `increment` task is invoked 8 times (with a mimic computation time of the value received as parameter.)

```python
from pycompss.api.api import compss_wait_on
from pycompss.api.task import task
import time

@task(returns=1)
def increment(value):
    import pyextrae.multiprocessing as pyextrae
    pyextrae.eventandcounters(9100000, 2)
    time.sleep(value)  # mimic some computation
    pyextrae.eventandcounters(9100000, 0)
    return value + 1

def main():
    import pyextrae.sequential as pyextrae
    elements = [1, 2, 3, 4, 5, 6, 7, 8]
```

```python
    results = []
    pyextrae.eventandcounters(9100000, 1)
    for element in elements:
        results.append(increment(element))
    results = compss_wait_on(results)
    pyextrae.eventandcounters(9100000, 0)
    print("results: " + str(results))

if __name__ == "__main__":
    main()
```

After launching with tracing enabled (`-t` flag), the trace has been generated into the logs folder:

- `$HOME/.COMPSs/events.py_01/trace` if using `runcompss`.
- `$HOME/.COMPSs/<JOB_ID>/trace` if using `enqueue_compss`.

Now it is time to modify the `.pcf` file including the folling text at the end of the file with your favourite text editor:

```
EVENT_TYPE
0    9100000     User events
VALUES
0       End
1       Main code event
2       Task event
```

> **Caution:** Keep value 0 with the End message.
>
> Add all values defined in the application with a descriptive short name to ease the event identification in PARAVER.

Open PARAVER, load the tracefile (`.prv`) and open the `user_events.cfg` configuration file. The result (see Figure 42) shows that there are 8 "Task event" (in white), and 1 "Main code event" (in blue) as we expected. Their length can be seen with the event flags (green flags), and measured by double clicking on the event of interest.
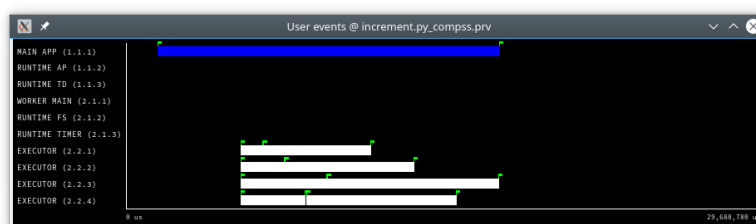
Figure 42: User events trace file

Paraver uses by default the `.pcf` with the same name as the tracefile so if you add them to one, you can reuse it just by changing its name to the tracefile.

6.4 Data Provenance

In order to achieve **Reproducibility** and **Replicability** with your experiments using COMPSs, the runtime includes the capacity of recording details of the application's execution, also known as *Data Provenance.* This is supported for both Python and Java COMPSs applications.

When the provenance option is activated, the runtime records every access to a file or directory specified in the application, as well as its direction (IN, OUT, INOUT). In addition to this, other information such as the parameters passed as inputs in the command line that submitted the application, its source files, workflow image and profiling statistics, authors and their institutions, ... are also stored. All this information is later used to record the Data Provenance of your workflow using the RO-Crate specification, and with the assistance of the ro-crate-py library. RO-Crate is based on JSON-LD (JavaScript Object Notation for Linked Data), is much simpler than other standards and tools created to record Provenance, and that is why it has been adopted in a number of communities. Using RO-Crate to register the execution's information ensures not only to register correctly the Provenance of a COMPSs application run, but also compatibility with some existing portals that already embrace RO-Crate as their core format for representing metadata, such as WorkflowHub.

6.4.1 Software dependencies

Provenance generation in COMPSs depends on the ro-crate-py library, thus, it must be installed before the provenance option can be used. Depending on the target system, different options are available using `pip`:

If the installation is in a laptop or machine you manage, you can use the command:

```
$ pip install rocrate
```

If you do not manage the target machine, you can install the library in your own user space using:

```
$ pip install rocrate --user
```

This would typically install the library in `~/.local/`. Another option is to specify the target directory with:

```
$ pip install -t install_path rocrate
```

Our implementation has been tested with `ro-crate-py` version `0.7.0` and earlier.

6.4.2 Previous needed information

There are certain pieces of information which must be included when registering the provenance of a workflow that the COMPSs runtime cannot automatically infer, such as the authors of an application. For specifying all these fields that are needed to generate an RO-Crate but cannot be automatically obtained, we have created a simple YAML structure where the user can specify them. They need to provide in their working directory (i.e., where the application is going to be run) a YAML file named `ro-crate-info.yaml` that follows the next template structure:

```yaml
COMPSs Workflow Information:
  name: Name of your COMPSs application
  description: Detailed description of your COMPSs application
  license: Apache-2.0 #Provide better a URL, but these strings are accepted:
                  # https://about.workflowhub.eu/Workflow-RO-Crate/#supported-licenses
  sources_dir: [path_to_dir_1, path_to_dir_2]   # Optional: List of directories containing the␣
↪application source files.
        # Relative or absolute paths can be used
  sources_main_file: my_main_file.py  # Optional: Name of the main file of the application,␣
↪located in one of the
        # sources_dir. Relative paths from a sources_dir entry, or absolute paths can be used
  files: [main_file.py, aux_file_1.py, aux_file_2.py] # List of application files
        # Relative or absolute paths can be used
```

(continues on next page)

```
Authors:
 - name: Author_1 Name
   e-mail: author_1@email.com
   orcid: https://orcid.org/XXXX-XXXX-XXXX-XXXX
   organisation_name: Institution_1 name
   ror: https://ror.org/XXXXXXXX # Find them in ror.org
 - name: Author_2 Name
   e-mail: author2@email.com
   orcid: https://orcid.org/YYYY-YYYY-YYYY-YYYY
   organisation_name: Institution_2 name
   ror: https://ror.org/YYYYYYYYY # Find them in ror.org
```

> **Warning:** If no YAML file is provided, the runtime will fail to generate provenance, and will automatically generate an `ro-crate-info_TEMPLATE.yaml` file that the user can edit to add their details.

As you can see, there are two main blocks in the YAML:

- **COMPSs Workflow Information:** Where details on the application are provided.
- **Authors:** Where authors' details are given.

More specifically, in the **COMPSs Workflow Information** section:

- The `name` and `description` fields are free text, where a long name and description of the application must be provided.
- The `license` field is preferred to be specified by providing an URL to the license, but a set of predefined strings are also supported, and can be found here: https://about.workflowhub.eu/Workflow-RO-Crate/#supported-licenses
- `sources_dir` can be a single path, or a list of paths where application source files can be found. Our script will add ALL files (i.e., not only source files, but any file found) and sub-directories inside each of the paths specified. The sub-directories structure is respected when the files are added in the crate (inside a sub-directory `application_sources`).
- `sources_main_file` is the name of the main source file of the application, and may be specified if the user wants to select a particular file as such. The COMPSs runtime detects automatically the main source of an application, therefore this is a way to override the detected file. The file can be specified with a relative path inside one of the directories listed in `sources_dir`. An absolute path can also be used.
- `files` is a single or a list of all the source files of the application (typically all `.py` files for Python applications, or `.java`, `.class`, `.jar` files for Java ones). Both relative and absolute paths can be used. All files specified here will be added in the root of the sub-directory `application_sources` from the resulting crate. If the script is unable to automatically identify the main source file of the application, the first file of this list may be considered as such.

The `sources_dir` and `files` terms are complementary to each other. An `ro-crate-info.yaml` could use the term `files` alone or `sources_dir` alone, but also both, if the user is willing to add a number of sub-directories with source files, but also several files by hand.

> **Warning:** The term `sources_main_file` can only be used when `sources_dir` is defined. While the runtime is able to detect automatically the main file from application execution, this would enable to modify that automatic selection in case of need.

And in the **Authors** section:

- `name`, `e-mail` and `organisation_name` are strings corresponding to the author's name, e-mail and their institution. They are free text, but the `e-mail` field must follow the `user@domain.top` format.
- `orcid` refers to the ORCID identifier of the author. The IDs can be found and created at https://orcid.org/
- `ror` refers to the Research Organization Registry (ROR) identifier for an institution. They can be found at http://ror.org/

Tip: It is very important that the list of source files (defined with `sources_dir` or `files`), `orcid` and `ror` terms are correctly defined, since the runtime will only register information for the list of source files defined, and the `orcid` and `ror` are used as unique identifiers in the RO-Crate specification.

In the following lines, we provide a YAML example for an out-of-core Matrix Multiplication PyCOMPSs application, distributed with license Apache v2.0, with 2 source files, and authored by 3 persons from two different institutions.

```
COMPSs Workflow Information:
  name: COMPSs Matrix Multiplication, out-of-core using files
  description: Hypermatrix size 2x2 blocks, block size 2x2 elements
  license: Apache-2.0 #Provide better a URL, but these strings are accepted:
                      # https://about.workflowhub.eu/Workflow-RO-Crate/#supported-licenses
  files: [matmul_directory.py, matmul_tasks.py]

Authors:
  - name: Raül Sirvent
    e-mail: Raul.Sirvent@bsc.es
    orcid: https://orcid.org/0000-0003-0606-2512
    organisation_name: Barcelona Supercomputing Center
    ror: https://ror.org/05sd8tv96
  - name: Rosa M. Badia
    e-mail: Rosa.M.Badia@bsc.es
    orcid: https://orcid.org/0000-0003-2941-5499
    organisation_name: Barcelona Supercomputing Center
    ror: https://ror.org/05sd8tv96
  - name: Adam Hospital
    e-mail: adam.hospital@irbbarcelona.org
    orcid: https://orcid.org/0000-0002-8291-8071
    organisation_name: IRB Barcelona
    ror: https://ror.org/01z1gye03
```

Also, another example of a COMPSs Java K-means application, where the usage of the `sources_dir` term can be seen. We add to the crate the sub-directories that contain the `.jar` and `.java` files correspondingly.

```
COMPSs Workflow Information:
  name: COMPSs K-means
  description: K-means clustering is a method of cluster analysis that aims to partition ''n'
↪' points into ''k''
    clusters in which each point belongs to the cluster with the nearest mean. It follows an␣
↪iterative refinement
    strategy to find the centers of natural clusters in the data.
  license: https://opensource.org/licenses/Apache-2.0 #Provide better a URL, but these␣
↪strings are accepted:
                      # https://about.workflowhub.eu/Workflow-RO-Crate/#supported-licenses
  sources_dir: [jar/, src/]

Authors:
  - name: Raül Sirvent
    e-mail: Raul.Sirvent@bsc.es
    orcid: https://orcid.org/0000-0003-0606-2512
    organisation_name: Barcelona Supercomputing Center
    ror: https://ror.org/05sd8tv96
```

6.4.3 Usage

The way of activating the recording of Data Provenance with COMPSs is very simple. One must only enable
the `-p` or `--provenance` flag when using `runcomps` or `enqueue_compss` to run or submit a COMPSs application,
respectively. As shown in the help option:

```
$ runcompss -h

(...)
--provenance, -p    Generate COMPSs workflow provenance data in RO-Crate format from YAML␣
↪file. Automatically
                    activates -graph and -output_profile.
                    Default: false
```

> **Warning:**   As stated in the help, provenance automatically activates both `--graph` and `--output_profile`
> options. Consider that the graph image generation can take some extra seconds at the end of the execution of
> your application, therefore, adjust the `--exec_time` accordingly.

In the case of extremely large workflows (e.g., a workflow with tenths of thousands of task nodes, or tenths of
thousands of files used as inputs or outputs), the extra time needed to generate the data provenance with RO-Crate
may be a problem in systems with strict run time constraints. In these cases, the workflow execution may end
correctly, but the extra processing to generate the provenance may be killed by the system if it exceeds a certain
limit, and the provenance will not be created correctly.

For this or any other similar situation, our data provenance generation script can be triggered offline at any moment
after the workflow has executed correctly, thanks to our design. From the working directory of the application,
the following commands may be used:

```
$ $COMPSS_HOME/Runtime/scripts/utils/compss_gengraph svg $BASE_LOG_DIR/monitor/complete_graph.
↪dot

$ python $COMPSS_HOME/Runtime/scripts/system/provenance/generate_COMPSs_RO-Crate.py ro-crate-
↪info.yaml $BASE_LOG_DIR/dataprovenance.log
```

In these commands, `COMPSS_HOME` is where your COMPSs installation is located, and `BASE_LOG_DIR` points to the
path where the application run logs are stored (see Section *Logs* for more details on where to locate these logs).
`compss_gengraph` generates the workflow image to be added to the crate, but if its generation time is a concern, or
the user does not want it to be included in the crate, the command can be skipped. The second command runs the
`generate_COMPSs_RO-Crate.py` Python script, that uses the information provided by the user in `ro-crate-info.`
`yaml` combined with the file accesses information registered by the COMPSs runtime in the `dataprovenance.log`
file. The result is a sub-directory `COMPSs_RO-Crate_[uuid]/` that contains the data provenance of the run (see
next sub-section for a detailed description).

6.4.4 Result

Once the application has finished, a new sub-folder under the application's Working Directory will be created with
the name `COMPSs_RO-Crate_[uuid]/`, which is also known as *crate*. The contents of the folder include all the
elements needed to reproduce a COMPSs execution, and are:

- **Application Source Files:** As detailed by the user in the `ro-crate-info.yaml` file, with the terms
  `sources_dir` and/or `files`. They have to include the main source file and all auxiliary files that the
  application needs (e.g.: `.py`, `.java`, `.class` or `.jar`). Optionally, the term `sources_main_file` can be used
  to manually select the main source file of the application. All application files are added to a sub-folder in the
  crate named `application_sources`, where the `sources_dir` locations are included with their same folder
  tree structure. The files included with the `files` term are added to the root of the `application_sources`
  sub-folder in the crate.

- **complete_graph.svg:** The image of the workflow generated by the COMPSs runtime, as generated with the `runcompss -g` or `--graph` option.
- **App_Profile.json:** A set of statistics of the application run recorded by the COMPSs runtime, as if the `runcompss --output_profile=<path>` option was enabled. It includes, for each resource and method executed: number of executions of the specific method, as well as maximum, average and minimum run time. The name of the file can be customized using the `--output_profile=<path>` option.
- **compss_command_line_arguments.txt:** Stores the options passed by the command line when the application was submitted. This is very important for reproducing a COMPSs application, since input parameters could even potentially change the resulting workflow generated by the COMPSs runtime.
- **ro-crate-metadata.json:** The RO-Crate JSON main file describing the contents of this directory (crate) in the RO-Crate specification format. You can find examples in the following Sections.

> **Warning:** All previous file names (`complete_graph.svg`, `App_Profile.json` and `compss_command_line_-arguments.txt`) are automatically used to generate new files when using the `-p` or `--provenance` option. Avoid using these file names among your own files to avoid unwanted overwritings. You can change the resulting `App_-Profile.json` name by using the `--output_profile=/path_to/file` flag.

6.4.5 Log and time statistics

When the provenance generation is activated, and after the application has finished, the data provenance generation script will be automatically triggered. A number of log messages related to provenance can bee seen, which return interesting information regarding the provenance generation process. They can all be filtered by doing a `grep` in the output log of the application using the `PROVENANCE` expression.

```
PROVENANCE | GENERATING GRAPH FOR DATA PROVENANCE
Output file: /Users/rsirvent/.COMPSs/matmul_directory.py_07//monitor/complete_graph.svg
INFO: Generating Graph with legend
DONE
PROVENANCE | ENDED GENERATING GRAPH FOR DATA PROVENANCE. TIME: 0
```

This first block indicates that the workflow image in SVG format is being generated. When this part finishes, the time in seconds will be reported. As mentioned earlier, complex workflows can lead to large graph generation times.

```
PROVENANCE | RUNNING DATA PROVENANCE SCRIPT
PROVENANCE | Number of source files detected: 2
PROVENANCE | COMPSs version: 3.0.rc2210, main_entity is: /Users/rsirvent/COMPSs-DP/matmul_
↪directory/matmul_directory.py, out_profile is: App_Profile.json
```

This second block details how many source files have been detected from the `sources_dir` and `files` terms defined in the `ro-crate-py.yaml` file. It also shows the COMPSs version detected, the mainEntity detected (i.e., the source file that contains the main method from the COMPSs application), and the name of the file containing the execution profile of the application.

```
PROVENANCE | RO-CRATE data_provenance.log processing TIME (process_accessed_files): 0.
↪00011706352233886719 s
PROVENANCE | RO-CRATE adding physical files TIME (add_file_to_crate): 0.001096963882446289 s
PROVENANCE | RO-CRATE adding input files' references TIME (add_file_not_in_crate): 0.
↪001238107681274414 s
PROVENANCE | RO-CRATE adding output files' references TIME (add_file_not_in_crate): 0.
↪00026798248291015625 s
```

The third block provides a set of times to understand if any overhead is caused by the script. The first time is the time taken to process the data_provenance.log. The second is the time taken to add the files that are included physically in the crate (this is, source files, workflow image, . . .). And the third and the fourth are the times spent by the script to add all input and output files of the workflow as references in the RO-Crate, respectively.

```
PROVENANCE | COMPSs RO-Crate created successfully in subfolder COMPSs_RO-Crate_aaf0cb82-a500-
↪4c28-bbc8-439c37c2e210/
PROVENANCE | RO-CRATE dump TIME: 0.004969120025634766 s
PROVENANCE | RO-CRATE GENERATION TOTAL EXECUTION TIME: 0.014089107513427734 s
PROVENANCE | ENDED DATA PROVENANCE SCRIPT
```

The fourth and final block details the name of the sub-folder where the RO-Crate has been generated, while stating the time to record the `ro-crate-metadata.json` file to disk, and the total time execution of the whole script.

6.4.6 ro-crate-metadata.json PyCOMPSs example (Laptop)

In the RO-Crate specification, the root file containing the metadata referring to the crate created is named `ro-crate-metadata.json`. In these lines, we provide an example of an ro-crate-metadata.json file resulting from a PyCOMPSs application execution in a laptop, specifically an out-of-core matrix multiplication example that includes matrices `A` and `B` as inputs in an `inputs/` sub-directory, and matrix `C` as the result of their multiplication (which in the code is also passed as input, to have a matrix initialized with 0s). For all the specific details on the fields provided in the JSON file, please refer to the RO-Crate specification Website. Intuitively, if you search through the JSON file you can find several interesting terms:

- **creator:** List of authors, identified by their ORCID.
- **publisher:** Organisations of the authors.
- **hasPart in ./:** lists all the files and directories this workflow needs and generates, and also the ones included in the crate. The URIs point to the hostname where the application has been run, thus, tells the user where the inputs and outputs can be found (in this example, a BSC laptop).
- **ComputationalWorkflow:** Main file of the application (in the example, `application_sources/matmul_-directory.py`). Includes the `inputs` and `outputs` needed and generated by the workflow, and a reference to the generated workflow image in the `image` field.
- **version:** The COMPSs specific version and build used to run this application. In the example: `3.0.rc2210`. This is a very important field to achieve reproducibility or replicability, since COMPSs features may vary their behaviour in different versions of the programming model runtime.

We encourage the reader to navigate through this `ro-crate-metadata.json` file example to get familiar with its contents. Many of the fields are easily and directly understandable.

```json
{
    "@context": "https://w3id.org/ro/crate/1.1/context",
    "@graph": [
        {
            "@id": "./",
            "@type": "Dataset",
            "creator": [
                {
                    "@id": "https://orcid.org/0000-0003-0606-2512"
                },
                {
                    "@id": "https://orcid.org/0000-0002-8291-8071"
                },
                {
                    "@id": "https://orcid.org/0000-0003-2941-5499"
                }
            ],
            "datePublished": "2022-10-24T08:52:07+00:00",
            "description": "Hypermatrix size 2x2 blocks, block size 2x2 elements",
            "hasPart": [
                {
                    "@id": "application_sources/matmul_directory.py"
                },
```

(continues on next page)

```
                        {
                              "@id": "complete_graph.svg"
                        },
                        {
                              "@id": "App_Profile.json"
                        },
                        {
                              "@id": "compss_command_line_arguments.txt"
                        },
                        {
                              "@id": "application_sources/matmul_tasks.py"
                        },
                        {
                              "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/inputs/A/A.0.0"
                        },
                        {
                              "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/inputs/A/A.0.1"
                        },
                        {
                              "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/inputs/A/A.1.0"
                        },
                        {
                              "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/inputs/A/A.1.1"
                        },
                        {
                              "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/inputs/B/B.0.0"
                        },
                        {
                              "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/inputs/B/B.0.1"
                        },
                        {
                              "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/inputs/B/B.1.0"
                        },
                        {
                              "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/inputs/B/B.1.1"
                        },
                        {
                              "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/inputs/"
                        },
                        {
                              "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/C.0.0"
                        },
                        {
                              "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/C.0.1"
                        },
```

```
                {
                    "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/C.1.0"
                },
                {
                    "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/C.1.1"
                }
            ],
            "license": "Apache-2.0",
            "mainEntity": {
                "@id": "application_sources/matmul_directory.py"
            },
            "name": "COMPSs Matrix Multiplication, out-of-core using files",
            "publisher": [
                {
                    "@id": "https://ror.org/05sd8tv96"
                },
                {
                    "@id": "https://ror.org/01z1gye03"
                }
            ]
        },
        {
            "@id": "ro-crate-metadata.json",
            "@type": "CreativeWork",
            "about": {
                "@id": "./"
            },
            "conformsTo": [
                {
                    "@id": "https://w3id.org/ro/crate/1.1"
                },
                {
                    "@id": "https://w3id.org/workflowhub/workflow-ro-crate/1.0"
                }
            ]
        },
        {
            "@id": "https://orcid.org/0000-0003-0606-2512",
            "@type": "Person",
            "affiliation": {
                "@id": "https://ror.org/05sd8tv96"
            },
            "contactPoint": {
                "@id": "mailto:Raul.Sirvent@bsc.es"
            },
            "name": "Ra\u00fcl Sirvent"
        },
        {
            "@id": "mailto:Raul.Sirvent@bsc.es",
            "@type": "ContactPoint",
            "contactType": "Author",
            "email": "Raul.Sirvent@bsc.es",
            "identifier": "Raul.Sirvent@bsc.es",
            "url": "https://orcid.org/0000-0003-0606-2512"
```

```
        },
        {
            "@id": "https://ror.org/05sd8tv96",
            "@type": "Organization",
            "name": "Barcelona Supercomputing Center"
        },
        {
            "@id": "https://orcid.org/0000-0003-2941-5499",
            "@type": "Person",
            "affiliation": {
                "@id": "https://ror.org/05sd8tv96"
            },
            "contactPoint": {
                "@id": "mailto:Rosa.M.Badia@bsc.es"
            },
            "name": "Rosa M. Badia"
        },
        {
            "@id": "mailto:Rosa.M.Badia@bsc.es",
            "@type": "ContactPoint",
            "contactType": "Author",
            "email": "Rosa.M.Badia@bsc.es",
            "identifier": "Rosa.M.Badia@bsc.es",
            "url": "https://orcid.org/0000-0003-2941-5499"
        },
        {
            "@id": "https://orcid.org/0000-0002-8291-8071",
            "@type": "Person",
            "affiliation": {
                "@id": "https://ror.org/01z1gye03"
            },
            "contactPoint": {
                "@id": "mailto:adam.hospital@irbbarcelona.org"
            },
            "name": "Adam Hospital"
        },
        {
            "@id": "mailto:adam.hospital@irbbarcelona.org",
            "@type": "ContactPoint",
            "contactType": "Author",
            "email": "adam.hospital@irbbarcelona.org",
            "identifier": "adam.hospital@irbbarcelona.org",
            "url": "https://orcid.org/0000-0002-8291-8071"
        },
        {
            "@id": "https://ror.org/01z1gye03",
            "@type": "Organization",
            "name": "IRB Barcelona"
        },
        {
            "@id": "application_sources/matmul_directory.py",
            "@type": [
                "File",
                "SoftwareSourceCode",
                "ComputationalWorkflow"
            ],
```

```
            "contentSize": 2151,
            "description": "Main file of the COMPSs workflow source files",
            "encodingFormat": "text/plain",
            "image": {
                "@id": "complete_graph.svg"
            },
            "input": [
                {
                    "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/inputs/"
                },
                {
                    "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/C.0.0"
                },
                {
                    "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/C.0.1"
                },
                {
                    "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/C.1.0"
                },
                {
                    "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/C.1.1"
                }
            ],
            "name": "matmul_directory.py",
            "output": [
                {
                    "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/C.0.0"
                },
                {
                    "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/C.0.1"
                },
                {
                    "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/C.1.0"
                },
                {
                    "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/C.1.1"
                }
            ],
            "programmingLanguage": {
                "@id": "#compss"
            }
        },
        {
            "@id": "#compss",
            "@type": "ComputerLanguage",
            "alternateName": "COMPSs",
            "citation": "https://doi.org/10.1007/s10723-013-9272-5",
```

```
        "name": "COMPSs Programming Model",
        "url": "http://compss.bsc.es/",
        "version": "3.0.rc2210"
    },
    {

        "@id": "https://www.nationalarchives.gov.uk/PRONOM/fmt/92",
        "@type": "WebSite",
        "name": "Scalable Vector Graphics"
    },
    {

        "@id": "complete_graph.svg",
        "@type": [
            "File",
            "ImageObject",
            "WorkflowSketch"
        ],
        "about": {
            "@id": "application_sources/matmul_directory.py"
        },
        "contentSize": 6163,
        "description": "The graph diagram of the workflow, automatically generated by␣
→COMPSs runtime",
        "encodingFormat": [
            [
                "image/svg+xml",
                {
                    "@id": "https://www.nationalarchives.gov.uk/PRONOM/fmt/92"
                }
            ]
        ],
        "name": "complete_graph.svg"
    },
    {

        "@id": "https://www.nationalarchives.gov.uk/PRONOM/fmt/817",
        "@type": "WebSite",
        "name": "JSON Data Interchange Format"
    },
    {

        "@id": "App_Profile.json",
        "@type": "File",
        "contentSize": 248,
        "description": "COMPSs application Tasks profile",
        "encodingFormat": [
            "application/json",
            {
                "@id": "https://www.nationalarchives.gov.uk/PRONOM/fmt/817"
            }
        ],
        "name": "App_Profile.json"
    },
    {

        "@id": "compss_command_line_arguments.txt",
        "@type": "File",
        "contentSize": 24,
        "description": "COMPSs command line execution command, including parameters passed
→",
```

```
            "encodingFormat": "text/plain",
            "name": "compss_command_line_arguments.txt"
        },
        {
            "@id": "application_sources/matmul_tasks.py",
            "@type": "File",
            "contentSize": 1721,
            "description": "Auxiliary File",
            "encodingFormat": "text/plain",
            "name": "matmul_tasks.py"
        },
        {
            "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_directory/
↪inputs/A/A.0.0",
            "@type": "File",
            "contentSize": 16,
            "dateModified": "2022-05-17T10:14:33",
            "name": "A.0.0",
            "sdDatePublished": "2022-10-24T08:52:07+00:00"
        },
        {
            "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_directory/
↪inputs/A/A.0.1",
            "@type": "File",
            "contentSize": 16,
            "dateModified": "2022-05-17T10:14:33",
            "name": "A.0.1",
            "sdDatePublished": "2022-10-24T08:52:07+00:00"
        },
        {
            "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_directory/
↪inputs/A/A.1.0",
            "@type": "File",
            "contentSize": 16,
            "dateModified": "2022-05-17T10:14:33",
            "name": "A.1.0",
            "sdDatePublished": "2022-10-24T08:52:07+00:00"
        },
        {
            "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_directory/
↪inputs/A/A.1.1",
            "@type": "File",
            "contentSize": 16,
            "dateModified": "2022-05-17T10:14:33",
            "name": "A.1.1",
            "sdDatePublished": "2022-10-24T08:52:07+00:00"
        },
        {
            "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_directory/
↪inputs/B/B.0.0",
            "@type": "File",
            "contentSize": 16,
            "dateModified": "2022-05-17T10:14:33",
            "name": "B.0.0",
            "sdDatePublished": "2022-10-24T08:52:07+00:00"
        },
```

```
        {
            "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_directory/
→inputs/B/B.0.1",
            "@type": "File",
            "contentSize": 16,
            "dateModified": "2022-05-17T10:14:33",
            "name": "B.0.1",
            "sdDatePublished": "2022-10-24T08:52:07+00:00"
        },
        {
            "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_directory/
→inputs/B/B.1.0",
            "@type": "File",
            "contentSize": 16,
            "dateModified": "2022-05-17T10:14:33",
            "name": "B.1.0",
            "sdDatePublished": "2022-10-24T08:52:07+00:00"
        },
        {
            "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_directory/
→inputs/B/B.1.1",
            "@type": "File",
            "contentSize": 16,
            "dateModified": "2022-05-17T10:14:33",
            "name": "B.1.1",
            "sdDatePublished": "2022-10-24T08:52:07+00:00"
        },
        {
            "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_directory/
→inputs/",
            "@type": "Dataset",
            "dateModified": "2022-05-17T10:14:33",
            "hasPart": [
                {
                    "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/inputs/A/A.0.0"
                },
                {
                    "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/inputs/A/A.0.1"
                },
                {
                    "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/inputs/A/A.1.0"
                },
                {
                    "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/inputs/A/A.1.1"
                },
                {
                    "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/inputs/B/B.0.0"
                },
                {
                    "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/inputs/B/B.0.1"
```

```
                    },
                    {
                        "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/inputs/B/B.1.0"
                    },
                    {
                        "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_
→directory/inputs/B/B.1.1"
                    }
                ],
                "name": "inputs",
                "sdDatePublished": "2022-10-24T08:52:07+00:00"
            },
            {

                "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_directory/C.0.0
→",

                "@type": "File",
                "contentSize": 20,
                "dateModified": "2022-10-24T08:52:02",
                "name": "C.0.0",
                "sdDatePublished": "2022-10-24T08:52:07+00:00"
            },
            {

                "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_directory/C.0.1
→",

                "@type": "File",
                "contentSize": 20,
                "dateModified": "2022-10-24T08:52:02",
                "name": "C.0.1",
                "sdDatePublished": "2022-10-24T08:52:07+00:00"
            },
            {

                "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_directory/C.1.0
→",

                "@type": "File",
                "contentSize": 20,
                "dateModified": "2022-10-24T08:52:02",
                "name": "C.1.0",
                "sdDatePublished": "2022-10-24T08:52:07+00:00"
            },
            {

                "@id": "file://bsccs742.int.bsc.es/Users/rsirvent/COMPSs-DP/matmul_directory/C.1.1
→",

                "@type": "File",
                "contentSize": 20,
                "dateModified": "2022-10-24T08:52:02",
                "name": "C.1.1",
                "sdDatePublished": "2022-10-24T08:52:07+00:00"
            }
        ]
}
```

6.4.7 ro-crate-metadata.json Java COMPSs example (MN4 supercomputer)

In this second `ro-crate-metadata.json` example, we want to illustrate the provenance result of a Java COMPSs application execution in the MareNostrum 4 supercomputer. We show the execution of a matrix LU factorization for out-of-core sparse matrices implemented with COMPSs and using the Java programming language. In this algorithm, matrix `A` is both input and output of the workflow, since the factorization overwrites the original value of `A`. In addition, we have used a 4x4 blocks hyper-matrix (i.e., the matrix is divided in 16 blocks, that contain 16 elements each) and, if a block is all 0s, the corresponding file will not be created in the file system (in the example, this happens for blocks `A.0.3`, `A.1.3`, `A.3.0` and `A.3.1`).

Apart from the terms already mentioned in the previous example (`creator`, `publisher`, `hasPart`, `ComputationalWorkflow`, `version`), if we first observe the `ro-crate-info.yaml` file:

```yaml
COMPSs Workflow Information:
  name: COMPSs Sparse LU
  description: The Sparse LU application computes an LU matrix factorization on a sparse␣
→blocked matrix. The matrix size (number of blocks) and the block size are parameters of the␣
→application.
  license: Apache-2.0 #Provide better a URL, but these strings are accepted:
                      # https://about.workflowhub.eu/Workflow-RO-Crate/#supported-licenses
  sources_dir: [src, jar, xml]
  files: [Readme, pom.xml, ro-crate-info.yaml]

Authors:
  - name: Raül Sirvent
    e-mail: Raul.Sirvent@bsc.es
    orcid: https://orcid.org/0000-0003-0606-2512
    organisation_name: Barcelona Supercomputing Center
    ror: https://ror.org/05sd8tv96
```

We can see that we have specified several directories to be added as source files: the `src` folder that contains the `.java` and `.class` files, the `jar` folder with the `sparseLU.jar` file, and the `xml` folder with extra xml configuration files. Besides, we also add the `Readme`, `pom.xml`, and the `ro-crate-info.yaml` file itself, so they are packed in the resulting crate. This example also shows that the script is able to select the correct `SparseLU.java` main file from the ones specified as `sources_dir`, although three files use the same file name (i.e., they implement 3 versions of the same algorithm: using files, arrays or objects).

It is also interesting to note the differences in the URIs used to reference input and output files when provenance is run in a supercomputer, instead of a laptop (as shown in the previous example). Since we do not add explicitly the input and output files of a workflow (because they could be extremely large), our crate only includes references to them, which are ment as pointers to where files can be found, rather than a publicly accessible URI reference. Therefore, while in the PyCOMPSs previous example files could be found in the `bsccs742.int.bsc.es` laptop, in this Java COMPSs example files can be found in `s02r2b45-ib0` hostname, which is an internal hostname of MN4. This means that, for reproducibility purposes, a new user would have to request input and output files to `bsccs742.int.bsc.es` laptop's owner in the first case, or request access to the MN4 paths specified by the corresponding URIs, in the second case.

```json
{
    "@context": "https://w3id.org/ro/crate/1.1/context",
    "@graph": [
        {
            "@id": "./",
            "@type": "Dataset",
            "creator": [
                {
                    "@id": "https://orcid.org/0000-0003-0606-2512"
                }
            ],
            "datePublished": "2022-10-24T09:31:39+00:00",
```

<div align="right">(continues on next page)</div>

```
            "description": "The Sparse LU application computes an LU matrix factorization on␣
↪a sparse blocked matrix. The matrix size (number of blocks) and the block size are␣
↪parameters of the application.",
            "hasPart": [
                {
                    "@id": "application_sources/src/main/java/sparseLU/files/Block.java"
                },
                {
                    "@id": "application_sources/src/main/java/sparseLU/files/SparseLUItf.class
↪"
                },
                {
                    "@id": "application_sources/src/main/java/sparseLU/files/SparseLUImpl.java
↪"
                },
                {
                    "@id": "application_sources/src/main/java/sparseLU/files/SparseLU.java"
                },
                {
                    "@id": "complete_graph.svg"
                },
                {
                    "@id": "App_Profile.json"
                },
                {
                    "@id": "compss_command_line_arguments.txt"
                },
                {
                    "@id": "application_sources/src/main/java/sparseLU/files/Block.class"
                },
                {
                    "@id": "application_sources/src/main/java/sparseLU/files/SparseLUItf.java"
                },
                {
                    "@id": "application_sources/src/main/java/sparseLU/files/SparseLUImpl.
↪class"
                },
                {
                    "@id": "application_sources/src/main/java/sparseLU/files/SparseLU.class"
                },
                {
                    "@id": "application_sources/src/main/java/sparseLU/objects/Block.java"
                },
                {
                    "@id": "application_sources/src/main/java/sparseLU/objects/SparseLUItf.
↪class"
                },
                {
                    "@id": "application_sources/src/main/java/sparseLU/objects/SparseLU.java"
                },
                {
                    "@id": "application_sources/src/main/java/sparseLU/objects/Block.class"
                },
                {
                    "@id": "application_sources/src/main/java/sparseLU/objects/SparseLUItf.
↪java"
```

```
                },
                {
                        "@id": "application_sources/src/main/java/sparseLU/objects/SparseLU.class"
                },
                {
                        "@id": "application_sources/src/main/java/sparseLU/arrays/SparseLUItf.
→class"
                },
                {
                        "@id": "application_sources/src/main/java/sparseLU/arrays/SparseLUImpl.
→java"
                },
                {
                        "@id": "application_sources/src/main/java/sparseLU/arrays/SparseLU.java"
                },
                {
                        "@id": "application_sources/src/main/java/sparseLU/arrays/SparseLUItf.java
→"
                },
                {
                        "@id": "application_sources/src/main/java/sparseLU/arrays/SparseLUImpl.
→class"
                },
                {
                        "@id": "application_sources/src/main/java/sparseLU/arrays/SparseLU.class"
                },
                {
                        "@id": "application_sources/jar/sparseLU.jar"
                },
                {
                        "@id": "application_sources/xml/resources.xml"
                },
                {
                        "@id": "application_sources/xml/project.xml"
                },
                {
                        "@id": "application_sources/Readme"
                },
                {
                        "@id": "application_sources/pom.xml"
                },
                {
                        "@id": "application_sources/ro-crate-info.yaml"
                },
                {
                        "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.0.0"
                },
                {
                        "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.0.1"
                },
                {
                        "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.0.2"
                },
```

```
                {
                    "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.1.0"
                },
                {
                    "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.1.1"
                },
                {
                    "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.1.2"
                },
                {
                    "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.2.0"
                },
                {
                    "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.2.1"
                },
                {
                    "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.2.2"
                },
                {
                    "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.2.3"
                },
                {
                    "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.3.2"
                },
                {
                    "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.3.3"
                }
            ],
            "license": "Apache-2.0",
            "mainEntity": {
                "@id": "application_sources/src/main/java/sparseLU/files/SparseLU.java"
            },
            "name": "COMPSs Sparse LU",
            "publisher": [
                {
                    "@id": "https://ror.org/05sd8tv96"
                }
            ]
        },
        {
            "@id": "ro-crate-metadata.json",
            "@type": "CreativeWork",
            "about": {
                "@id": "./"
            },
            "conformsTo": [
                {
```

```
                    "@id": "https://w3id.org/ro/crate/1.1"
                },
                {
                    "@id": "https://w3id.org/workflowhub/workflow-ro-crate/1.0"
                }
            ]
        },
        {
            "@id": "https://orcid.org/0000-0003-0606-2512",
            "@type": "Person",
            "affiliation": {
                "@id": "https://ror.org/05sd8tv96"
            },
            "contactPoint": {
                "@id": "mailto:Raul.Sirvent@bsc.es"
            },
            "name": "Ra\u00fcl Sirvent"
        },
        {
            "@id": "mailto:Raul.Sirvent@bsc.es",
            "@type": "ContactPoint",
            "contactType": "Author",
            "email": "Raul.Sirvent@bsc.es",
            "identifier": "Raul.Sirvent@bsc.es",
            "url": "https://orcid.org/0000-0003-0606-2512"
        },
        {
            "@id": "https://ror.org/05sd8tv96",
            "@type": "Organization",
            "name": "Barcelona Supercomputing Center"
        },
        {
            "@id": "application_sources/src/main/java/sparseLU/files/Block.java",
            "@type": "File",
            "contentSize": 5589,
            "description": "Auxiliary File",
            "encodingFormat": "text/plain",
            "name": "Block.java"
        },
        {
            "@id": "https://www.nationalarchives.gov.uk/PRONOM/x-fmt/415",
            "@type": "WebSite",
            "name": "Java Compiled Object Code"
        },
        {
            "@id": "application_sources/src/main/java/sparseLU/files/SparseLUItf.class",
            "@type": "File",
            "contentSize": 904,
            "description": "Auxiliary File",
            "encodingFormat": [
                [
                    "Java .class",
                    {
                        "@id": "https://www.nationalarchives.gov.uk/PRONOM/x-fmt/415"
                    }
                ]
```

```
            ],
            "name": "SparseLUItf.class"
        },
        {
            "@id": "application_sources/src/main/java/sparseLU/files/SparseLUImpl.java",
            "@type": "File",
            "contentSize": 2431,
            "description": "Auxiliary File",
            "encodingFormat": "text/plain",
            "name": "SparseLUImpl.java"
        },
        {
            "@id": "application_sources/src/main/java/sparseLU/files/SparseLU.java",
            "@type": [
                "File",
                "SoftwareSourceCode",
                "ComputationalWorkflow"
            ],
            "contentSize": 6602,
            "description": "Main file of the COMPSs workflow source files",
            "encodingFormat": "text/plain",
            "image": {
                "@id": "complete_graph.svg"
            },
            "input": [
                {
                    "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.0.0"
                },
                {
                    "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.0.1"
                },
                {
                    "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.0.2"
                },
                {
                    "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.1.0"
                },
                {
                    "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.1.1"
                },
                {
                    "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.1.2"
                },
                {
                    "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.2.0"
                },
                {
                    "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.2.1"
```

```
                        },
                        {
                                "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.2.2"
                        },
                        {
                                "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.2.3"
                        },
                        {
                                "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.3.2"
                        },
                        {
                                "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.3.3"
                        }
                ],
                "name": "SparseLU.java",
                "output": [
                        {
                                "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.0.0"
                        },
                        {
                                "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.0.1"
                        },
                        {
                                "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.0.2"
                        },
                        {
                                "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.1.0"
                        },
                        {
                                "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.1.1"
                        },
                        {
                                "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.1.2"
                        },
                        {
                                "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.2.0"
                        },
                        {
                                "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.2.1"
                        },
                        {
                                "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.2.2"
                        },
```

6.4. Data Provenance

```
                {
                    "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.2.3"
                },
                {
                    "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.3.2"
                },
                {
                    "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_
→apps/java/sparseLU/A.3.3"
                }
            ],
            "programmingLanguage": {
                "@id": "#compss"
            }
        },
        {
            "@id": "#compss",
            "@type": "ComputerLanguage",
            "alternateName": "COMPSs",
            "citation": "https://doi.org/10.1007/s10723-013-9272-5",
            "name": "COMPSs Programming Model",
            "url": "http://compss.bsc.es/",
            "version": "3.0.rc2210"
        },
        {
            "@id": "https://www.nationalarchives.gov.uk/PRONOM/fmt/92",
            "@type": "WebSite",
            "name": "Scalable Vector Graphics"
        },
        {
            "@id": "complete_graph.svg",
            "@type": [
                "File",
                "ImageObject",
                "WorkflowSketch"
            ],
            "about": {
                "@id": "application_sources/src/main/java/sparseLU/files/SparseLU.java"
            },
            "contentSize": 21106,
            "description": "The graph diagram of the workflow, automatically generated by␣
→COMPSs runtime",
            "encodingFormat": [
                [
                    "image/svg+xml",
                    {
                        "@id": "https://www.nationalarchives.gov.uk/PRONOM/fmt/92"
                    }
                ]
            ],
            "name": "complete_graph.svg"
        },
        {
            "@id": "https://www.nationalarchives.gov.uk/PRONOM/fmt/817",
```

```
            "@type": "WebSite",
            "name": "JSON Data Interchange Format"
        },
        {
            "@id": "App_Profile.json",
            "@type": "File",
            "contentSize": 1584,
            "description": "COMPSs application Tasks profile",
            "encodingFormat": [
                "application/json",
                {
                    "@id": "https://www.nationalarchives.gov.uk/PRONOM/fmt/817"
                }
            ],
            "name": "App_Profile.json"
        },
        {

            "@id": "compss_command_line_arguments.txt",
            "@type": "File",
            "contentSize": 28,
            "description": "COMPSs command line execution command, including parameters passed
↪",
            "encodingFormat": "text/plain",
            "name": "compss_command_line_arguments.txt"
        },
        {

            "@id": "application_sources/src/main/java/sparseLU/files/Block.class",
            "@type": "File",
            "contentSize": 4135,
            "description": "Auxiliary File",
            "encodingFormat": [
                [
                    "Java .class",
                    {
                        "@id": "https://www.nationalarchives.gov.uk/PRONOM/x-fmt/415"
                    }
                ]
            ],
            "name": "Block.class"
        },
        {

            "@id": "application_sources/src/main/java/sparseLU/files/SparseLUItf.java",
            "@type": "File",
            "contentSize": 1808,
            "description": "Auxiliary File",
            "encodingFormat": "text/plain",
            "name": "SparseLUItf.java"
        },
        {

            "@id": "application_sources/src/main/java/sparseLU/files/SparseLUImpl.class",
            "@type": "File",
            "contentSize": 1310,
            "description": "Auxiliary File",
            "encodingFormat": [
                [
                    "Java .class",
```

```
                {
                    "@id": "https://www.nationalarchives.gov.uk/PRONOM/x-fmt/415"
                }
            ]
        ],
        "name": "SparseLUImpl.class"
    },
    {
        "@id": "application_sources/src/main/java/sparseLU/files/SparseLU.class",
        "@type": "File",
        "contentSize": 4682,
        "description": "Auxiliary File",
        "encodingFormat": [
            [
                "Java .class",
                {
                    "@id": "https://www.nationalarchives.gov.uk/PRONOM/x-fmt/415"
                }
            ]
        ],
        "name": "SparseLU.class"
    },
    {
        "@id": "application_sources/src/main/java/sparseLU/objects/Block.java",
        "@type": "File",
        "contentSize": 4345,
        "description": "Auxiliary File",
        "encodingFormat": "text/plain",
        "name": "Block.java"
    },
    {
        "@id": "application_sources/src/main/java/sparseLU/objects/SparseLUItf.class",
        "@type": "File",
        "contentSize": 816,
        "description": "Auxiliary File",
        "encodingFormat": [
            [
                "Java .class",
                {
                    "@id": "https://www.nationalarchives.gov.uk/PRONOM/x-fmt/415"
                }
            ]
        ],
        "name": "SparseLUItf.class"
    },
    {
        "@id": "application_sources/src/main/java/sparseLU/objects/SparseLU.java",
        "@type": "File",
        "contentSize": 4740,
        "description": "Auxiliary File",
        "encodingFormat": "text/plain",
        "name": "SparseLU.java"
    },
    {
        "@id": "application_sources/src/main/java/sparseLU/objects/Block.class",
        "@type": "File",
```

```
            "contentSize": 2991,
            "description": "Auxiliary File",
            "encodingFormat": [
                [
                    "Java .class",
                    {
                        "@id": "https://www.nationalarchives.gov.uk/PRONOM/x-fmt/415"
                    }
                ]
            ],
            "name": "Block.class"
        },
        {
            "@id": "application_sources/src/main/java/sparseLU/objects/SparseLUItf.java",
            "@type": "File",
            "contentSize": 1529,
            "description": "Auxiliary File",
            "encodingFormat": "text/plain",
            "name": "SparseLUItf.java"
        },
        {
            "@id": "application_sources/src/main/java/sparseLU/objects/SparseLU.class",
            "@type": "File",
            "contentSize": 3403,
            "description": "Auxiliary File",
            "encodingFormat": [
                [
                    "Java .class",
                    {
                        "@id": "https://www.nationalarchives.gov.uk/PRONOM/x-fmt/415"
                    }
                ]
            ],
            "name": "SparseLU.class"
        },
        {
            "@id": "application_sources/src/main/java/sparseLU/arrays/SparseLUItf.class",
            "@type": "File",
            "contentSize": 808,
            "description": "Auxiliary File",
            "encodingFormat": [
                [
                    "Java .class",
                    {
                        "@id": "https://www.nationalarchives.gov.uk/PRONOM/x-fmt/415"
                    }
                ]
            ],
            "name": "SparseLUItf.class"
        },
        {
            "@id": "application_sources/src/main/java/sparseLU/arrays/SparseLUImpl.java",
            "@type": "File",
            "contentSize": 4114,
            "description": "Auxiliary File",
            "encodingFormat": "text/plain",
```

6.4. Data Provenance

```
                    "name": "SparseLUImpl.java"
                },
                {
                    "@id": "application_sources/src/main/java/sparseLU/arrays/SparseLU.java",
                    "@type": "File",
                    "contentSize": 4840,
                    "description": "Auxiliary File",
                    "encodingFormat": "text/plain",
                    "name": "SparseLU.java"
                },
                {
                    "@id": "application_sources/src/main/java/sparseLU/arrays/SparseLUItf.java",
                    "@type": "File",
                    "contentSize": 1899,
                    "description": "Auxiliary File",
                    "encodingFormat": "text/plain",
                    "name": "SparseLUItf.java"
                },
                {
                    "@id": "application_sources/src/main/java/sparseLU/arrays/SparseLUImpl.class",
                    "@type": "File",
                    "contentSize": 2430,
                    "description": "Auxiliary File",
                    "encodingFormat": [
                        [
                            "Java .class",
                            {
                                "@id": "https://www.nationalarchives.gov.uk/PRONOM/x-fmt/415"
                            }
                        ]
                    ],
                    "name": "SparseLUImpl.class"
                },
                {
                    "@id": "application_sources/src/main/java/sparseLU/arrays/SparseLU.class",
                    "@type": "File",
                    "contentSize": 3304,
                    "description": "Auxiliary File",
                    "encodingFormat": [
                        [
                            "Java .class",
                            {
                                "@id": "https://www.nationalarchives.gov.uk/PRONOM/x-fmt/415"
                            }
                        ]
                    ],
                    "name": "SparseLU.class"
                },
                {
                    "@id": "https://www.nationalarchives.gov.uk/PRONOM/x-fmt/412",
                    "@type": "WebSite",
                    "name": "Java Archive Format"
                },
                {
                    "@id": "application_sources/jar/sparseLU.jar",
                    "@type": "File",
```

```
            "contentSize": 28758,
            "description": "Auxiliary File",
            "encodingFormat": [
                [
                    "application/java-archive",
                    {
                        "@id": "https://www.nationalarchives.gov.uk/PRONOM/x-fmt/412"
                    }
                ]
            ],
            "name": "sparseLU.jar"
        },
        {

            "@id": "application_sources/xml/resources.xml",
            "@type": "File",
            "contentSize": 983,
            "description": "Auxiliary File",
            "name": "resources.xml"
        },
        {

            "@id": "application_sources/xml/project.xml",
            "@type": "File",
            "contentSize": 289,
            "description": "Auxiliary File",
            "name": "project.xml"
        },
        {

            "@id": "application_sources/Readme",
            "@type": "File",
            "contentSize": 1935,
            "description": "Auxiliary File",
            "name": "Readme"
        },
        {

            "@id": "application_sources/pom.xml",
            "@type": "File",
            "contentSize": 4454,
            "description": "Auxiliary File",
            "name": "pom.xml"
        },
        {

            "@id": "application_sources/ro-crate-info.yaml",
            "@type": "File",
            "contentSize": 699,
            "description": "Auxiliary File",
            "name": "ro-crate-info.yaml"
        },
        {

            "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_apps/java/
→sparseLU/A.0.0",
            "@type": "File",
            "contentSize": 304,
            "dateModified": "2022-10-24T09:31:37",
            "name": "A.0.0",
            "sdDatePublished": "2022-10-24T09:31:39+00:00"
        },
```

6.4. Data Provenance 257

```
        {
            "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_apps/java/
→sparseLU/A.0.1",
            "@type": "File",
            "contentSize": 303,
            "dateModified": "2022-10-24T09:31:37",
            "name": "A.0.1",
            "sdDatePublished": "2022-10-24T09:31:39+00:00"
        },
        {
            "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_apps/java/
→sparseLU/A.0.2",
            "@type": "File",
            "contentSize": 306,
            "dateModified": "2022-10-24T09:31:37",
            "name": "A.0.2",
            "sdDatePublished": "2022-10-24T09:31:39+00:00"
        },
        {
            "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_apps/java/
→sparseLU/A.1.0",
            "@type": "File",
            "contentSize": 311,
            "dateModified": "2022-10-24T09:31:37",
            "name": "A.1.0",
            "sdDatePublished": "2022-10-24T09:31:39+00:00"
        },
        {
            "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_apps/java/
→sparseLU/A.1.1",
            "@type": "File",
            "contentSize": 320,
            "dateModified": "2022-10-24T09:31:37",
            "name": "A.1.1",
            "sdDatePublished": "2022-10-24T09:31:39+00:00"
        },
        {
            "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_apps/java/
→sparseLU/A.1.2",
            "@type": "File",
            "contentSize": 312,
            "dateModified": "2022-10-24T09:31:37",
            "name": "A.1.2",
            "sdDatePublished": "2022-10-24T09:31:39+00:00"
        },
        {
            "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_apps/java/
→sparseLU/A.2.0",
            "@type": "File",
            "contentSize": 319,
            "dateModified": "2022-10-24T09:31:37",
            "name": "A.2.0",
            "sdDatePublished": "2022-10-24T09:31:39+00:00"
        },
        {
            "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_apps/java/
→sparseLU/A.2.1",
```

```
            "@type": "File",
            "contentSize": 323,
            "dateModified": "2022-10-24T09:31:37",
            "name": "A.2.1",
            "sdDatePublished": "2022-10-24T09:31:39+00:00"
        },
        {
            "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_apps/java/
→sparseLU/A.2.2",
            "@type": "File",
            "contentSize": 311,
            "dateModified": "2022-10-24T09:31:37",
            "name": "A.2.2",
            "sdDatePublished": "2022-10-24T09:31:39+00:00"
        },
        {
            "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_apps/java/
→sparseLU/A.2.3",
            "@type": "File",
            "contentSize": 303,
            "dateModified": "2022-10-24T09:31:37",
            "name": "A.2.3",
            "sdDatePublished": "2022-10-24T09:31:39+00:00"
        },
        {
            "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_apps/java/
→sparseLU/A.3.2",
            "@type": "File",
            "contentSize": 320,
            "dateModified": "2022-10-24T09:31:37",
            "name": "A.3.2",
            "sdDatePublished": "2022-10-24T09:31:39+00:00"
        },
        {
            "@id": "file://s02r2b45-ib0/gpfs/home/bsc19/bsc19057/COMPSs-DP/tutorial_apps/java/
→sparseLU/A.3.3",
            "@type": "File",
            "contentSize": 310,
            "dateModified": "2022-10-24T09:31:37",
            "name": "A.3.3",
            "sdDatePublished": "2022-10-24T09:31:39+00:00"
        }
    ]
}
```

Chapter 7

Persistent Storage

COMPSs is able to interact with Persistent Storage frameworks. To this end, it is necessary to take some considerations in the application code and on its execution. This section is intended to walk you through the COMPSs' storage interface and its integration with some Persistent Storage frameworks.

7.1 First steps

COMPSs relies on a Storage API to enable the interation with persistent storage frameworks (Figure 43), which is composed by two main modules: *Storage Object Interface* (SOI) and *Storage Runtime Interface* (SRI)
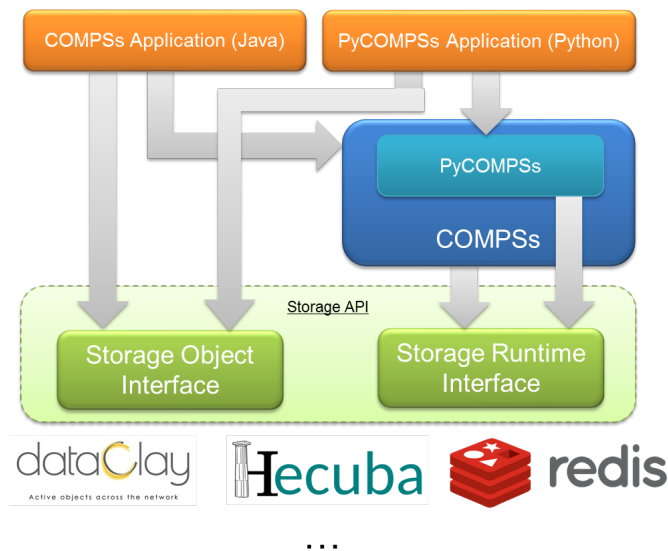
Figure 43: COMPSs with persistent storage architecture

Any COMPSs application aimed at using a persistent storage framework has to include calls to:

- The SOI in order to define the data model (see *Defining the data model*), and relies on COMPSs, which interacts with the persistent storage framework through the SRI.
- The SRI in order to interact directly with the storage backend (e.g. retrieve data, etc.) (see *Interacting with the persistent storage*).

In addition, it must be taken into account that the execution of an application using a persistent storage framework requires some specific flags in `runcompss` and `enqueue_compss` (see *Running with persistent storage*).

Currently, there exists storage interfaces for dataClay, Hecuba and Redis. They are thoroughly described from the developer and user point of view in Sections:

- *COMPSs + dataClay*
- *COMPSs + Hecuba*
- *COMPSs + Redis*

The interface is open to any other storage framework by implementing the required functionalities described in *Implement your own Storage interface for COMPSs*.

7.1.1 Defining the data model

The data model consists of a set of related classes programmed in one of the supported languages aimed are representing the objects used in the application (e.g. in a wordcount application, the data model would be text).

In order to define that the application objects are going to be stored in the underlying persistent storage backend, the data model must be enriched with the *Storage Object Interface* (SOI).

The SOI provides a set of functionalities that all objects stored in the persistent storage backend will need. Consequently, the user must inherit the SOI on its data model classes, and give some insights of the class attributes.

The following subsections detail how to enrich the data model in Java and Python applications.

7.1.1.1 Java

To define that a class objects are going to be stored in the persistent storage backend, the class must extend the `StorageObject` class (as well as implement the `Serializable` interface). This class is provided by the persistent storage backend.

```java
import storage.StorageObject;
import java.io.Serializable;

class MyClass extends StorageObject implements Serializable {

    private double[] vector;

    /**
     * Write here your class-specific
     * constructors, attributes and methods.
     */
}
```

The `StorageObject` object enriches the class with some methods that allow the user to interact with the persistent storage backend. These methods can be found in Table 21.

Table 21: Available methods from StorageObject

Name	Returns	Comments
makePersistent(String id)	Nothing	Inserts the object in the database with the id. If id is null, a random UUID will be computed instead.
deletePersistent()	Nothing	Removes the object from the storage. It does nothing if it was not already there.
getID()	String	Returns the current object identifier if the object is not persistent (null instead).

These functions can be used from the application in order to persist an object (pushing the object into the persistent storage) with `make_persistent`, remove it from the persistent storage with `delete_persistent` or getting the object identifier with `getID` for the later interaction with the storage backend.

```java
import MyPackage.MyClass;

class Test{
    // ...
    public static void main(String args[]){
        // ...
        MyClass my_obj = new MyClass();
        my_obj.matrix = new double[10];
        my_obj.makePersistent();            // make persistent without parameter
        String obj_id = my_obj.getID();     // get the idenfier provided by the storage framework
        // ...
        my_obj.deletePersistent();
        // ...
        MyClass my_obj2 = new MyClass();
        my_obj2.matrix = new double[20];
        my_obj2.makePersistent("obj2");     // make persistent providing identifier
        // ...
        my_obj2.delete_persistent();
        // ...
    }
}
```

7.1.1.2 Python

To define that a class objects are going to be stored in the persistent storage backend, the class must inherit the `StorageObject` class. This class is provided by the persistent storage backend.

```python
from storage.api import StorageObject

class MyClass(StorageObject):
    ...
```

In addition, the user has to give details about the class attributes using the class documentation. For example, if the user wants to define a class containing a numpy ndarray as attribute, the user has to specify this attribute starting with `@ClassField` followed by the attribute name and type:

```python
from storage.api import StorageObject

class MyClass(StorageObject):
    """
    @ClassField matrix numpy.ndarray
    """
    pass
```

Important: Methods inside the class are not supported by all storage backends. dataClay is currently the only backend that provides support for them (see *Enabling COMPSs applications with dataClay*).

Then, the user can use the instantiated object normally:

```python
from MyFile import MyClass
import numpy as np

my_obj = MyClass()
my_obj.matrix = np.random.rand(10, 2)
...
```

The following code snippet gives some examples of several types of attributes:

```python
from storage.api import StorageObject

class MyClass(StorageObject):
    """
    # Elemmental types
    @ClassField field1 int
    @ClassField field2 str
    @ClassField field3 np.ndarray

    # Structured types
    @ClassField field4 list <int>
    @ClassField field5 set <list<float>>

    # Another class instance as attribute
    @ClassField field6 AnotherClassName

    # Complex dictionaries:
    @ClassField field7 dict <<int,str>, dict<<int>, list<str>>>
    @ClassField field8 dict <<int>, AnotherClassName>

    # Dictionary with structured value:
```

(continues on next page)

```
    @ClassField field9 dict <<k1: int, k2: int>, tuple<v1: int, v2: float, v3: text>>
    # Plain definition of the same dictionary:
    @ClassField field10 dict <<int,int>, str>
    """
    pass
```

Finally, the `StorageObject` class includes some functions in the class that will be available from the instantiated objects (Table 22).

Table 22: Available methods from StorageObject in Python

Name	Returns	Comments
make_persistent(String id)	Nothing	Inserts the object in the database with the id. If id is null, a random UUID will be computed instead.
delete_persistent()	Nothing	Removes the object from the storage. It does nothing if it was not already there.
getID()	String	Returns the current object identifier if the object is not persistent (`None` instead).

These functions can be used from the application in order to persist an object (pushing the object into the persistent storage) with `make_persistent`, remove it from the persistent storage with `delete_persistent` or getting the object identifier with `getID` for the later interaction with the storage backend.

```python
import numpy as np

my_obj = MyClass()
my_obj.matrix = np.random.rand(10, 2)
my_obj.make_persistent()   # make persistent without parameter
obj_id = my_obj.getID()    # get the idenfier provided by the storage framework
...
my_obj.delete_persistent()
...
my_obj2 = MyClass()
my_obj2.matrix = np.random.rand(10, 3)
my_obj2.make_persistent('obj2')   # make persistent providing identifier
...
my_obj2.delete_persistent()
...
```

7.1.1.3 C/C++

Unsupported

Persistent storage is not supported with C/C++ COMPSs applications.

7.1.2 Interacting with the persistent storage

The **Storage Runtime Interface** (SRI) provides some functions to interact with the storage backend. All of them are aimed at enabling the COMPSs runtime to deal with persistent data across the infrastructure.

However, the function to retrieve an object from the storage backend from its identifier can be useful for the user. Consequently, users can import the SRI and use the `getByID` function when needed necessary. This function requires a String parameter with the object identifier, and returns the object associated with that identifier (`null` or `None` otherwise).

The following subsections detail how to call the `getByID` function in Java and Python applications.

7.1.2.1 Java

Import the `getByID` function from the storage api and use it:

```java
import storage.StorageItf;
import MyPackage.MyClass;

class Test{
    // ...
    public static void main(String args[]){
        // ...
        obj = StorageItf.getByID("my_obj");
        // ...
    }
}
```

7.1.2.2 Python

Import the `getByID` function from the storage api and use it:

```python
from storage.api import getByID

..
obj = getByID('my_obj')
...
```

7.1.2.3 C/C++

Unsupported

Persistent storage is not supported with C/C++ COMPSs applications.

7.1.3 Running with persistent storage

7.1.3.1 Local

In order to run a COMPSs application locally, the `runcompss` command is used.

The `runcompss` command includes some flags to execute the application considering a running persistent storage framework. These flags are: `--classpath`, `--pythonpath` and `--storage_conf`.

Consequently, the `runcompss` requirements to run an application with a running persistent storage backend are:

--classpath	Add the `--classpath=${path_to_storage_api.jar}` flag to the `runcompss` command.
--pythonpath	If you are running a python application, also add the `--pythonpath=${path_-to_the_storage_api}/python` flag to the `runcompss` command.
--storage_conf	Add the flag `--storage_conf=${path_to_your_storage_conf_dot_cfg_file}` to the `runcompss` command. The storage configuration file (usually `storage_-conf.cfg`) contains the configuration parameters needed by the storage framework for the execution (it depends on the storage framework).

As usual, the `project.xml` and `resources.xml` files must be correctly set.

7.1.3.2 Supercomputer

In order to run a COMPSs application in a Supercomputer or cluster, the `enqueue_compss` command is used.

The `enqueue_compss` command includes some flags to execute the application considering a running persistent storage framework. These flags are: `--classpath`, `--pythonpath`, `--storage-home` and `--storage-props`.

Consequently, the `enqueue_compss` requirements to run an application with a running persistent storage backend are:

--classpath	`--classpath=${path_to_storage_interface.jar}` As with the `runcompss` command, the JAR with the storage API must be specified. It is usally available in a environment variable (check the persistent storage framework).
--pythonpath	If you are running a Python application, also add the `--pythonpath=${path_-to_the_storage_api}/python` flag. It is usally available in a environment variable (check the persistent storage framework).
--storage-home	`--storage-home=${path_to_the_storage_api}` This must point to the root of the storage folder. This folder must contain a `scripts` folder where the scripts to start and stop the persistent framework are. It is usally available in a environment variable (check the persistent storage framework).
--storage-props	`--storage-props=${path_to_the_storage_props_file}` This must point to the storage properties configuration file (usually `storage_props.cfg`) It contains the configuration parameters needed by the storage framework for the execution (it depends on the storage framework).

7.2 COMPSs + dataClay

> **Warning:** Under construction

7.2.1 COMPSs + dataClay Dependencies

7.2.1.1 dataClay

7.2.1.2 Other dependencies

7.2.2 Enabling COMPSs applications with dataClay

7.2.2.1 Java

7.2.2.2 Python

7.2.2.3 C/C++

Unsupported

C/C++ COMPSs applications are not supported with dataClay.

7.2.3 Executing a COMPSs application with dataClay

7.2.3.1 Launching using an existing dataClay deployment

7.2.3.2 Launching on queue system based environments

7.3 COMPSs + Hecuba

Hecuba is a set of tools and interfaces that implement a simple and efficient access to data stores for big data applications. The current implementation of Hecuba supports Python applications that store data in memory or Apache Cassandra databases.

The Hecuba manual is available in its Github Wiki.

Hecuba is developed by a team composed of BSC (Data-driven Scientific Computing research line) and UPC staff.

7.3.1 COMPSs + Hecuba Dependencies

The required dependency is Hecuba.

Download the Hecuba source code from the following repository: https://github.com/bsc-dd/hecuba.

And follow the instructions for the Hecuba Installation Procedure.

7.3.2 Enabling COMPSs applications with Hecuba

7.3.2.1 Java

Unsupported

Java COMPSs applications are not supported with Hecuba.

7.3.2.2 Python

PyCOMPSs allow programmers to write sequential code and to indicate, through a decorator, which functions can be executed in parallel. The COMPSs runtime interprets this decorator and executes, transparent to the programmer, all the code necessary to schedule each task on a computing node, to manage dependencies between tasks and to send and to serialize the parameters and the returns of the tasks.

When input/output parameters of a tasks are persistent objects (i.e. their classes implement the Storage API defined to interact with PyCOMPSs), the runtime asks the storage system for the data locality information and uses this information to try to schedule the task on the node containing the data. This way no data sending or serialization is needed.

The following code shows an example of PyCOMPSs task. The input parameter of the task could be an object resulting from splitting a StorageDict (`partition` can be an object intance of `MyClass` that can be persistent). In this example the return of the task is a Python dictionary.

```python
from pycompss.api.task import task
from hecuba import StorageDict

class MyClass(StorageDict):
    '''
    @TypeSpec dict<<str>, int>
    '''

@task(returns = dict)
def wordcountTask(partition):
    partialResult = {}
    for word in partition.values():
        if word not in partialResult:
            partialResult[word] = 1
        else:
            partialResult[word] = partialResult[word] + 1
    return partialResult
```
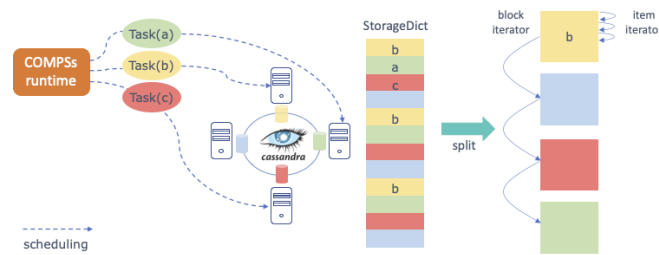
7.3.2.3 C/C++

Unsupported

C/C++ COMPSs applications are not supported with Hecuba.

7.3.3 Executing a COMPSs application with Hecuba

7.3.3.1 Launching using an existing Hecuba deployment

If Hecuba is already running on the node/s where the COMPSs application will run then only the following steps must be followed:

1. Create a `storage_conf.cfg` file that lists, one per line, the nodes where the storage is present. Only hostnames or IPs are needed, ports are not necessary here.
2. Add the flag `--classpath=${path_to_Hecuba.jar}` to the `runcompss` command that launches the application.
3. Add the flag `--storage_conf=${path_to_your_storage_conf_dot_cfg_file}` to the `runcompss` command that launches the application.
4. If you are running a python app, also add the `--pythonpath=${app_path}:${path_to_the_bundle_-folder}/python` flag to the `runcompss` command that launches the application.

As usual, the `project.xml` and `resources.xml` files must be correctly set. It must be noted that there can be Hecuba nodes that are not COMPSs nodes.

7.3.3.2 Launching on queue system based environments

To run a parallel Hecuba application using PyCOMPSs you should execute the `enqueue_compss` command setting the options `--storage_props` and `--storage_home`.

The `--storage_props` option is mandatory and should contain the path of an existing file. This file can contain all the Hecuba configuration options that the user needs to set (can be an empty file). The `--storage_home` option contains the path to the Hecuba implementation of the Storage API required by COMPSs.

Following, we show an example of how to use PyCOMPSs and Hecuba to run the python application in the file myapp.py.

```
compss job submit \
  --num_nodes=4 \
  --storage_props=storage_props.cfg \
  --storage_home=$HECUBA_ROOT/compss/ \
  --scheduler=es.bsc.compss.scheduler.lookahead.locality.LocalityTS \
  --lang=python \
  $(pwd)/myapp.py
```

In this example, we ask PyCOMPSs to allocate 4 nodes and to use the scheduler that enhances data locality for tasks using persistent objects. We assume that the variable `HECUBA_ROOT` contains the path to the installation directory of Hecuba.

Hecuba Configuration Parameters There are several parameters that can be defined when running our application. The basic parameters are the following:

CONTACT_NAMES (default value: 'localhost') list of the Storage System nodes separated by a comma (example: `export CONTACT_NAMES=node1,node2,node3`)

NODE_PORT (default value: 9042) Storage System listening port

EXECUTION_NAME (default value: 'my_app') Default name for the upper level in the app namespace hierarchy

CREATE_SCHEMA (default value: False) If set to True, Hecuba will create its metadata structures into the storage system. Notice that these metadata structures are kept from one execution to another so it is only necessary to create them if you have deployed from scratch the storage system.

Hecuba Advanced Configuration Parameters There are several parameters that can be defined for Hecuba configuration:

NUMBER_OF_BLOCKS (default value: 1024) Number of partitions in which the data will be divided for each node

CONCURRENT_CREATION (default value: False) You should set it to True if you need to support concurrent persistent object creation. Setting this variable slows-down the creation task so you should keep it to False if only sequential creation is used or if the concurrent creation involves disjoint objects

LOAD_ON_DEMAND (default value: True) If set to True data is retrieved only when it is accessed. If it is set to False data is loaded when an instance to the object is created. It is necessary to set to True if you code uses those functions of the numpy library that do not use the interface to access the elements of the numpy ndarray.

DEBUG (default value: False) If set to True Hecuba shows during the execution of the application some output messages describing the steps performed

SPLITS_PER_NODE (default value: 32) Number of partitions that generates the split method

MAX_CACHE_SIZE (default value: 1000) Size of the cache. You should set it to 0 (and thus deactivate the utilization of the cache) if the persistent objects are small enough to keep them in memory while they are in used

PREFETCH_SIZE (default value: 10000) Number of elements read in advance when iterating on a persistent object

WRITE_BUFFER_SIZE (default value: 1000) Size of the internal buffer used to group insertions to reduce the number of interactions with the storage system

WRITE_CALLBACKS_NUMBER (default value: 16) Number of concurrent on-the-fly insertions that Hecuba can support

REPLICATION_STRATEGY (default value: 'SimpleStrategy') Strategy to follow in the Cassandra database

REPLICA_FACTOR (default value: 1) The amount of replicas of each data available in the Cassandra cluster

Hecuba Specific Configuration Parameters for the `storage_props` file There are several parameters that can be defined for the `storage_props` file for PyCOMPSs:

CONTACT_NAMES (default value: empty) If this variable is set in the `storage_props` file, then COMPSs assumes that the variable contains the list of of an already running Cassandra cluster. If this variable is not set in the `storage_props` file, then the `enqueue_compss` command will use the Hecuba scripts to deploy and launch a new Cassandra cluster using all the nodes assigned to workers.

RECOVER (default value: empty) If this variable is set in the storage_props file, then the `enqueue_compss` command will use the Hecuba scripts to deploy and launch a new Cassandra cluster starting from the snapshot identified by the variable. Notice that in this case, the number of nodes used to generate the snapshot should match the number of workers requested by the `enqueue_compss` command.

MAKE_SNAPSHOT (default value: 0) The user should set this variable to 1 in the storage_props file if a snapshot of the database should be generated and stored once the application ends the execution (this feature is still under development, users can currently generate snapshots of the database using the `c4s` tool provided as part of Hecuba).

7.4 COMPSs + Redis

COMPSs provides a built-in interface to use Redis as persistent storage from COMPSs' applications.

Note: We assume that COMPSs is already installed. See *Installation and Administration*

The next subsections focus on how to install the Redis utilities and the storage API for COMPSs.

Hint: It is advisable to read the Redis Cluster tutorial for beginners[1] in order to understand all the terminology that is used.

7.4.1 COMPSs + Redis Dependencies

The required dependencies are:

- *Redis Server*
- *Redis Cluster script*
- *COMPSs-Redis Bundle*

7.4.1.1 Redis Server

`redis-server` is the core Redis program. It allows to create standalone Redis instances that may form part of a cluster in the future. `redis-server` can be obtained by following these steps:

1. Go to `https://redis.io/download` and download the last stable version. This should download a `redis-${version}.tar.gz` file to your computer, where `${version}` is the current latest version.
2. Unpack the compressed file to some directory, open a terminal on it and then type `sudo make install` if you want to install Redis for all users. If you want to have it installed only for yourself you can simply type `make redis-server`. This will leave the `redis-server` executable file inside the directory `src`, allowing you to move it to a more convenient place. By *convenient place* we mean a folder that is in your `PATH` environment variable. It is advisable to not delete the uncompressed folder yet.
3. If you want to be sure that Redis will work well on your machine then you can type `make test`. This will run a very exhaustive test suite on Redis features.

Important: Do not delete the uncompressed folder yet.

7.4.1.2 Redis Cluster script

Redis needs an additional script to form a cluster from various Redis instances. This script is called `redis-trib.rb` and can be found in the same tar.gz file that contains the sources to compile `redis-server` in `src/redis-trib.rb`. Two things must be done to make this script work:

1. Move it to a convenient folder. By *convenient folder* we mean a folder that is in your `PATH` environment variable.
2. Make sure that you have Ruby and `gem` installed. Type `gem install redis`.
3. In order to use COMPSs + Redis with Python you must also install the `redis` and `redis-py-cluster` PyPI packages.

   Hint: It is also advisable to have the PyPI package `hiredis`, which is a library that makes the interactions with the storage to go faster.

---

[1] https://redis.io/topics/cluster-tutorial

7.4.1.3 COMPSs-Redis Bundle

`COMPSs-Redis Bundle` is a software package that contains the following:

1. A java JAR file named `compss-redisPSCO.jar`. This JAR contains the implementation of a Storage Object that interacts with a given Redis backend. We will discuss the details later.
2. A folder named `scripts`. This folder contains a bunch of scripts that allows a COMPSs-Redis app to create a custom, in-place cluster for the application.
3. A folder named `python` that contains the Python equivalent to `compss-redisPSCO.jar`

This package can be obtained from the COMPSs source as follows:

1. Go to `trunk/utils/storage/redisPSCO`
2. Type `./make_bundle`. This will leave a folder named `COMPSs-Redis-bundle` with all the bundle contents.

7.4.2 Enabling COMPSs applications with Redis

7.4.2.1 Java

This section describes how to develop Java applications with the Redis storage. The application project should have the dependency induced by `compss-redisPSCO.jar` satisfied. That is, it should be included in the application's `pom.xml` if you are using Maven, or it should be listed in the dependencies section of the used development tool.

The application is almost identical to a regular COMPSs application except for the presence of Storage Objects. A Storage Object is an object that it is capable to interact with the storage backend. If a custom object extends the Redis Storage Object and implements the Serializable interface then it will be ready to be stored and retrieved from a Redis database. An example signature could be the following:

```java
import storage.StorageObject;
import java.io.Serializable;

  /**
  * A PSCO that contains a KD point
  */
  class RedisPoint
  extends StorageObject implements Serializable {

    // Coordinates of our point
    private double[] coordinates;
    /**
    * Write here your class-specific
    * constructors, attributes and methods.
    */
    double getManhattanDistance(RedisPoint other) {
      ...
    }
  }
```

The `StorageObject` object has some inherited methods that allow the user to write custom objects that interact with the Redis backend. These methods can be found in Table 23.

Table 23: Available methods from StorageObject

Name	Returns	Comments
makePersistent(String id)	Nothing	Inserts the object in the database with the id. If id is null, a random UUID will be computed instead.
deletePersistent()	Nothing	Removes the object from the storage. It does nothing if it was not already there.
getID()	String	Returns the current object identifier if the object is not persistent (null instead).

Caution: Redis Storage Objects that are used as INOUTs must be manually updated. This is due to the fact that COMPSs does not know the exact effects of the interaction between the object and the storage, so the runtime cannot know if it is necessary to call `makePersistent` after having used an INOUT or not (other storage approaches do live modifications to its storage objects). The followingexample illustrates this situation:

```
/**
* A is passed as INOUT
*/
void accumulativePointSum(RedisPoint a, RedisPoint b) {
  // This method computes the coordinate-wise sum between a and b
  // and leaves the result in a
  for(int i=0; i<a.getCoordinates().length; ++i) {
    a.setComponent(i, a.getComponent(i) + b.getComponent(i));
  }
  // Delete the object from the storage and
  // re-insert the object with the same old identifier
  String objectIdentifier = a.getID();
  // Redis contains the old version of the object
  a.deletePersistent();
  // Now we will insert the updated one
  a.makePersistent(objectIdentifier);
}
```

If the last three statements were not present, the changes would never be reflected on the `RedisPoint a` object.

7.4.2.2 Python

Redis is also available for Python. As happens with Java, we first need to define a custom Storage Object. Let's suppose that we want to write an application that multiplies two matrices A, and B by blocks. We can define a `Block` object that lets us store and write matrix blocks in our Redis backend:

```python
from storage.storage_object import StorageObject
import storage.api

class Block(StorageObject):
  def __init__(self, block):
      super(Block, self).__init__()
      self.block = block

  def get_block(self):
      return self.block

  def set_block(self, new_block):
      self.block = new_block
```

Let's suppose that we are multiplying our matrices in the usual blocked way:

```python
for i in range(MSIZE):
  for j in range(MSIZE):
    for k in range(MSIZE):
      multiply(A[i][k], B[k][j], C[i][j])
```

Where A and B are `Block` objects and C is a regular Python object (e.g: a Numpy matrix), then we can define `multiply` as a task as follows:

```python
@task(c = INOUT)
def multiply(a_object, b_object, c, MKLProc):
    c += a_object.block * b_object.block
```

Let's also suppose that we are interested to store the final result in our storage. A possible solution is the following:

```python
for i in range(MSIZE):
    for j in range(MSIZE):
        persist_result(C[i][j])
```

Where `persist_result` can be defined as a task as follows:

```python
@task()
def persist_result(obj):
    to_persist = Block(obj)
    to_persist.make_persistent()
```

This way is preferred for two main reasons:

- we avoid to bring the resulting matrix to the master node,
- and we can exploit the data locality by executing the task in the node where last version of `obj` is located.

7.4.2.3 C/C++

Unsupported

C/C++ COMPSs applications are not supported with Redis.

7.4.3 Executing a COMPSs application with Redis

7.4.3.1 Launching using an existing Redis Cluster

If there is already a running Redis Cluster on the node/s where the COMPSs application will run then only the following steps must be followed:

1. Create a `storage_conf.cfg` file that lists, one per line, the nodes where the storage is present. Only hostnames or IPs are needed, ports are not necessary here.
2. Add the flag `--classpath=${path_to_COMPSs-redisPSCO.jar}` to the `runcompss` command that launches the application.
3. Add the flag `--storage_conf=${path_to_your_storage_conf_dot_cfg_file}` to the `runcompss` command that launches the application.
4. If you are running a python app, also add the `--pythonpath=${app_path}:${path_to_the_bundle_-folder}/python` flag to the `runcompss` command that launches the application.

As usual, the `project.xml` and `resources.xml` files must be correctly set. It must be noted that there can be Redis nodes that are not COMPSs nodes (although **this is a highly unrecommended practice**). As a requirement, **there must be at least one Redis instance on each COMPSs node listening to the official Redis port 6379**[2]. This is required because nodes without running Redis instances would cause a great amount of transfers (they will **always** need data that must be transferred from another node). Also, any locality policy will likely cause this node to have a very low workload, rendering it almost useless.

7.4.3.2 Launching on queue system based environments

`COMPSs-Redis-Bundle` also includes a collection of scripts that allow the user to create an in-place Redis cluster with his/her COMPSs application. These scripts will create a cluster using only the COMPSs nodes provided by the queue system (e.g. SLURM, PBS, etc.). Some parameters can be tuned by the user via a `storage_props.cfg` file. This file must have the following form:

```
REDIS_HOME=some_path
REDIS_NODE_TIMEOUT=some_nonnegative_integer_value
REDIS_REPLICAS=some_nonnegative_integer_value
```

There are some observations regarding to this configuration file:

REDIS_HOME Must be equal to a path to some location that is **not** shared between nodes. This is the location where the Redis sandboxes for the instances will be created.
REDIS_NODE_TIMEOUT Must be a nonnegative integer number that represents the amount of milliseconds that must pass before Redis declares the cluster broken in the case that some instance is not available.
REDIS_REPLICAS Must be equal to a nonnegative integer. This value will represent the amount of replicas that a given shard will have. If possible, Redis will ensure that all replicas of a given shard will be on different nodes.

In order to run a COMPSs + Redis application on a queue system the user must add the following flags to the `enqueue_compss` command:

1. `--storage-home=${path_to_the_bundle_folder}` This must point to the root of the COMPSs-Redis bundle.

---

[2] https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

2. `--storage-props=${path_to_the_storage_props_file}` This must point to the `storage_props.cfg` mentioned above.
3. `--classpath=${path_to_COMPSs-redisPSCO.jar}` As in the previous section, the JAR with the storage API must be specified.
4. If you are running a Python application, also add the `--pythonpath=${app_path}:${path_to_the_-bundle_folder}` flag

---

Caution: As a requirement, the supercomputer **MUST NOT** kill daemonized processes running on the provided computing nodes during the execution.

7.5 Implement your own Storage interface for COMPSs

In order to implement an interface for a Storage framework, it is necessary to implement the Java SRI (mandatory), and depending on the desired language, implement the Python SRI and the specific SOI inheriting from the generic SOI provided by COMPSs.

7.5.1 Generic Storage Object Interface

Table 24 shows the functions that must exist in the storage object interface, that enables the object that inherits it to interact with the storage framework.

Table 24: SCO object definition

Name	Returns	Comments
Constructor	Nothing	Instantiates the object.
get_by_alias(String id)	Object	Retrieve the object with alias "name".
makePersistent(String id)	Nothing	Inserts the object in the storage framework with the id. If id is null, a random UUID will be computed instead.
deletePersistent()	Nothing	Removes the object from the storage. It does nothing if it was not already there.
getID()	String	Returns the current object identifier if the object is not persistent (null instead).

For example, the **makePersistent** function is intended to store the object content into the persistent storage, **deletePersistent** to remove it, and **getID** to provide the object identifier.

Important: An object will be considered persisted if the `getID` function retrieves something different from `None`.

This interface must be implemented in the target language desired (e.g. Java or Python).

7.5.2 Generic Storage Runtime Interfaces

Table 25 shows the functions that must exist in the storage runtime interface, that enables the COMPSs runtime to interact with the storage framework.

Table 25: Java API

Name	Returns	Comments	Signature
init(String storage_conf)	Nothing	Do any initialization action before starting to execute the application. Receives the storage configuration file path defined in the `runcompss` or `enqueue_composs` command.	public static void init(String storageConf) throws StorageException {}
finish()	Nothing	Do any finalization action after executing the application.	public static void finish() throws StorageException
getLocations(String id)	List<String>	Retrieve the locations where a particular object is from its identifier.	public static List<String> getLocations(String id) throws StorageException
getByID(String id)	Object	Retrieve an object from its identifier.	public static Object getByID(String id) throws StorageException
newReplica(String id, String hostName)	String	Create a new replica of an object in the storage framework.	public static void newReplica(String id, String hostName) throws StorageException
newVersion(String id, String hostname)	String	Create a new version of an object in the storage framework.	public static String newVersion(String id, String hostName) throws StorageException
consolidateVersion(String id)	Nothing	Consolidate a version of an object in the storage framework.	public static void consolidateVersion(String idFinal) throws StorageException
executeTask(String id, ...)	String	Execute the task into the datastore.	public static String executeTask(String id, String descriptor, Object[] values, String hostName, CallbackHandler callback) throws StorageException
getResult(CallbackEvent event())	Object	Retrieve the result of the execution into the storage framework.	public static Object getResult(CallbackEvent event) throws StorageException

7.5 Implement your own Storage interface for COMPSs 279

This functions enable the COMPSs runtime to keep the data consistency through the distributed execution.

In addition, Table 26 shows the functions that must exist in the storage runtime interface, that enables the COMPSs Python binding to interact with the storage framework. It is only necessary if the target language is Python.

Table 26: Python API

Name	Returns	Comments	Signature
init(String storage_conf)	Nothing	Do any initialization action before starting to execute the application. Receives the storage configuration file path defined in the `runcompss` or `enqueue_composs` command.	def initWorker(config_-file_path=None, **kwargs) # Does not return
finish()	Nothing	Do any finalization action after executing the application.	def finishWorker(**kwargs) # Does not return
getByID(String id)	Object	Retrieve an object from its identifier.	def getByID(id) # Returns the object with Id 'id'

TaskContext	Context	Define a task context (task enter/exit actions).	class TaskContext(object):
			
			  def __init__(self, logger, values, config_file_-path=None, **kwargs):
			    self.logger = logger
			    self.values = values
			    self.config_-file_path = config_file_-path
			
			  def __enter__(self):
			    # Do something for task prolog
			
			  def __exit__(self, type, value, traceback):
			    # Do something for task epilog

7.5.3 Storage Interface usage

7.5.3.1 Using `runcompss`

The first consideration is to deploy the storage framework, and then follow the next steps:

1. Create a `storage_conf.cfg` file with the configuation required by the `init` SRIs functions.
2. Add the flag `--classpath=${path_to_SRI.jar}` to the `runcompss` command.
3. Add the flag `--storage_conf="path to storage_conf.cfg file` to the `runcompss` command.
4. If you are running a Python app, also add the `--pythonpath=${app_path}:${path_to_the_bundle_-folder}/python` flag to the `runcompss` command.

As usual, the `project.xml` and `resources.xml` files must be correctly set. It must be noted that there can be nodes that are not COMPSs nodes (although **this is a highly unrecommended** practice since they will **always** need data that must be transferred from another node). Also, any locality policy will likely cause this node to have a very low workload.

7.5.3.2 Using `enqueue_compss`

In order to run a COMPSs + your storage on a queue system the user must add the following flags to the `enqueue_compss` command:

1. `--storage-home=${path_to_the_user_storage_folder}` This must point to the root of the user storage folder, where the scripts for starting (`storage_init.sh`) and stopping (`storage_stop.sh`) the storage framework must exist.
   - `storage_init.sh` **is called before the application execution and it** is intended to deploy the storage framework within the nodes provided by the queuing system. The parameters that receives are (in order):
     JOBID The job identifier provided by the queuing system.
     MASTER_NODE The name of the master node considered by COMPSs.
     STORAGE_MASTER_NODE The name of the node to be considere the master for the Storage framework.
     WORKER_NODES The set of nodes provided by the queuing system that will be considered as worker nodes by COMPSs.
     NETWORK Network interface (e.g. ib0)
     STORAGE_PROPS Storage properties file path (defined as `enqueue_compss` flag).
     VARIABLES_TO_BE_SOURCED If environment variables for the Storage framework need to be defined COMPSs provides an empty file to be filled by the `storage_init.sh` script, that will be sourced afterwards. This file is cleaned inmediately after sourcing it.
   - `storage_stop.sh` **is called after the application execution and it** is intended to stop the storage framework within the nodes provided by the queuing system. The parameters that receives are (in order):
     JOBID The job identifier provided by the queuing system.
     MASTER_NODE The name of the master node considered by COMPSs.
     STORAGE_MASTER_NODE The name of the node to be considere the master for the Storage framework.
     WORKER_NODES The set of nodes provided by the queuing system that will be considered as worker nodes by COMPSs.
     NETWORK Network interface (e.g. ib0)
     STORAGE_PROPS Storage properties file path (defined as `enqueue_compss` flag).
2. `--storage-props=${path_to_the_storage_props_file}` This must point to the `storage_props.cfg` specific for the storage framework that will be used by the start and stop scripts provided in the `--storage-home` path.
3. `--classpath=${path_to_SRI.jar}` As in the previous section, the JAR with the Java SRI must be specified.
4. If you are running a Python application, also add the `--pythonpath=${app_path}:${path_to_the_user_-storage_folder}` flag, where the SOI for Python must exist.

Chapter 8

Sample Applications

This section is intended to walk you through some COMPSs applications. Source code of different sample applications are available in https://github.com/bsc-wdc/apps

8.1 Java Sample applications

The first two examples in this section are simple applications developed in COMPSs to easily illustrate how to code, compile and run COMPSs applications. These applications are executed locally and show different ways to take advantage of all the COMPSs features.

The rest of the examples are more elaborated and consider the execution in a cloud platform where the VMs mount a common storage on **/sharedDisk** directory. This is useful in the case of applications that require working with big files, allowing to transfer data only once, at the beginning of the execution, and to enable the application to access the data directly during the rest of the execution.

The Virtual Machine available at our webpage (http://compss.bsc.es/) provides a development environment with all the applications listed in the following sections. The codes of all the applications can be found under the `/home/compss/tutorial_apps/java/` folder.

8.1.1 Hello World

The Hello Wolrd is a Java application that creates a task and prints a Hello World! message. Its purpose is to clarify that the COMPSs tasks output is redirected to the job files and it is **not** available at the standard output.

Next we provide the important parts of the application's code.

```java
// hello.Hello

public static void main(String[] args) throws Exception {
        // Check and get parameters
        if (args.length != 0) {
                usage();
                throw new Exception("[ERROR] Incorrect number of parameters");
        }

        // Hello World from main application
        System.out.println("Hello World! (from main application)");

        // Hello World from a task
        HelloImpl.sayHello();
}
```

As shown in the main code, this application has no input arguments.

```
// hello.HelloImpl

public static void sayHello() {
        System.out.println("Hello World! (from a task)");
}
```

Remember that, to run with COMPSs, java applications must provide an interface. For simplicity, in this example, the content of the interface only declares the task which has no parameters:

```
// hello.HelloItf

@Method(declaringClass = "hello.HelloImpl")
  void sayHello(
);
```

Notice that there is a first Hello World message printed from the main code and, a second one, printed inside a task. When executing sequentially this application users will be able to see both messages at the standard output. However, when executing this application with COMPSs, users will only see the message from the main code at the standard output. The message printed from the task will be stored inside the job log files.

Let's try it. First we proceed to compile the code by running the following instructions:

```
compss@bsc:~$ cd ~/tutorial_apps/java/hello/src/main/java/hello/
compss@bsc:~/tutorial_apps/java/hello/src/main/java/hello$ javac *.java
compss@bsc:~/tutorial_apps/java/hello/src/main/java/hello$ cd ..
compss@bsc:~/tutorial_apps/java/hello/src/main/java$ jar cf hello.jar hello
compss@bsc:~/tutorial_apps/java/hello/src/main/java$ mv hello.jar ~/tutorial_apps/java/hello/
↪jar/
```

Alternatively, this example application is prepared to be compiled with *maven*:

```
compss@bsc:~$ cd ~/tutorial_apps/java/hello/
compss@bsc:~/tutorial_apps/java/hello$ mvn clean package
```

Once done, we can sequentially execute the application by directly invoking the *jar* file.

```
compss@bsc:~$ cd ~/tutorial_apps/java/hello/jar/
compss@bsc:~/tutorial_apps/java/hello/jar$ java -cp hello.jar hello.Hello
Hello World! (from main application)
Hello World! (from a task)
```

And we can also execute the application with COMPSs:

```
compss@bsc:~$ cd ~/tutorial_apps/java/hello/jar/
compss@bsc:~/tutorial_apps/java/hello/jar$ runcompss -d hello.Hello
[  INFO] Using default execution type: compss
[  INFO] Using default location for project file: /opt/COMPSs/Runtime/configuration/xml/
↪projects/default_project.xml
[  INFO] Using default location for resources file: /opt/COMPSs/Runtime/configuration/xml/
↪resources/default_resources.xml


----------------- Executing hello.Hello --------------------------

WARNING: COMPSs Properties file is null. Setting default values
[(928)    API]  -  Deploying COMPSs Runtime v<version>
[(931)    API]  -  Starting COMPSs Runtime v<version>
[(931)    API]  -  Initializing components
```

(continues on next page)

```
[(1472)    API]  -  Ready to process tasks
Hello World! (from main application)
[(1474)    API]  -  Creating task from method sayHello in hello.HelloImpl
[(1474)    API]  -  There is 0 parameter
[(1477)    API]  -  No more tasks for app 1
[(4029)    API]  -  Getting Result Files 1
[(4030)    API]  -  Stop IT reached
[(4030)    API]  -  Stopping AP...
[(4031)    API]  -  Stopping TD...
[(4161)    API]  -  Stopping Comm...
[(4163)    API]  -  Runtime stopped
[(4166)    API]  -  Execution Finished


------------------------------------------------------------
```

Notice that the COMPSs execution is using the *-d* option to allow the job logging. Thus, we can check out the application jobs folder to look for the task output.

```
compss@bsc:~$ cd ~/.COMPSs/hello.Hello_01/jobs/
compss@bsc:~/.COMPSs/hello.Hello_01/jobs$ ls -1
job1_NEW.err
job1_NEW.out
compss@bsc:~/.COMPSs/hello.Hello_01/jobs$ cat job1_NEW.out
[JAVA EXECUTOR] executeTask - Begin task execution
WORKER - Parameters of execution:
  * Method type: METHOD
  * Method definition: [DECLARING CLASS=hello.HelloImpl, METHOD NAME=sayHello]
  * Parameter types:
  * Parameter values:
Hello World! (from a task)
[JAVA EXECUTOR] executeTask - End task execution
```

8.1.2 Simple

The Simple application is a Java application that increases a counter by means of a task. The counter is stored inside a file that is transferred to the worker when the task is executed. Thus, the tasks inferface is defined as follows:

```
// simple.SimpleItf

@Method(declaringClass = "simple.SimpleImpl")
void increment(
        @Parameter(type = Type.FILE, direction = Direction.INOUT) String file
);
```

Next we also provide the invocation of the task from the main code and the increment's method code.

```
// simple.Simple

public static void main(String[] args) throws Exception {
        // Check and get parameters
        if (args.length != 1) {
                usage();
                throw new Exception("[ERROR] Incorrect number of parameters");
        }
```

```java
        int initialValue = Integer.parseInt(args[0]);

        // Write value
        FileOutputStream fos = new FileOutputStream(fileName);
        fos.write(initialValue);
        fos.close();
        System.out.println("Initial counter value is " + initialValue);

        //Execute increment
        SimpleImpl.increment(fileName);

        // Write new value
        FileInputStream fis = new FileInputStream(fileName);
        int finalValue = fis.read();
        fis.close();
        System.out.println("Final counter value is " + finalValue);
}
```

```java
// simple.SimpleImpl

public static void increment(String counterFile) throws FileNotFoundException, IOException {
        // Read value
        FileInputStream fis = new FileInputStream(counterFile);
        int count = fis.read();
        fis.close();

        // Write new value
        FileOutputStream fos = new FileOutputStream(counterFile);
        fos.write(++count);
        fos.close();
}
```

Finally, to compile and execute this application users must run the following commands:

```
compss@bsc:~$ cd ~/tutorial_apps/java/simple/src/main/java/simple/
compss@bsc:~/tutorial_apps/java/simple/src/main/java/simple$ javac *.java
compss@bsc:~/tutorial_apps/java/simple/src/main/java/simple$ cd ..
compss@bsc:~/tutorial_apps/java/simple/src/main/java$ jar cf simple.jar simple
compss@bsc:~/tutorial_apps/java/simple/src/main/java$ mv simple.jar ~/tutorial_apps/java/
↪simple/jar/

compss@bsc:~$ cd ~/tutorial_apps/java/simple/jar
compss@bsc:~/tutorial_apps/java/simple/jar$ runcompss simple.Simple 1
compss@bsc:~/tutorial_apps/java/simple/jar$ runcompss simple.Simple 1
[  INFO] Using default execution type: compss
[  INFO] Using default location for project file: /opt/COMPSs/Runtime/configuration/xml/
↪projects/default_project.xml
[  INFO] Using default location for resources file: /opt/COMPSs/Runtime/configuration/xml/
↪resources/default_resources.xml


---------------- Executing simple.Simple --------------------------

WARNING: COMPSs Properties file is null. Setting default values
[(772)    API]  -  Starting COMPSs Runtime v<version>
Initial counter value is 1
Final counter value is 2
```

```
[(3813)    API] - Execution Finished


-------------------------------------------------------------
```

8.1.3 Increment

The Increment application is a Java application that increases N times three different counters. Each increase step is developed by a separated task. The purpose of this application is to show parallelism between the three counters.

Next we provide the main code of this application. The code inside the *increment* task is the same than the previous example.

```java
// increment.Increment

public static void main(String[] args) throws Exception {
        // Check and get parameters
        if (args.length != 4) {
                usage();
                throw new Exception("[ERROR] Incorrect number of parameters");
        }
        int N = Integer.parseInt(args[0]);
        int counter1 = Integer.parseInt(args[1]);
        int counter2 = Integer.parseInt(args[2]);
        int counter3 = Integer.parseInt(args[3]);

        // Initialize counter files
        System.out.println("Initial counter values:");
        initializeCounters(counter1, counter2, counter3);

        // Print initial counters state
        printCounterValues();

        // Execute increment tasks
        for (int i = 0; i < N; ++i) {
                IncrementImpl.increment(fileName1);
                IncrementImpl.increment(fileName2);
                IncrementImpl.increment(fileName3);
        }

        // Print final counters state (sync)
        System.out.println("Final counter values:");
        printCounterValues();
}
```

As shown in the main code, this application has 4 parameters that stand for:

1. **N:** Number of times to increase a counter
2. **InitialValue1:** Initial value for counter 1
3. **InitialValue2:** Initial value for counter 2
4. **InitialValue3:** Initial value for counter 3

Next we will compile and run the Increment application with the *-g* option to be able to generate the final graph at the end of the execution.

```
compss@bsc:~$ cd ~/tutorial_apps/java/increment/src/main/java/increment/
compss@bsc:~/tutorial_apps/java/increment/src/main/java/increment$ javac *.java
```

```
compss@bsc:~/tutorial_apps/java/increment/src/main/java/increment$ cd ..
compss@bsc:~/tutorial_apps/java/increment/src/main/java$ jar cf increment.jar increment
compss@bsc:~/tutorial_apps/java/increment/src/main/java$ mv increment.jar ~/tutorial_apps/
→java/increment/jar/

compss@bsc:~$ cd ~/tutorial_apps/java/increment/jar
compss@bsc:~/tutorial_apps/java/increment/jar$ runcompss -g increment.Increment 10 1 2 3
[  INFO] Using default execution type: compss
[  INFO] Using default location for project file: /opt/COMPSs/Runtime/configuration/xml/
→projects/default_project.xml
[  INFO] Using default location for resources file: /opt/COMPSs/Runtime/configuration/xml/
→resources/default_resources.xml

----------------- Executing increment.Increment -------------------------

WARNING: COMPSs Properties file is null. Setting default values
[(1028)    API]  -  Starting COMPSs Runtime v<version>
Initial counter values:
- Counter1 value is 1
- Counter2 value is 2
- Counter3 value is 3
Final counter values:
- Counter1 value is 11
- Counter2 value is 12
- Counter3 value is 13
[(4403)    API]  -  Execution Finished

------------------------------------------------------------
```

By running the *compss_gengraph* command users can obtain the task graph of the above execution. Next we provide the set of commands to obtain the graph show in Figure 44.

```
compss@bsc:~$ cd ~/.COMPSs/increment.Increment_01/monitor/
compss@bsc:~/.COMPSs/increment.Increment_01/monitor$ compss_gengraph complete_graph.dot
compss@bsc:~/.COMPSs/increment.Increment_01/monitor$ evince complete_graph.pdf
```

8.1.4 Matrix multiplication

The Matrix Multiplication (Matmul) is a pure Java application that multiplies two matrices in a direct way. The application creates 2 matrices of N x N size initialized with values, and multiply the matrices by blocks.

This application provides three different implementations that only differ on the way of storing the matrix:

matmul.objects.Matmul Matrix stored by means of objects
matmul.files.Matmul Matrix stored in files
matmul.arrays.Matmul Matrix represented by an array

In all the implementations the multiplication is implemented in the multiplyAccumulative method that is thus selected as the task to be executed remotely. As example, we we provide next the task implementation and the tasks interface for the objects implementation.

```
// matmul.objects.Block

public void multiplyAccumulative(Block a, Block b) {
        for (int i = 0; i < M; i++) {
                for (int j = 0; j < M; j++) {
```
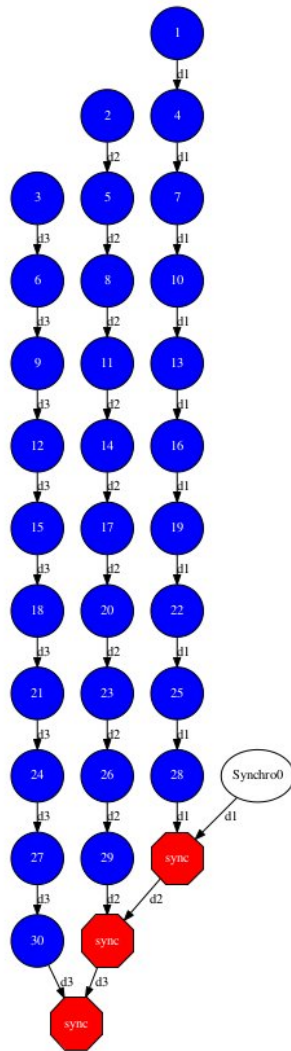
Figure 44: Java increment tasks graph

Figure 45: Matrix multiplication

```
                    for (int k = 0; k < M; k++) {
                            data[i][j] += a.data[i][k]*b.data[k][j];
                    }
            }
        }
}
```

```
// matmul.objects.MatmulItf

@Method(declaringClass = "matmul.objects.Block")
void multiplyAccumulative(
        @Parameter Block a,
        @Parameter Block b
);
```

In order to run the application the matrix dimension (number of blocks) and the dimension of each block have to be supplied. Consequently, any of the implementations must be executed by running the following command.

```
compss@bsc:~$ runcompss matmul.<IMPLEMENTATION_TYPE>.Matmul <matrix_dim> <block_dim>
```

Finally, we provide an example of execution for each implementation.

```
compss@bsc:~$ cd ~/tutorial_apps/java/matmul/jar/
compss@bsc:~/tutorial_apps/java/matmul/jar$ runcompss matmul.objects.Matmul 8 4
[  INFO] Using default execution type: compss
[  INFO] Using default location for project file: /opt/COMPSs/Runtime/configuration/xml/
↪projects/default_project.xml
[  INFO] Using default location for resources file: /opt/COMPSs/Runtime/configuration/xml/
↪resources/default_resources.xml

----------------- Executing matmul.objects.Matmul ------------------------

WARNING: COMPSs Properties file is null. Setting default values
[(887)    API]  -  Starting COMPSs Runtime v<version>
[LOG] MSIZE parameter value = 8
[LOG] BSIZE parameter value = 4
[LOG] Allocating A/B/C matrix space
[LOG] Computing Result
[LOG] Main program finished.
[(7415)    API]  -  Execution Finished

------------------------------------------------------------
```

```
compss@bsc:~$ cd ~/tutorial_apps/java/matmul/jar/
compss@bsc:~/tutorial_apps/java/matmul/jar$ runcompss matmul.files.Matmul 8 4
[  INFO] Using default execution type: compss
[  INFO] Using default location for project file: /opt/COMPSs/Runtime/configuration/xml/
↪projects/default_project.xml
[  INFO] Using default location for resources file: /opt/COMPSs/Runtime/configuration/xml/
↪resources/default_resources.xml

----------------- Executing matmul.files.Matmul ------------------------

WARNING: COMPSs Properties file is null. Setting default values
[(907)    API]  -  Starting COMPSs Runtime v<version>
[LOG] MSIZE parameter value = 8
```

```
[LOG] BSIZE parameter value = 4
[LOG] Computing result
[LOG] Main program finished.
[(9925)    API]  -  Execution Finished


----------------------------------------------------------------
```

```
compss@bsc:~$ cd ~/tutorial_apps/java/matmul/jar/
compss@bsc:~/tutorial_apps/java/matmul/jar$ runcompss matmul.arrays.Matmul 8 4
[  INFO] Using default execution type: compss
[  INFO] Using default location for project file: /opt/COMPSs/Runtime/configuration/xml/
↪projects/default_project.xml
[  INFO] Using default location for resources file: /opt/COMPSs/Runtime/configuration/xml/
↪resources/default_resources.xml

----------------- Executing matmul.arrays.Matmul ------------------------

WARNING: COMPSs Properties file is null. Setting default values
[(1062)    API]  -  Starting COMPSs Runtime v<version>
[LOG] MSIZE parameter value = 8
[LOG] BSIZE parameter value = 4
[LOG] Allocating C matrix space
[LOG] Computing Result
[LOG] Main program finished.
[(7811)    API]  -  Execution Finished


----------------------------------------------------------------
```

8.1.5 Sparse LU decomposition

SparseLU multiplies two matrices using the factorization method of LU decomposition, which factorizes a matrix as a product of a lower triangular matrix and an upper one.

$$
\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}.
$$

Figure 46: Sparse LU decomposition

The matrix is divided into N x N blocks on where 4 types of operations will be applied modifying the blocks: **lu0**, **fwd**, **bdiv** and **bmod**. These four operations are implemented in four methods that are selecetd as the tasks that will be executed remotely. In order to run the application the matrix dimension has to be provided.

As the previous application, the sparseLU is provided in three different implementations that only differ on the way of storing the matrix:

1. **sparseLU.objects.SparseLU** Matrix stored by means of objects
2. **sparseLU.files.SparseLU** Matrix stored in files
3. **sparseLU.arrays.SparseLU** Matrix represented by an array

Thus, the commands needed to execute the application is with each implementation are:

```
compss@bsc:~$ cd tutorial_apps/java/sparseLU/jar/
compss@bsc:~/tutorial_apps/java/sparseLU/jar$ runcompss sparseLU.objects.SparseLU 16 8
[  INFO] Using default execution type: compss
```

```
[  INFO] Using default location for project file: /opt/COMPSs/Runtime/configuration/xml/
↪projects/default_project.xml
[  INFO] Using default location for resources file: /opt/COMPSs/Runtime/configuration/xml/
↪resources/default_resources.xml

----------------- Executing sparseLU.objects.SparseLU -------------------------

WARNING: COMPSs Properties file is null. Setting default values
[(1221)    API]  -  Starting COMPSs Runtime v<version>
[LOG] Running with the following parameters:
[LOG]  - Matrix Size: 16
[LOG]  - Block Size:  8
[LOG] Initializing Matrix
[LOG] Computing SparseLU algorithm on A
[LOG] Main program finished.
[(13642)    API]  -  Execution Finished


-------------------------------------------------------------
```

```
compss@bsc:~$ cd tutorial_apps/java/sparseLU/jar/
compss@bsc:~/tutorial_apps/java/sparseLU/jar$ runcompss sparseLU.files.SparseLU 4 8
[  INFO] Using default execution type: compss
[  INFO] Using default location for project file: /opt/COMPSs/Runtime/configuration/xml/
↪projects/default_project.xml
[  INFO] Using default location for resources file: /opt/COMPSs/Runtime/configuration/xml/
↪resources/default_resources.xml

----------------- Executing sparseLU.files.SparseLU -------------------------

WARNING: COMPSs Properties file is null. Setting default values
[(1082)    API]  -  Starting COMPSs Runtime v<version>
[LOG] Running with the following parameters:
[LOG]  - Matrix Size: 16
[LOG]  - Block Size:  8
[LOG] Initializing Matrix
[LOG] Computing SparseLU algorithm on A
[LOG] Main program finished.
[(13605)    API]  -  Execution Finished


-------------------------------------------------------------
```

```
compss@bsc:~$ cd tutorial_apps/java/sparseLU/jar/
compss@bsc:~/tutorial_apps/java/sparseLU/jar$ runcompss sparseLU.arrays.SparseLU 8 8
[  INFO] Using default execution type: compss
[  INFO] Using default location for project file: /opt/COMPSs/Runtime/configuration/xml/
↪projects/default_project.xml
[  INFO] Using default location for resources file: /opt/COMPSs/Runtime/configuration/xml/
↪resources/default_resources.xml

----------------- Executing sparseLU.arrays.SparseLU -------------------------

WARNING: COMPSs Properties file is null. Setting default values
[(1082)    API]  -  Starting COMPSs Runtime v<version>
[LOG] Running with the following parameters:
[LOG]  - Matrix Size: 16
```

```
[LOG]  - Block Size:  8
[LOG] Initializing Matrix
[LOG] Computing SparseLU algorithm on A
[LOG] Main program finished.
[(13605)    API]  - Execution Finished


--------------------------------------------------------------
```

8.1.6 BLAST Workflow

BLAST is a widely-used bioinformatics tool for comparing primary biological sequence information, such as the amino-acid sequences of different proteins or the nucleotides of DNA sequences with sequence databases, identifying sequences that resemble the query sequence above a certain threshold. The work performed by the COMPSs Blast workflow is computationally intensive and embarrassingly parallel.
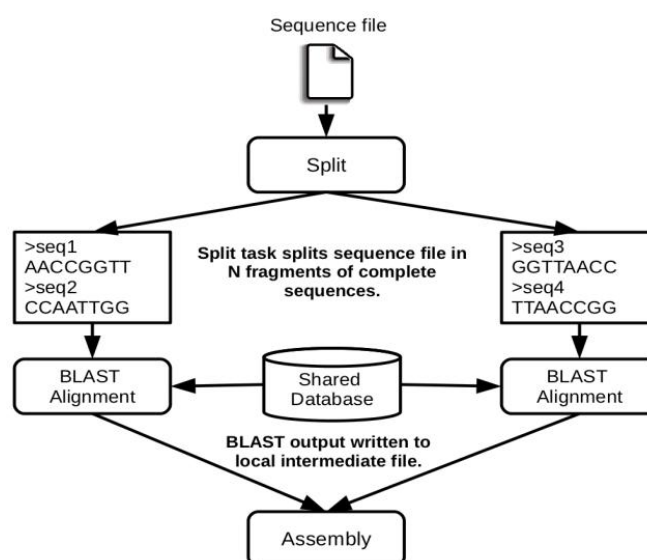
Figure 47: The COMPSs Blast workflow

The workflow describes the three blocks of the workflow implemented in the **Split**, **Align** and **Assembly** methods. The second one is the only method that is chosen to be executed remotely, so it is the unique method defined in the interface file. The **Split** method chops the query sequences file in N fragments, **Align** compares each sequence fragment against the database by means of the Blast binary, and **Assembly** combines all intermediate files into a single result file.

This application uses a database that will be on the shared disk space avoiding transferring the entire database (which can be large) between the virtual machines.

```
compss@bsc:~$ cp ~/workspace/blast/package/Blast.tar.gz /home/compss/
compss@bsc:~$ tar xzf Blast.tar.gz
```

The command line to execute the workflow:

```
compss@bsc:~$ runcompss blast.Blast <debug> \
                                    <bin_location> \
                                    <database_file> \
                                    <sequences_file> \
                                    <frag_number> \
                                    <tmpdir> \
                                    <output_file>
```

Where:

- **debug**: The debug flag of the application (true or false).
- **bin_location**: Path of the Blast binary.
- **database_file**: Path of database file; the shared disk /**sharedDisk**/ is suggested to avoid big data transfers.
- **sequences_file**: Path of sequences file.
- **frag_number**: Number of fragments of the original sequence file, this number determines the number of parallel Align tasks.
- **tmpdir**: Temporary directory (/**home**/**compss**/**tmp**/).
- **output_file**: Path of the result file.

Example:

```
compss@bsc:~$ runcompss blast.Blast true \
                        /home/compss/tutorial_apps/java/blast/binary/blastall \
                        /sharedDisk/Blast/databases/swissprot/swissprot \
                        /sharedDisk/Blast/sequences/sargasso_test.fasta \
                        4 \
                        /tmp/ \
                        /home/compss/out.txt
```

8.2 Python Sample applications

The first two examples in this section are simple applications developed in COMPSs to easily illustrate how to code, compile and run COMPSs applications. These applications are executed locally and show different ways to take advantage of all the COMPSs features.

The rest of the examples are more elaborated and consider the execution in a cloud platform where the VMs mount a common storage on /**sharedDisk** directory. This is useful in the case of applications that require working with big files, allowing to transfer data only once, at the beginning of the execution, and to enable the application to access the data directly during the rest of the execution.

The Virtual Machine available at our webpage (http://compss.bsc.es/) provides a development environment with all the applications listed in the following sections. The codes of all the applications can be found under the /home/compss/tutorial_apps/python/ folder.

8.2.1 Simple

The Simple application is a Python application that increases a counter by means of a task. The counter is stored inside a file that is transfered to the worker when the task is executed. Next, we provide the main code and the task declaration:

```python
from pycompss.api.task import task
from pycompss.api.parameter import FILE_INOUT


@task(filePath=FILE_INOUT)
def increment(filePath):
    # Read value
    fis = open(filePath, "r")
    value = fis.read()
    fis.close()

    # Write value
    fos = open(filePath, "w")
    fos.write(str(int(value) + 1))
    fos.close()
```

(continues on next page)

```python
def main_program():
    from pycompss.api.api import compss_open

    # Check and get parameters
    if len(sys.argv) != 2:
        exit(-1)
    initialValue = sys.argv[1]

    fileName = "counter"

    # Write value
    fos = open(fileName, "w")
    fos.write(initialValue)
    fos.close()
    print("Initial counter value is %s" % str(initialValue))

    # Execute increment
    increment(fileName)

    # Write new value
    fis = compss_open(fileName, "r+")
    finalValue = fis.read()
    fis.close()
    print("Final counter value is %s" % str(finalValue))


if __name__ == "__main__":
    main_program()
```

The simple application can be executed by invoking the `runcompss` command with the application file name and the initial counter value.

The following lines provide an example of its execution.

```
compss@bsc:~$ runcompss simple.py 1
[ INFO ] Inferred PYTHON language
[ INFO ] Using default location for project file: /opt/COMPSs//Runtime/configuration/xml/
↪projects/default_project.xml
[ INFO ] Using default location for resources file: /opt/COMPSs//Runtime/configuration/xml/
↪resources/default_resources.xml
[ INFO ] Using default execution type: compss


---------------- Executing simple.py -------------------------

WARNING: COMPSs Properties file is null. Setting default values
[(692)    API]  -  Starting COMPSs Runtime v3.1 (build 20221107-1044.
↪r7c414d34bd2ef4525a7146fbb80f57111e10f780)
Initial counter value is 1
Final counter value is 2
[(9286)   API]  -  Execution Finished


------------------------------------------------------------
```

8.2.2 Increment

The Increment application is a Python application that increases N times three different counters. Each increase step is developed by a separated task. The purpose of this application is to show parallelism between the three counters.

Next we provide the main code of this application. The code inside the *increment* task is the same than the previous example.

```python
# IMPORTS
import sys

# PyCOMPSs imports
from pycompss.api.task import task
from pycompss.api.parameter import FILE_INOUT
from pycompss.api.api import compss_open

# GLOBAL VARIABLES
FILENAME1 = "file1"
FILENAME2 = "file2"
FILENAME3 = "file3"


@task(file_path=FILE_INOUT)
def increment(file_path):
    """Increment the value contained within file_path.

    :param file_path: Path of the file that contains the value to be incremented.
    """
    # Read value
    fis = open(file_path, "r")
    value = fis.read()
    fis.close()

    # Write value
    fos = open(file_path, "w")
    fos.write(str(int(value) + 1))
    fos.close()


def usage():
    """Show the application usage."""
    print("[ERROR] Bad numnber of parameters")
    print(
        "    Usage: increment <num_iterations> "
        "<counter_value_1> <counter_value_2> <counter_value_3>"
    )


def initialize_counters(counter1, counter2, counter3):
    """Create the initial files with the given counter values.

    :param counter1: First counter.
    :param counter2: Second counter.
    :param counter3: Third counter.
    """
    # Write value counter 1
    fos = open(FILENAME1, "w")
```

(continues on next page)

```python
        fos.write(str(counter1))
        fos.close()

        # Write value counter 2
        fos = open(FILENAME2, "w")
        fos.write(str(counter2))
        fos.close()

        # Write value counter 3
        fos = open(FILENAME3, "w")
        fos.write(str(counter3))
        fos.close()


def print_counter_values():
    """Display the values contained in the counter files."""
    # Read value counter 1
    fis = compss_open(FILENAME1, "r+")
    counter1 = fis.read()
    fis.close()

    # Read value counter 1
    fis = compss_open(FILENAME2, "r+")
    counter2 = fis.read()
    fis.close()

    # Read value counter 1
    fis = compss_open(FILENAME3, "r+")
    counter3 = fis.read()
    fis.close()

    # Print values
    print("- Counter1 value is " + str(counter1))
    print("- Counter2 value is " + str(counter2))
    print("- Counter3 value is " + str(counter3))


def main_program():
    """Main increment function."""

    # Check parameters
    if len(sys.argv) != 5:
        usage()
        raise Exception("ERROR: Please fix the input parameters.")

    # Get parameters
    num_iterations = int(sys.argv[1])
    counter1 = int(sys.argv[2])
    counter2 = int(sys.argv[3])
    counter3 = int(sys.argv[4])

    # Initialize counter files
    initialize_counters(counter1, counter2, counter3)
    print("Initial counter values:")
    print_counter_values()
```

```
    # Execute increment
    for _ in range(num_iterations):
        increment(FILENAME1)
        increment(FILENAME2)
        increment(FILENAME3)

    # Write final counters state (sync)
    print("Final counter values:")
    print_counter_values()


if __name__ == "__main__":
    main_program()
```

As shown in the main code, this application has 4 parameters that stand for:

num_iterations Number of times to increase a counter
counter1 Initial value for counter 1
counter2 Initial value for counter 2
counter3 Initial value for counter 3

Next we run the Increment application with the `-g` option to be able to generate the final graph at the end of the execution.

```
compss@bsc:~/tutorial_apps/python/increment$ runcompss -g increment.py 10 1 2 3
[ INFO ] Inferred PYTHON language
[ INFO ] Using default location for project file: /opt/COMPSs//Runtime/configuration/xml/
↪projects/default_project.xml
[ INFO ] Using default location for resources file: /opt/COMPSs//Runtime/configuration/xml/
↪resources/default_resources.xml
[ INFO ] Using default execution type: compss


---------------- Executing incr.py ------------------------

WARNING: COMPSs Properties file is null. Setting default values
[(693)    API]  -  Starting COMPSs Runtime v3.0.rc2210 (build 20221026-1333.
↪r8e1717372084e4c839cba4ab821c543c080cbd10)
Initial counter values:
- Counter1 value is 1
- Counter2 value is 2
- Counter3 value is 3
Final counter values:
- Counter1 value is 11
- Counter2 value is 12
- Counter3 value is 13
[(9216)   API]  -  Execution Finished


-----------------------------------------------------------
```

By running the `compss_gengraph` command users can obtain the task graph of the above execution. Next we provide the set of commands to obtain the graph show in Figure 48.

```
compss@bsc:~$ cd ~/.COMPSs/increment.py_01/monitor/
compss@bsc:~/.COMPSs/increment.py_01/monitor$ compss_gengraph complete_graph.dot
compss@bsc:~/.COMPSs/increment.py_01/monitor$ evince complete_graph.pdf
```
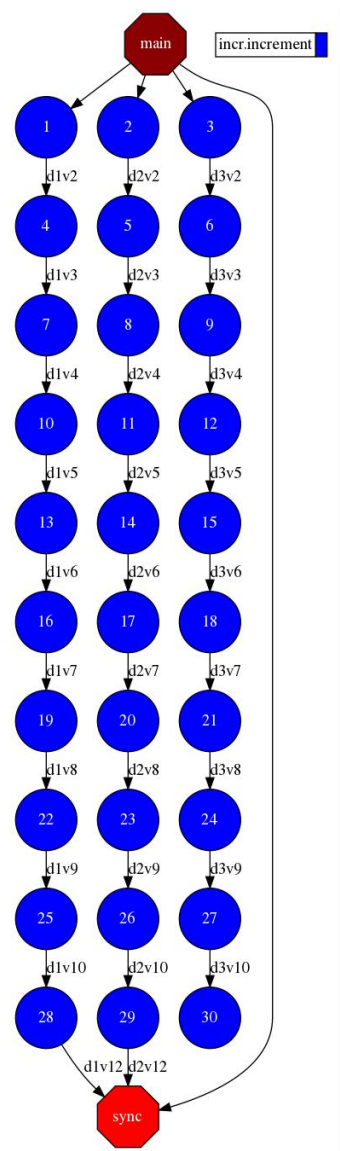
Figure 48: Python increment tasks graph

8.2.3 Kmeans

KMeans is machine-learning algorithm (NP-hard), popularly employed for cluster analysis in data mining, and interesting for benchmarking and performance evaluation.

The objective of the Kmeans algorithm to group a set of multidimensional points into a predefined number of clusters, in which each point belongs to the closest cluster (with the nearest mean distance), in an iterative process.

```python
import numpy as np
import time

from sklearn.metrics import pairwise_distances
from sklearn.metrics.pairwise import paired_distances

from pycompss.api.task import task
from pycompss.api.api import compss_wait_on
from pycompss.api.api import compss_barrier


@task(returns=np.ndarray)
def partial_sum(fragment, centres):
    partials = np.zeros((centres.shape[0], 2), dtype=object)
    close_centres = pairwise_distances(fragment, centres).argmin(axis=1)
    for center_idx, _ in enumerate(centres):
        indices = np.argwhere(close_centres == center_idx).flatten()
        partials[center_idx][0] = np.sum(fragment[indices], axis=0)
        partials[center_idx][1] = indices.shape[0]
    return partials


@task(returns=dict)
def merge(*data):
    accum = data[0].copy()
    for d in data[1:]:
        accum += d
    return accum


def converged(old_centres, centres, epsilon, iteration, max_iter):
    if old_centres is None:
        return False
    dist = np.sum(paired_distances(centres, old_centres))
    return dist < epsilon**2 or iteration >= max_iter


def recompute_centres(partials, old_centres, arity):
    centres = old_centres.copy()
    while len(partials) > 1:
        partials_subset = partials[:arity]
        partials = partials[arity:]
        partials.append(merge(*partials_subset))
    partials = compss_wait_on(partials)
    for idx, sum_ in enumerate(partials[0]):
        if sum_[1] != 0:
            centres[idx] = sum_[0] / sum_[1]
    return centres
```

```python
def kmeans_frag(
    fragments,
    dimensions,
    num_centres=10,
    iterations=20,
    seed=0.0,
    epsilon=1e-9,
    arity=50,
):
    """
    A fragment-based K-Means algorithm.
    Given a set of fragments, the desired number of clusters and the
    maximum number of iterations, compute the optimal centres and the
    index of the centre for each point.
    :param fragments: Number of fragments
    :param dimensions: Number of dimensions
    :param num_centres: Number of centres
    :param iterations: Maximum number of iterations
    :param seed: Random seed
    :param epsilon: Epsilon (convergence distance)
    :param arity: Reduction arity
    :return: Final centres
    """
    # Set the random seed
    np.random.seed(seed)
    # Centres is usually a very small matrix, so it is affordable to have it in
    # the master.
    centres = np.asarray([np.random.random(dimensions) for _ in range(num_centres)])
    # Note: this implementation treats the centres as files, never as PSCOs.
    old_centres = None
    iteration = 0
    while not converged(old_centres, centres, epsilon, iteration, iterations):
        print("Doing iteration #%d/%d" % (iteration + 1, iterations))
        old_centres = centres.copy()
        partials = []
        for frag in fragments:
            partial = partial_sum(frag, old_centres)
            partials.append(partial)
        centres = recompute_centres(partials, old_centres, arity)
        iteration += 1
    return centres


def parse_arguments():
    """
    Parse command line arguments. Make the program generate
    a help message in case of wrong usage.
    :return: Parsed arguments
    """
    import argparse

    parser = argparse.ArgumentParser(description="KMeans Clustering.")
    parser.add_argument(
        "-s", "--seed", type=int, default=0, help="Pseudo-random seed. Default = 0"
    )
```

```python
    parser.add_argument(
        "-n",
        "--numpoints",
        type=int,
        default=100,
        help="Number of points. Default = 100",
    )
    parser.add_argument(
        "-d",
        "--dimensions",
        type=int,
        default=2,
        help="Number of dimensions. Default = 2",
    )
    parser.add_argument(
        "-c",
        "--num_centres",
        type=int,
        default=5,
        help="Number of centres. Default = 2",
    )
    parser.add_argument(
        "-f",
        "--fragments",
        type=int,
        default=10,
        help="Number of fragments." + " Default = 10. Condition: fragments < points",
    )
    parser.add_argument(
        "-m",
        "--mode",
        type=str,
        default="uniform",
        choices=["uniform", "normal"],
        help="Distribution of points. Default = uniform",
    )
    parser.add_argument(
        "-i", "--iterations", type=int, default=20, help="Maximum number of iterations"
    )
    parser.add_argument(
        "-e",
        "--epsilon",
        type=float,
        default=1e-9,
        help="Epsilon. Kmeans will stop when:" + " |old - new| < epsilon.",
    )
    parser.add_argument(
        "-a",
        "--arity",
        type=int,
        default=50,
        help="Arity of the reduction carried out during \
                        the computation of the new centroids",
    )
    return parser.parse_args()
```

```python
@task(returns=1)
def generate_fragment(points, dim, mode, seed):
    """
    Generate a random fragment of the specified number of points using the
    specified mode and the specified seed. Note that the generation is
    distributed (the master will never see the actual points).
    :param points: Number of points
    :param dim: Number of dimensions
    :param mode: Dataset generation mode
    :param seed: Random seed
    :return: Dataset fragment
    """
    # Random generation distributions
    rand = {
        "normal": lambda k: np.random.normal(0, 1, k),
        "uniform": lambda k: np.random.random(k),
    }
    r = rand[mode]
    np.random.seed(seed)
    mat = np.asarray([r(dim) for __ in range(points)])
    # Normalize all points between 0 and 1
    mat -= np.min(mat)
    mx = np.max(mat)
    if mx > 0.0:
        mat /= mx

    return mat


def main(
    seed,
    numpoints,
    dimensions,
    num_centres,
    fragments,
    mode,
    iterations,
    epsilon,
    arity,
):
    """
    This will be executed if called as main script. Look at the kmeans_frag
    for the KMeans function.
    This code is used for experimental purposes.
    I.e it generates random data from some parameters that determine the size,
    dimensionality and etc and returns the elapsed time.
    :param seed: Random seed
    :param numpoints: Number of points
    :param dimensions: Number of dimensions
    :param num_centres: Number of centres
    :param fragments: Number of fragments
    :param mode: Dataset generation mode
    :param iterations: Number of iterations
    :param epsilon: Epsilon (convergence distance)
    :param arity: Reduction arity
```

```python
    :return: None
    """
    start_time = time.time()

    # Generate the data
    fragment_list = []
    # Prevent infinite loops
    points_per_fragment = max(1, numpoints // fragments)

    for l in range(0, numpoints, points_per_fragment):
        # Note that the seed is different for each fragment.
        # This is done to avoid having repeated data.
        r = min(numpoints, l + points_per_fragment)

        fragment_list.append(generate_fragment(r - l, dimensions, mode, seed + l))

    compss_barrier()
    print("Generation/Load done")
    initialization_time = time.time()
    print("Starting kmeans")

    # Run kmeans
    centres = kmeans_frag(
        fragments=fragment_list,
        dimensions=dimensions,
        num_centres=num_centres,
        iterations=iterations,
        seed=seed,
        epsilon=epsilon,
        arity=arity,
    )
    compss_barrier()
    print("Ending kmeans")
    kmeans_time = time.time()

    print("-----------------------------------------")
    print("-------------- RESULTS ------------------")
    print("-----------------------------------------")
    print("Initialization time: %f" % (initialization_time - start_time))
    print("Kmeans time: %f" % (kmeans_time - initialization_time))
    print("Total time: %f" % (kmeans_time - start_time))
    print("-----------------------------------------")
    centres = compss_wait_on(centres)
    print("CENTRES:")
    print(centres)
    print("-----------------------------------------")


if __name__ == "__main__":
    options = parse_arguments()
    main(**vars(options))
```

The kmeans application can be executed by invoking the `runcompss` command with the desired parameters (in this case we use `-g` to generate the task depedency graph) and application. The following lines provide an example of its execution considering 10M points, of 3 dimensions, divided into 8 fragments, looking for 8 clusters and a maximum number of iterations set to 10.

```
compss@bsc:~$ runcompss -g kmeans.py -n 10240000 -f 8 -d 3 -c 8 -i 10
[ INFO ] Inferred PYTHON language
[ INFO ] Using default location for project file: /opt/COMPSs//Runtime/configuration/xml/
↪projects/default_project.xml
[ INFO ] Using default location for resources file: /opt/COMPSs//Runtime/configuration/xml/
↪resources/default_resources.xml
[ INFO ] Using default execution type: compss

----------------- Executing kmeans.py --------------------------

WARNING: COMPSs Properties file is null. Setting default values
[(877)    API]  -  Starting COMPSs Runtime v3.1 (build 20221107-1044.
↪r7c414d34bd2ef4525a7146fbb80f57111e10f780)
Generation/Load done
Starting kmeans
Doing iteration #1/10
Doing iteration #2/10
Doing iteration #3/10
Doing iteration #4/10
Doing iteration #5/10
Doing iteration #6/10
Doing iteration #7/10
Doing iteration #8/10
Doing iteration #9/10
Doing iteration #10/10
Ending kmeans
----------------------------------------
-------------- RESULTS ------------------
----------------------------------------
Initialization time: 11.720157
Kmeans time: 21.592080
Total time: 33.312237
----------------------------------------
CENTRES:
[[0.69828619 0.74530239 0.48171237]
 [0.54765031 0.20253203 0.21191319]
 [0.24201614 0.74466519 0.75560619]
 [0.21853824 0.66978432 0.23275263]
 [0.7724606  0.68585097 0.16247501]
 [0.22674374 0.23357703 0.67253838]
 [0.75316023 0.73748642 0.83358697]
 [0.75816592 0.23837464 0.71580623]]
----------------------------------------
[(39715)    API]  -  Execution Finished

------------------------------------------------------------
```

Figure 49 depicts the generated task dependency graph. The dataset generation can be identified in the 8 blue tasks, while the five iterations appear next. Between the iteration there is a synchronization which corresponds to the convergence/max iterations check.
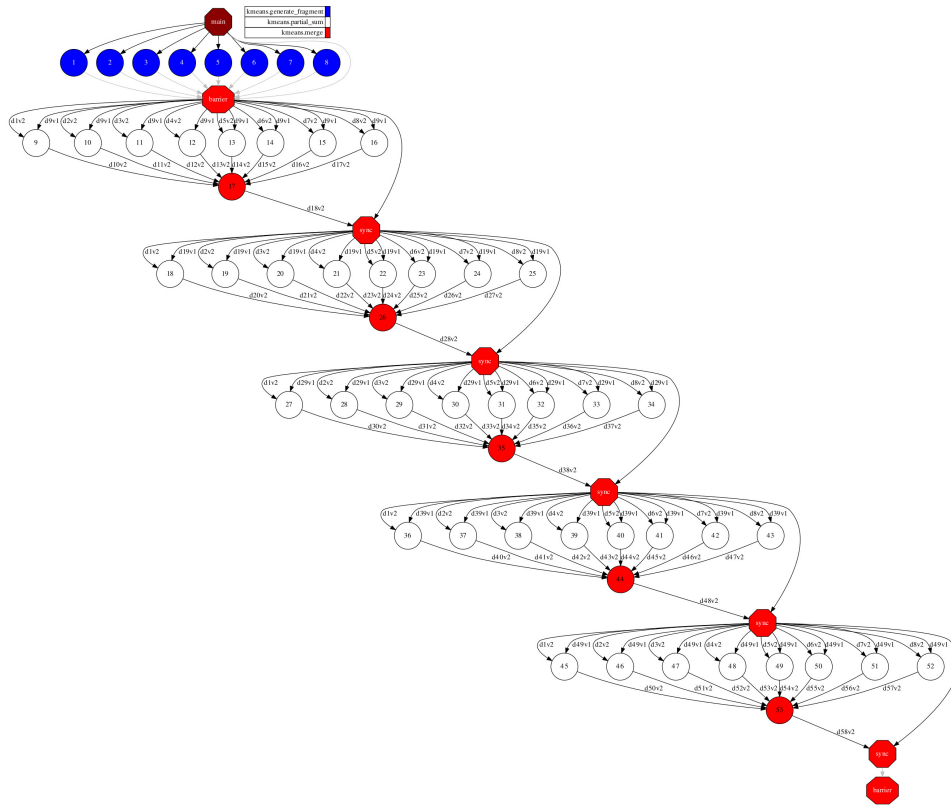
Figure 49: Python kmeans tasks graph

8.2.4 Matmul

The matmul performs the matrix multiplication of two matrices.

```python
import time
import numpy as np

from pycompss.api.task import task
from pycompss.api.parameter import INOUT
from pycompss.api.api import compss_barrier
from pycompss.api.api import compss_wait_on


@task(returns=1)
def generate_block(size, num_blocks, seed=0, set_to_zero=False):
    """
    Generate a square block of given size.
    :param size: <Integer> Block size
    :param num_blocks: <Integer> Number of blocks
    :param seed: <Integer> Random seed
    :param set_to_zero: <Boolean> Set block to zeros
    :return: Block
    """
    np.random.seed(seed)
    if not set_to_zero:
        b = np.random.random((size, size))
        # Normalize matrix to ensure more numerical precision
```

(continues on next page)

```python
        b /= np.sum(b) * float(num_blocks)
    else:
        b = np.zeros((size, size))
    return b


@task(C=INOUT)
def fused_multiply_add(A, B, C):
    """
    Multiplies two Blocks and accumulates the result in an INOUT Block (FMA).
    :param A: Block A
    :param B: Block B
    :param C: Result Block
    :return: None
    """
    C += np.dot(A, B)


def dot(A, B, C):
    """
    A COMPSs blocked matmul algorithm.
    :param A: Block A
    :param B: Block B
    :param C: Result Block
    :return: None
    """
    n, m = len(A), len(B[0])
    # as many rows as A, as many columns as B
    for i in range(n):
        for j in range(m):
            for k in range(n):
                fused_multiply_add(A[i][k], B[k][j], C[i][j])


def main(num_blocks, elems_per_block, seed):
    """
    Matmul main.
    :param num_blocks: <Integer> Number of blocks
    :param elems_per_block: <Integer> Number of elements per block
    :param seed: <Integer> Random seed
    :return: None
    """
    start_time = time.time()

    # Generate the dataset in a distributed manner
    # i.e: avoid having the master a whole matrix
    A, B, C = [], [], []
    matrix_name = ["A", "B"]
    for i in range(num_blocks):
        for l in [A, B, C]:
            l.append([])
        # Keep track of blockId to initialize with different random seeds
        bid = 0
        for j in range(num_blocks):
            for ix, l in enumerate([A, B]):
                l[-1].append(generate_block(elems_per_block,
```

```
                                    num_blocks,
                                    seed=seed + bid))
            bid += 1
        C[-1].append(generate_block(elems_per_block,
                                    num_blocks,
                                    set_to_zero=True))
    compss_barrier()
    initialization_time = time.time()

    # Do matrix multiplication
    dot(A, B, C)

    compss_barrier()
    multiplication_time = time.time()

    print("----------------------------------------")
    print("-------------- RESULTS -----------------")
    print("----------------------------------------")
    print("Initialization time: %f" % (initialization_time -
                                       start_time))
    print("Multiplication time: %f" % (multiplication_time -
                                       initialization_time))
    print("Total time: %f" % (multiplication_time - start_time))
    print("----------------------------------------")


def parse_args():
    """
    Arguments parser.
    Code for experimental purposes.
    :return: Parsed arguments.
    """
    import argparse
    description = 'COMPSs blocked matmul implementation'
    parser = argparse.ArgumentParser(description=description)
    parser.add_argument('-b', '--num_blocks', type=int, default=1,
                        help='Number of blocks (N in NxN)'
                        )
    parser.add_argument('-e', '--elems_per_block', type=int, default=2,
                        help='Elements per block (N in NxN)'
                        )
    parser.add_argument('--seed', type=int, default=0,
                        help='Pseudo-Random seed'
                        )
    return parser.parse_args()


if __name__ == "__main__":
    opts = parse_args()
    main(**vars(opts))
```

The matrix multiplication application can be executed by invoking the `runcompss` command with the desired parameters (in this case we use `-g` to generate the task depedency graph) and application. The following lines provide an example of its execution considering 4 x 4 Blocks of 1024 x 1024 elements each block, which conforms matrices of 4096 x 4096 elements.

```
compss@bsc:~$ runcompss -g matmul.py -b 4 -e 1024
[ INFO ] Inferred PYTHON language
[ INFO ] Using default location for project file: /opt/COMPSs//Runtime/configuration/xml/
↪projects/default_project.xml
[ INFO ] Using default location for resources file: /opt/COMPSs//Runtime/configuration/xml/
↪resources/default_resources.xml
[ INFO ] Using default execution type: compss

----------------- Executing matmul.py -------------------------

WARNING: COMPSs Properties file is null. Setting default values
[(882)    API] -  Starting COMPSs Runtime v3.1 (build 20221107-1044.
↪r7c414d34bd2ef4525a7146fbb80f57111e10f780)
-----------------------------------------
-------------- RESULTS ------------------
-----------------------------------------
Initialization time: 5.322150
Multiplication time: 4.371828
Total time: 9.693978
-----------------------------------------
[(14620)    API] -  Execution Finished


----------------------------------------------------------
```

Figure 50 depicts the generated task dependency graph. The dataset generation can be identified in the blue tasks, while the white tasks represent the multiplication of a block with another.
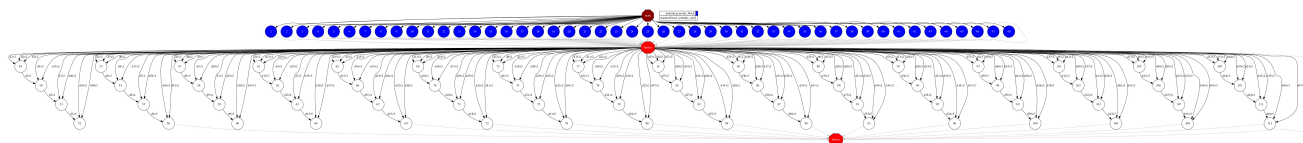
Figure 50: Python matrix multiplication tasks graph

8.2.5 Lysozyme in water

This example will guide a new user through the usage of the `@binary`, `@mpi` and `@constraint` decorators for setting up a simulation system containing a set of proteins (lysozymes) in boxes of water with ions. Each step contains an explanation of input and output, using typical settings for general use.

Extracted from: http://www.mdtutorials.com/gmx/lysozyme/index.html Originally done by: Justin A. Lemkul, Ph.D. From: Virginia Tech Department of Biochemistry

---

Note: This example reaches up to stage 4 (energy minimization).

Important: This application requires **Gromacs** `gmx` and `gmx_mpi`.

```python
from os import listdir
from os.path import isfile, join
import sys

from pycompss.api.task import task
```

(continues on next page)

```python
from pycompss.api.constraint import constraint
from pycompss.api.binary import binary
from pycompss.api.mpi import mpi
from pycompss.api.parameter import *


# ############ #
# Step 1 tasks #
# ############ #

@binary(binary='${GMX_BIN}/gmx')
@task(protein=FILE_IN,
      structure=FILE_OUT,
      topology=FILE_OUT)
def generate_topology(mode='pdb2gmx',
                      protein_flag='-f', protein=None,
                      structure_flag='-o', structure=None,
                      topology_flag='-p', topology=None,
                      flags='-ignh',
                      forcefield_flag='-ff', forcefield='oplsaa',
                      water_flag='-water', water='spce'):
    # Command: gmx pdb2gmx -f protein.pdb -o structure.gro -p topology.top -ignh -ff amber03 -
→water tip3p
    pass

# ############ #
# Step 2 tasks #
# ############ #

@binary(binary='${GMX_BIN}/gmx')
@task(structure=FILE_IN,
      structure_newbox=FILE_OUT)
def define_box(mode='editconf',
               structure_flag='-f', structure=None,
               structure_newbox_flag='-o', structure_newbox=None,
               center_flag='-c',
               distance_flag='-d', distance='1.0',
               boxtype_flag='-bt', boxtype='cubic'):
    # Command: gmx editconf -f structure.gro -o structure_newbox.gro -c -d 1.0 -bt cubic
    pass

# ############ #
# Step 3 tasks #
# ############ #

@binary(binary='${GMX_BIN}/gmx')
@task(structure_newbox=FILE_IN,
      protein_solv=FILE_OUT,
      topology=FILE_IN)
def add_solvate(mode='solvate',
                structure_newbox_flag='-cp', structure_newbox=None,
                configuration_solvent_flag='-cs', configuration_solvent='spc216.gro',
                protein_solv_flag='-o', protein_solv=None,
                topology_flag='-p', topology=None):
    # Command: gmx solvate -cp structure_newbox.gro -cs spc216.gro -o protein_solv.gro -p
→topology.top
```

　　　　　　　　　　　　　　　　　　　　　　Chapter 8. Sample Applications

```python
    pass

# ############# #
# Step 4 tasks #
# ############# #


@binary(binary='${GMX_BIN}/gmx')
@task(conf=FILE_IN,
      protein_solv=FILE_IN,
      topology=FILE_IN,
      output=FILE_OUT)
def assemble_tpr(mode='grompp',
                 conf_flag='-f', conf=None,
                 protein_solv_flag='-c', protein_solv=None,
                 topology_flag='-p', topology=None,
                 output_flag='-o', output=None):
    # Command: gmx grompp -f ions.mdp -c protein_solv.gro -p topology.top -o ions.tpr
    pass


@binary(binary='${GMX_BIN}/gmx')
@task(ions=FILE_IN,
      output=FILE_OUT,
      topology=FILE_IN,
      group={Type:FILE_IN, StdIOStream:STDIN})
def replace_solvent_with_ions(mode='genion',
                              ions_flag='-s', ions=None,
                              output_flag='-o', output=None,
                              topology_flag='-p', topology=None,
                              pname_flag='-pname', pname='NA',
                              nname_flag='-nname', nname='CL',
                              neutral_flag='-neutral',
                              group=None):
    # Command: gmx genion -s ions.tpr -o 1AKI_solv_ions.gro -p topol.top -pname NA -nname CL -
    →neutral < ../config/genion.group
    pass


# ############# #
# Step 5 tasks #
# ############# #


computing_units = "24"
computing_nodes = "1"


@constraint(computing_units=computing_units)
@mpi(runner="mpirun", binary="gmx_mpi", computing_nodes=computing_nodes)
@task(em=FILE_IN,
      em_energy=FILE_OUT)
def energy_minimization(mode='mdrun',
                        verbose_flag='-v',
                        ompthreads_flag='-ntomp', ompthreads='0',
                        em_flag='-s', em=None,
                        em_energy_flag='-e', em_energy=None):
    # Command: gmx mdrun -v -s em.tpr
    pass


# ############# #
```

```python
# Step 6 tasks #
# ############ #

@binary(binary='${GMX_BIN}/gmx')
@task(em=FILE_IN,
      output=FILE_OUT,
      selection={Type:FILE_IN, StdIOStream:STDIN})
def energy_analisis(mode='energy',
                    em_flag='-f', em=None,
                    output_flag='-o', output=None,
                    selection=None):
    # Command: gmx energy -f em.edr -o output.xvg
    pass


# ############# #
# MAIN FUNCTION #
# ############# #

def main(dataset_path, output_path, config_path):
    print("Starting demo")

    protein_names = []
    protein_pdbs = []

    # Look for proteins in the dataset folder
    for f in listdir(dataset_path):
        if isfile(join(dataset_path, f)):
            protein_names.append(f.split('.')[0])
            protein_pdbs.append(join(dataset_path, f))
    proteins = zip(protein_names, protein_pdbs)

    # Iterate over the proteins and process them
    result_image_paths = []
    for name, pdb in proteins:
        # 1st step - Generate topology
        structure = join(output_path, name + '.gro')
        topology = join(output_path, name + '.top')
        generate_topology(protein=pdb,
                          structure=structure,
                          topology=topology)
        # 2nd step - Define box
        structure_newbox = join(output_path, name + '_newbox.gro')
        define_box(structure=structure,
                   structure_newbox=structure_newbox)
        # 3rd step - Add solvate
        protein_solv = join(output_path, name + '_solv.gro')
        add_solvate(structure_newbox=structure_newbox,
                    protein_solv=protein_solv,
                    topology=topology)
        # 4th step - Add ions
        # Assemble with ions.mdp
        ions_conf = join(config_path, 'ions.mdp')
        ions = join(output_path, name + '_ions.tpr')
        assemble_tpr(conf=ions_conf,
                     protein_solv=protein_solv,
```

```
                    topology=topology,
                    output=ions)
        protein_solv_ions = join(output_path, name + '_solv_ions.gro')
        group = join(config_path, 'genion.group')   # 13 = SOL
        replace_solvent_with_ions(ions=ions,
                                  output=protein_solv_ions,
                                  topology=topology,
                                  group=group)
        # 5th step - Minimize energy
        # Reasemble with minim.mdp
        minim_conf = join(config_path, 'minim.mdp')
        em = join(output_path, name + '_em.tpr')
        assemble_tpr(conf=minim_conf,
                     protein_solv=protein_solv_ions,
                     topology=topology,
                     output=em)
        em_energy = join(output_path, name + '_em_energy.edr')
        energy_minimization(em=em,
                            em_energy=em_energy)
        # 6th step - Energy analysis (generate xvg image)
        energy_result = join(output_path, name + '_potential.xvg')
        energy_selection = join(config_path, 'energy.selection')  # 10 = potential
        energy_analisis(em=em_energy,
                        output=energy_result,
                        selection=energy_selection)


if __name__=='__main__':
    config_path = sys.argv[1]
    dataset_path = sys.argv[2]
    output_path = sys.argv[3]

    main(dataset_path, output_path, config_path)
```

This application can be executed by invoking the `runcompss` command defining the `config_path`, `dataset_path` and `output_path` where the application inputs and outputs are. For the sake of completeness, we show how to execute this application in a Supercomputer. In this case, the execution will be enqueued in the supercomputer queuing system (e.g. SLURM) through the use of the `enqueue_compss` command, where all parameters used in runcompss must appear, as well as some parameters required for the queuing system (e.g. walltime).

The following code shows a bash script to submit the execution in MareNostrum IV supercomputer:

```
#!/bin/bash -e

# Define script variables
scriptDir=$(pwd)/$(dirname $0)
execFile=${scriptDir}/src/lysozyme_in_water.py
appClasspath=${scriptDir}/src/
appPythonpath=${scriptDir}/src/

# Retrieve arguments
numNodes=$1
executionTime=$2
tracing=$3

# Leave application args on $@
shift 3
```

```
# Load necessary modules
module purge
module load intel/2017.4 impi/2017.4 mkl/2017.4 bsc/1.0
export COMPSS_PYTHON_VERSION=3
module load COMPSs/3.1.pr
module load gromacs/2016.4    # exposes gmx_mpi binary

export GMX_BIN=/home/user/lysozyme5.1.2/bin    # exposes gmx binary

# Enqueue the application
enqueue_compss \
    --num_nodes=$numNodes \
    --exec_time=$executionTime \
    --master_working_dir=/gpfs/home/user/lysozyme/tmpFiles/ \
    --worker_working_dir=/gpfs/home/user/lysozyme/ \
    --tracing=$tracing \
    --graph=true \
    -d \
    --classpath=$appClasspath \
    --pythonpath=$appPythonpath \
    --lang=python \
    $execFile $@


####################################################
# APPLICATION EXECUTION EXAMPLE
# Call:
#       ./launch_md.sh <NUMBER_OF_NODES> <EXECUTION_TIME> <TRACING> <CONFIG_PATH> <DATASET_
→PATH> <OUTPUT_PATH>
#
# Example:
#       ./launch_md.sh 2 10 false $(pwd)/config/ $(pwd)/dataset/ $(pwd)/output/
#
####################################################
```

Having the `1aki.pdb`, `1u3m.pdb` and `1xyw.pdb` proteins in the `dataset` folder, the execution of this script produces the submission of the job with the following output:

```
$ ./launch_md.sh 2 10 false $(pwd)/config/ $(pwd)/dataset/ $(pwd)/output/
remove mkl/2017.4 (LD_LIBRARY_PATH)
remove impi/2017.4 (PATH, MANPATH, LD_LIBRARY_PATH)
Set INTEL compilers as MPI wrappers backend
load impi/2017.4 (PATH, MANPATH, LD_LIBRARY_PATH)
load mkl/2017.4 (LD_LIBRARY_PATH)
load java/8u131 (PATH, MANPATH, JAVA_HOME, JAVA_ROOT, JAVA_BINDIR, SDK_HOME, JDK_HOME, JRE_
→HOME)
load papi/5.5.1 (PATH, LD_LIBRARY_PATH, C_INCLUDE_PATH)
load PYTHON/3.7.4 (PATH, MANPATH, LD_LIBRARY_PATH, LIBRARY_PATH, PKG_CONFIG_PATH, C_INCLUDE_
→PATH, CPLUS_INCLUDE_PATH, PYTHONHOME, PYTHONPATH)
load COMPSs/3.1 (PATH, CLASSPATH, MANPATH, GAT_LOCATION, COMPSS_HOME, JAVA_TOOL_OPTIONS,␣
→LDFLAGS, CPPFLAGS)
load gromacs/2016.4 (PATH, LD_LIBRARY_PATH)

SC Configuration:          default.cfg
JobName:                   COMPSs
```

```
Queue:                  default
Reservation:            disabled
Num Nodes:              2
Num Switches:           0
GPUs per node:          0
Job dependency:         None
Exec-Time:              00:10:00
QoS:                    debug
Constraints:            disabled
Storage Home:           null
Storage Properties:
Other:
                    --sc_cfg=default.cfg
                    --qos=debug
                    --master_working_dir=/gpfs/home/user/lysozyme/tmpFiles/
                    --worker_working_dir=/gpfs/home/user/lysozyme/
                    --tracing=false
                    --graph=true
                    --classpath=/home/user/lysozyme/./src/
                    --pythonpath=/home/user/lysozyme/./src/
                    --lang=python /home/user/lysozyme/./src/lysozyme_in_water.py /home/user/
↪lysozyme/config/ /home/user/lysozyme/dataset/ /home/user/lysozyme/output/

Temp submit script is: /scratch/tmp/tmp.sMHLsaTUJj
Requesting 96 processes
Submitted batch job 10178129
```

Once executed, it produces the `compss-10178129.out` file, containing all the standard output messages flushed during the execution:

```
$ cat compss-10178129.out

------ Launching COMPSs application ------
[  INFO] Using default execution type: compss
[  INFO] Relative Classpath resolved: /home/user/lysozyme/./src/:

----------------- Executing lysozyme_in_water.py --------------------------
[(692)    API]  -  Starting COMPSs Runtime v3.1 (build 20221107-1044.
↪r7c414d34bd2ef4525a7146fbb80f57111e10f780)
Starting demo

# Here it takes some time to process the dataset

[(290788)    API]  -  Execution Finished

------------------------------------------------------------
[LAUNCH_COMPSS] Waiting for application completion
```

Since the execution has been performed with the task dependency graph generation enabled, the result is depicted in Figure 51. It can be identified that PyCOMPSs has been able to analyse the three given proteins in parallel.

The output of the application is a set of files within the output folder. It can be seen that the files decorated with *FILE_OUT* are stored in this folder. In particular, potential (`.xvg`) files represent the final results of the application, which can be visualized with GRACE.

```
user@login:~/lysozyme/output> ls -l
total 79411
```
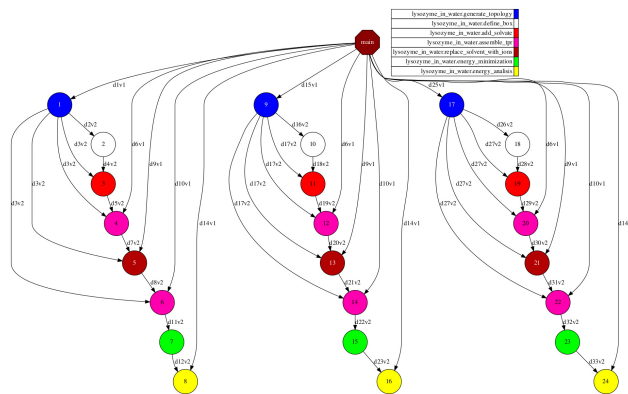
Figure 51: Python Lysozyme in Water tasks graph

```
-rw-r--r-- 1 user group    8976 may 19 17:06 1aki_em_energy.edr
-rw-r--r-- 1 user group 1280044 may 19 17:03 1aki_em.tpr
-rw-r--r-- 1 user group   88246 may 19 17:03 1aki.gro
-rw-r--r-- 1 user group 1279304 may 19 17:03 1aki_ions.tpr
-rw-r--r-- 1 user group   88246 may 19 17:03 1aki_newbox.gro
-rw-r--r-- 1 user group    2141 may 19 17:06 1aki_potential.xvg  <-------
-rw-r--r-- 1 user group 1525186 may 19 17:03 1aki_solv.gro
-rw-r--r-- 1 user group 1524475 may 19 17:03 1aki_solv_ions.gro
-rw-r--r-- 1 user group  577616 may 19 17:03 1aki.top
-rw-r--r-- 1 user group  577570 ene 24 16:11 #1aki.top.1#
-rw-r--r-- 1 user group  577601 may 19 16:59 #1aki.top.10#
-rw-r--r-- 1 user group  577570 may 19 17:03 #1aki.top.11#
-rw-r--r-- 1 user group  577601 may 19 17:03 #1aki.top.12#
-rw-r--r-- 1 user group  577601 ene 24 16:11 #1aki.top.2#
-rw-r--r-- 1 user group  577570 ene 24 16:20 #1aki.top.3#
-rw-r--r-- 1 user group  577601 ene 24 16:20 #1aki.top.4#
-rw-r--r-- 1 user group  577570 ene 24 16:25 #1aki.top.5#
-rw-r--r-- 1 user group  577601 ene 24 16:25 #1aki.top.6#
-rw-r--r-- 1 user group  577570 ene 24 16:31 #1aki.top.7#
-rw-r--r-- 1 user group  577601 ene 24 16:31 #1aki.top.8#
-rw-r--r-- 1 user group  577570 may 19 16:59 #1aki.top.9#
-rw-r--r-- 1 user group    8976 may 19 17:08 1u3m_em_energy.edr
-rw-r--r-- 1 user group 1416272 may 19 17:03 1u3m_em.tpr
-rw-r--r-- 1 user group   82046 may 19 17:03 1u3m.gro
-rw-r--r-- 1 user group 1415196 may 19 17:03 1u3m_ions.tpr
-rw-r--r-- 1 user group   82046 may 19 17:03 1u3m_newbox.gro
-rw-r--r-- 1 user group    2151 may 19 17:08 1u3m_potential.xvg  <-------
-rw-r--r-- 1 user group 1837046 may 19 17:03 1u3m_solv.gro
-rw-r--r-- 1 user group 1836965 may 19 17:03 1u3m_solv_ions.gro
-rw-r--r-- 1 user group  537950 may 19 17:03 1u3m.top
-rw-r--r-- 1 user group  537904 ene 24 16:11 #1u3m.top.1#
-rw-r--r-- 1 user group  537935 may 19 16:59 #1u3m.top.10#
-rw-r--r-- 1 user group  537904 may 19 17:03 #1u3m.top.11#
-rw-r--r-- 1 user group  537935 may 19 17:03 #1u3m.top.12#
-rw-r--r-- 1 user group  537935 ene 24 16:11 #1u3m.top.2#
-rw-r--r-- 1 user group  537904 ene 24 16:20 #1u3m.top.3#
-rw-r--r-- 1 user group  537935 ene 24 16:20 #1u3m.top.4#
-rw-r--r-- 1 user group  537904 ene 24 16:25 #1u3m.top.5#
-rw-r--r-- 1 user group  537935 ene 24 16:25 #1u3m.top.6#
```

```
-rw-r--r-- 1 user group  537904 ene 24 16:31 #1u3m.top.7#
-rw-r--r-- 1 user group  537935 ene 24 16:31 #1u3m.top.8#
-rw-r--r-- 1 user group  537904 may 19 16:59 #1u3m.top.9#
-rw-r--r-- 1 user group    8780 may 19 17:08 1xyw_em_energy.edr
-rw-r--r-- 1 user group 1408872 may 19 17:03 1xyw_em.tpr
-rw-r--r-- 1 user group   80112 may 19 17:03 1xyw.gro
-rw-r--r-- 1 user group 1407844 may 19 17:03 1xyw_ions.tpr
-rw-r--r-- 1 user group   80112 may 19 17:03 1xyw_newbox.gro
-rw-r--r-- 1 user group    2141 may 19 17:08 1xyw_potential.xvg  <-------
-rw-r--r-- 1 user group 1845237 may 19 17:03 1xyw_solv.gro
-rw-r--r-- 1 user group 1845066 may 19 17:03 1xyw_solv_ions.gro
-rw-r--r-- 1 user group  524026 may 19 17:03 1xyw.top
-rw-r--r-- 1 user group  523980 ene 24 16:11 #1xyw.top.1#
-rw-r--r-- 1 user group  524011 may 19 16:59 #1xyw.top.10#
-rw-r--r-- 1 user group  523980 may 19 17:03 #1xyw.top.11#
-rw-r--r-- 1 user group  524011 may 19 17:03 #1xyw.top.12#
-rw-r--r-- 1 user group  524011 ene 24 16:11 #1xyw.top.2#
-rw-r--r-- 1 user group  523980 ene 24 16:20 #1xyw.top.3#
-rw-r--r-- 1 user group  524011 ene 24 16:20 #1xyw.top.4#
-rw-r--r-- 1 user group  523980 ene 24 16:25 #1xyw.top.5#
-rw-r--r-- 1 user group  524011 ene 24 16:25 #1xyw.top.6#
-rw-r--r-- 1 user group  523980 ene 24 16:31 #1xyw.top.7#
-rw-r--r-- 1 user group  524011 ene 24 16:31 #1xyw.top.8#
-rw-r--r-- 1 user group  523980 may 19 16:59 #1xyw.top.9#
```

Figure 52 depicts the potential results obtained for the 1xyw protein.
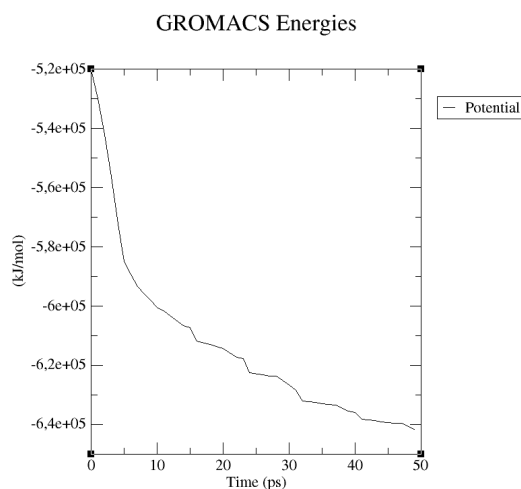
Figure 52: 1xyw Potential result (plotted with GRACE)

8.2.6 Persistent Storage

This section shows some sample applications using persistent storage.

8.2.6.1 Kmeans with dataClay

KMeans is machine-learning algorithm (NP-hard), popularly employed for cluster analysis in data mining, and interesting for benchmarking and performance evaluation.

The objective of the Kmeans algorithm to group a set of multidimensional points into a predefined number of clusters, in which each point belongs to the closest cluster (with the nearest mean distance), in an iterative process.

In this application we make use of the persistent storage API. In particular, the dataset fragments are considered `StorageObject`, delegating its content into the persistent framework. Since the data model (object declared as storage object) includes functions, it can run efficiently with dataClay.

First, lets see the data model (`storage_model/fragment.py`)

```python
from storage.api import StorageObject

try:
    from pycompss.api.task import task
    from pycompss.api.parameter import IN
except ImportError:
    # Required since the pycompss module is not ready during the registry
    from dataclay.contrib.dummy_pycompss import task, IN

from dataclay import dclayMethod

import numpy as np
from sklearn.metrics import pairwise_distances


class Fragment(StorageObject):
    """
    @ClassField points numpy.ndarray

    @dclayImport numpy as np
    @dclayImportFrom sklearn.metrics import pairwise_distances
    """
    @dclayMethod()
    def __init__(self):
        super(Fragment, self).__init__()
        self.points = None

    @dclayMethod(num_points='int', dim='int', mode='str', seed='int')
    def generate_points(self, num_points, dim, mode, seed):
        """
        Generate a random fragment of the specified number of points using the
        specified mode and the specified seed. Note that the generation is
        distributed (the master will never see the actual points).
        :param num_points: Number of points
        :param dim: Number of dimensions
        :param mode: Dataset generation mode
        :param seed: Random seed
        :return: Dataset fragment
        """
```

(continues on next page)

```python
        # Random generation distributions
        rand = {
            'normal': lambda k: np.random.normal(0, 1, k),
            'uniform': lambda k: np.random.random(k),
        }
        r = rand[mode]
        np.random.seed(seed)
        mat = np.asarray(
            [r(dim) for __ in range(num_points)]
        )
        # Normalize all points between 0 and 1
        mat -= np.min(mat)
        mx = np.max(mat)
        if mx > 0.0:
            mat /= mx

        self.points = mat

    @task(returns=np.ndarray, target_direction=IN)
    @dclayMethod(centres='numpy.ndarray', return_='anything')
    def partial_sum(self, centres):
        partials = np.zeros((centres.shape[0], 2), dtype=object)
        arr = self.points
        close_centres = pairwise_distances(arr, centres).argmin(axis=1)
        for center_idx, _ in enumerate(centres):
            indices = np.argwhere(close_centres == center_idx).flatten()
            partials[center_idx][0] = np.sum(arr[indices], axis=0)
            partials[center_idx][1] = indices.shape[0]
        return partials
```

Now we can focus in the main kmeans application (`kmeans.py`):

```python
import time
import numpy as np

from pycompss.api.task import task
from pycompss.api.api import compss_wait_on
from pycompss.api.api import compss_barrier

from storage_model.fragment import Fragment

from sklearn.metrics.pairwise import paired_distances


@task(returns=dict)
def merge(*data):
    accum = data[0].copy()
    for d in data[1:]:
        accum += d
    return accum


def converged(old_centres, centres, epsilon, iteration, max_iter):
    if old_centres is None:
        return False
    dist = np.sum(paired_distances(centres, old_centres))
```

```python
        return dist < epsilon ** 2 or iteration >= max_iter


def recompute_centres(partials, old_centres, arity):
    centres = old_centres.copy()
    while len(partials) > 1:
        partials_subset = partials[:arity]
        partials = partials[arity:]
        partials.append(merge(*partials_subset))
    partials = compss_wait_on(partials)
    for idx, sum_ in enumerate(partials[0]):
        if sum_[1] != 0:
            centres[idx] = sum_[0] / sum_[1]
    return centres


def kmeans_frag(fragments, dimensions, num_centres=10, iterations=20,
                seed=0., epsilon=1e-9, arity=50):
    """
    A fragment-based K-Means algorithm.
    Given a set of fragments (which can be either PSCOs or future objects that
    point to PSCOs), the desired number of clusters and the maximum number of
    iterations, compute the optimal centres and the index of the centre
    for each point.
    PSCO.mat must be a NxD float np.ndarray, where D = dimensions
    :param fragments: Number of fragments
    :param dimensions: Number of dimensions
    :param num_centres: Number of centres
    :param iterations: Maximum number of iterations
    :param seed: Random seed
    :param epsilon: Epsilon (convergence distance)
    :param arity: Arity
    :return: Final centres and labels
    """
    # Set the random seed
    np.random.seed(seed)
    # Centres is usually a very small matrix, so it is affordable to have it in
    # the master.
    centres = np.asarray(
        [np.random.random(dimensions) for _ in range(num_centres)]
    )
    # Note: this implementation treats the centres as files, never as PSCOs.
    old_centres = None
    iteration = 0
    while not converged(old_centres, centres, epsilon, iteration, iterations):
        print("Doing iteration #%d/%d" % (iteration + 1, iterations))
        old_centres = centres.copy()
        partials = []
        for frag in fragments:
            partial = frag.partial_sum(old_centres)
            partials.append(partial)
        centres = recompute_centres(partials, old_centres, arity)
        iteration += 1
    return centres
```

```python
def parse_arguments():
    """
    Parse command line arguments. Make the program generate
    a help message in case of wrong usage.
    :return: Parsed arguments
    """
    import argparse
    parser = argparse.ArgumentParser(description='KMeans Clustering.')
    parser.add_argument('-s', '--seed', type=int, default=0,
                        help='Pseudo-random seed. Default = 0')
    parser.add_argument('-n', '--numpoints', type=int, default=100,
                        help='Number of points. Default = 100')
    parser.add_argument('-d', '--dimensions', type=int, default=2,
                        help='Number of dimensions. Default = 2')
    parser.add_argument('-c', '--num_centres', type=int, default=5,
                        help='Number of centres. Default = 2')
    parser.add_argument('-f', '--fragments', type=int, default=10,
                        help='Number of fragments.' +
                             ' Default = 10. Condition: fragments < points')
    parser.add_argument('-m', '--mode', type=str, default='uniform',
                        choices=['uniform', 'normal'],
                        help='Distribution of points. Default = uniform')
    parser.add_argument('-i', '--iterations', type=int, default=20,
                        help='Maximum number of iterations')
    parser.add_argument('-e', '--epsilon', type=float, default=1e-9,
                        help='Epsilon. Kmeans will stop when:' +
                             ' |old - new| < epsilon.')
    parser.add_argument('-a', '--arity', type=int, default=50,
                        help='Arity of the reduction carried out during \
                        the computation of the new centroids')
    return parser.parse_args()


from storage_model.fragment import Fragment  # this will have to be removed


@task(returns=Fragment)
def generate_fragment(points, dim, mode, seed):
    """
    Generate a random fragment of the specified number of points using the
    specified mode and the specified seed. Note that the generation is
    distributed (the master will never see the actual points).
    :param points: Number of points
    :param dim: Number of dimensions
    :param mode: Dataset generation mode
    :param seed: Random seed
    :return: Dataset fragment
    """
    fragment = Fragment()
    # Make persistent before since it is populated in the task
    fragment.make_persistent()
    fragment.generate_points(points, dim, mode, seed)


def main(seed, numpoints, dimensions, num_centres, fragments, mode, iterations,
         epsilon, arity):
    """
    This will be executed if called as main script. Look at the kmeans_frag
```

```
    for the KMeans function.
    This code is used for experimental purposes.
    I.e it generates random data from some parameters that determine the size,
    dimensionality and etc and returns the elapsed time.
    :param seed: Random seed
    :param numpoints: Number of points
    :param dimensions: Number of dimensions
    :param num_centres: Number of centres
    :param fragments: Number of fragments
    :param mode: Dataset generation mode
    :param iterations: Number of iterations
    :param epsilon: Epsilon (convergence distance)
    :param arity: Arity
    :return: None
    """
    start_time = time.time()

    # Generate the data
    fragment_list = []
    # Prevent infinite loops in case of not-so-smart users
    points_per_fragment = max(1, numpoints // fragments)

    for l in range(0, numpoints, points_per_fragment):
        # Note that the seed is different for each fragment.
        # This is done to avoid having repeated data.
        r = min(numpoints, l + points_per_fragment)

        fragment_list.append(
            generate_fragment(r - l, dimensions, mode, seed + l)
        )

    compss_barrier()
    print("Generation/Load done")
    initialization_time = time.time()
    print("Starting kmeans")

    # Run kmeans
    centres = kmeans_frag(fragments=fragment_list,
                          dimensions=dimensions,
                          num_centres=num_centres,
                          iterations=iterations,
                          seed=seed,
                          epsilon=epsilon,
                          arity=arity)
    compss_barrier()
    print("Ending kmeans")
    kmeans_time = time.time()

    print("-----------------------------------------")
    print("-------------- RESULTS ------------------")
    print("-----------------------------------------")
    print("Initialization time: %f" % (initialization_time - start_time))
    print("Kmeans time: %f" % (kmeans_time - initialization_time))
    print("Total time: %f" % (kmeans_time - start_time))
    print("-----------------------------------------")
    centres = compss_wait_on(centres)
```

Chapter 8. Sample Applications

```python
    print("CENTRES:")
    print(centres)
    print("----------------------------------------")



if __name__ == "__main__":
    options = parse_arguments()
    main(**vars(options))
```

Tip: This code can work with Hecuba and Redis if the functions declared in the data model are declared outside the data model, and the kmeans application uses the `points` attribute explicitly.

Since this code is going to be executed with dataClay, it is be necessary to declare the `client.properties`, `session.properties` and `storage_props.cfg` files into the `dataClay_confs` with the following contents as example (more configuration options can be found in the dataClay manual):

client.properties

```
HOST=127.0.0.1
TCPPORT=11034
```

session.properties

```
Account=bsc_user
Password=bsc_user
StubsClasspath=./stubs
DataSets=hpc_dataset
DataSetForStore=hpc_dataset
DataClayClientConfig=./client.properties
```

storage_props.cfg

```
BACKENDS_PER_NODE=48
```

An example of the submission script that can be used in MareNostrum IV to launch this kmeans with PyCOMPSs and dataClay is:

```bash
#!/bin/bash -e

module load gcc/8.1.0
export COMPSS_PYTHON_VERSION=3-ML
module load COMPSs/3.1
module load mkl/2018.1
module load impi/2018.1
module load opencv/4.1.2
module load DATACLAY/2.4.dev

# Retrieve script arguments
job_dependency=${1:-None}
num_nodes=${2:-2}
execution_time=${3:-5}
tracing=${4:-false}
exec_file=${5:-$(pwd)/kmeans.py}

# Freeze storage_props into a temporal
# (allow submission of multiple executions with varying parameters)
STORAGE_PROPS=`mktemp -p ~`
```

```
cp $(pwd)/dataClay_confs/storage_props.cfg "${STORAGE_PROPS}"

if [[ ! ${tracing} == "false" ]]
then
  extra_tracing_flags="\
    --jvm_workers_opts=\"-javaagent:/apps/DATACLAY/dependencies/aspectjweaver.jar\" \
    --jvm_master_opts=\"-javaagent:/apps/DATACLAY/dependencies/aspectjweaver.jar\" \
  "
  echo "Adding DATACLAYSRV_START_CMD to storage properties file"
  echo "\${STORAGE_PROPS}=${STORAGE_PROPS}"
  echo "" >> ${STORAGE_PROPS}
  echo "DATACLAYSRV_START_CMD=\"--tracing\"" >> ${STORAGE_PROPS}
fi

# Define script variables
SCRIPT_DIR="$(cd "$(dirname "${BASH_SOURCE[0]}")" && pwd)"
WORK_DIR=${SCRIPT_DIR}/
APP_CLASSPATH=${SCRIPT_DIR}/
APP_PYTHONPATH=${SCRIPT_DIR}/

# Define application variables
graph=$tracing
log_level="off"
qos_flag="--qos=debug"
workers_flag=""
constraints="highmem"

CPUS_PER_NODE=48
WORKER_IN_MASTER=0

shift 5

# Those are evaluated at submit time, not at start time...
COMPSS_VERSION=`module load whatis COMPSs 2>&1 >/dev/null | awk '{print $1 ; exit}'`
DATACLAY_VERSION=`module load whatis DATACLAY 2>&1 >/dev/null | awk '{print $1 ; exit}'`

# Enqueue job
enqueue_compss \
  --job_name=kmeansOO_PyCOMPSs_dataClay \
  --job_dependency="${job_dependency}" \
  --exec_time="${execution_time}" \
  --num_nodes="${num_nodes}" \
  \
  --cpus_per_node="${CPUS_PER_NODE}" \
  --worker_in_master_cpus="${WORKER_IN_MASTER}" \
  --scheduler=es.bsc.compss.scheduler.loadbalancing.LoadBalancingScheduler \
  \
  "${workers_flag}" \
  \
  --worker_working_dir=/gpfs/scratch/user/ \
  \
  --constraints=${constraints} \
  --tracing="${tracing}" \
  --graph="${graph}" \
  --summary \
  --log_level="${log_level}" \
```

```
  "${qos_flag}" \
  \
  --classpath=${DATACLAY_JAR} \
  --pythonpath=${APP_PYTHONPATH}:${PYCLAY_PATH}:${PYTHONPATH} \
  --storage_props=${STORAGE_PROPS} \
  --storage_home=$COMPSS_STORAGE_HOME \
  --prolog="$DATACLAY_HOME/bin/dataclayprepare,$(pwd)/storage_model/,$(pwd)/,storage_model,
→python" \
  \
  ${extra_tracing_flags} \
  \
  --lang=python \
  \
  "$exec_file" $@ --use_storage
```

8.3 C/C++ Sample applications

The first two examples in this section are simple applications developed in COMPSs to easily illustrate how to code, compile and run COMPSs applications. These applications are executed locally and show different ways to take advantage of all the COMPSs features.

The rest of the examples are more elaborated and consider the execution in a cloud platform where the VMs mount a common storage on /**sharedDisk** directory. This is useful in the case of applications that require working with big files, allowing to transfer data only once, at the beginning of the execution, and to enable the application to access the data directly during the rest of the execution.

The Virtual Machine available at our webpage (http://compss.bsc.es/) provides a development environment with all the applications listed in the following sections. The codes of all the applications can be found under the /home/compss/tutorial_apps/c/ folder.

8.3.1 Simple

The Simple application is a C application that increases a counter by means of a task. The counter is stored inside a file that is transfered to the worker when the task is executed. Thus, the tasks inferface is defined as follows:

```
// simple.idl
interface simple {
    void increment(inout File filename);
};
```

Next we also provide the invocation of the task from the main code and the increment's method code.

```
// simple.cc

int main(int argc, char *argv[]) {
    // Check and get parameters
    if (argc != 2) {
    usage();
    return -1;
    }
    string initialValue = argv[1];
    file fileName = strdup(FILE_NAME);

    // Init compss
```

```cpp
    compss_on();

    // Write file
    ofstream fos (fileName);
    if (fos.is_open()) {
    fos << initialValue << endl;
    fos.close();
    } else {
    cerr << "[ERROR] Unable to open file" << endl;
    return -1;
    }
    cout << "Initial counter value is " << initialValue << endl;

    // Execute increment
    increment(&fileName);

    // Read new value
    string finalValue;
    ifstream fis;
    compss_ifstream(fileName, fis);
    if (fis.is_open()) {
    if (getline(fis, finalValue)) {
        cout << "Final counter value is " << finalValue << endl;
        fis.close();
    } else {
        cerr << "[ERROR] Unable to read final value" << endl;
        fis.close();
        return -1;
    }
    } else {
    cerr << "[ERROR] Unable to open file" << endl;
    return -1;
    }

    // Close COMPSs and end
    compss_off();
    return 0;
}
```

```cpp
//simple-functions.cc

void increment(file *fileName) {
    cout << "INIT TASK" << endl;
    cout << "Param: " << *fileName << endl;
    // Read value
    char initialValue;
    ifstream fis (*fileName);
    if (fis.is_open()) {
    if (fis >> initialValue) {
        fis.close();
    } else {
        cerr << "[ERROR] Unable to read final value" << endl;
        fis.close();
    }
    fis.close();
    } else {
```

Chapter 8. Sample Applications

```
    cerr << "[ERROR] Unable to open file" << endl;
    }

    // Increment
    cout << "INIT VALUE: " << initialValue << endl;
    int finalValue = ((int)(initialValue) - (int)('0')) + 1;
    cout << "FINAL VALUE: " << finalValue << endl;

    // Write new value
    ofstream fos (*fileName);
    if (fos.is_open()) {
    fos << finalValue << endl;
    fos.close();
    } else {
    cerr << "[ERROR] Unable to open file" << endl;
    }
    cout << "END TASK" << endl;
}
```

Finally, to compile and execute this application users must run the following commands:

```
compss@bsc:~$ cd ~/tutorial_apps/c/simple/
compss@bsc:~/tutorial_apps/c/simple$ compss_build_app simple
compss@bsc:~/tutorial_apps/c/simple$ runcompss --lang=c --project=./xml/project.xml --
→resources=./xml/resources.xml ~/tutorial_apps/c/simple/master/simple 1
[  INFO] Using default execution type: compss

----------------- Executing simple ------------------------

JVM_OPTIONS_FILE: /tmp/tmp.n2eZjgmDGo
COMPSS_HOME: /opt/COMPSs
Args: 1

WARNING: COMPSs Properties file is null. Setting default values
[(617)    API]  -  Starting COMPSs Runtime v<version>
Initial counter value is 1
[   BINDING]  -  @GS_register  -  Ref: 0x7fffa35d0f48
[   BINDING]  -  @GS_register  -  ENTRY ADDED
[   BINDING]  -  @GS_register  -  Entry.type: 9
[   BINDING]  -  @GS_register  -  Entry.classname: File
[   BINDING]  -  @GS_register  -  Entry.filename: counter
[   BINDING]  -  @GS_register  -  setting filename: counter
[   BINDING]  -  @GS_register  -  Filename: counter
[   BINDING]  -  @GS_register  - Result is 0
[   BINDING]  -  @compss_wait_on  -  Entry.type: 9
[   BINDING]  -  @compss_wait_on  -  Entry.classname: File
[   BINDING]  -  @compss_wait_on  -  Entry.filename: counter
[   BINDING]  -  @compss_wait_on  -  Runtime filename: /home/compss/.COMPSs/simple_01/
→tmpFiles/d1v2_1479141705574.IT
[   BINDING]  -  @compss_wait_on  -  File renaming: /home/compss/.COMPSs/simple_01/tmpFiles/
→d1v2_1479141705574.IT to counter
Final counter value is 2
[(3755)    API]  -  Execution Finished

------------------------------------------------------------
```

8.3.2 Increment

The Increment application is a C application that increases N times three different counters. Each increase step is developed by a separated task. The purpose of this application is to show parallelism between the three counters.

Next we provide the main code of this application. The code inside the *increment* task is the same than the previous example.

```
// increment.cc

int main(int argc, char *argv[]) {
    // Check and get parameters
    if (argc != 5) {
    usage();
    return -1;
    }
    int N = atoi( argv[1] );
    string counter1 = argv[2];
    string counter2 = argv[3];
    string counter3 = argv[4];

    // Init COMPSs
    compss_on();

    // Initialize counter files
    file fileName1 = strdup(FILE_NAME1);
    file fileName2 = strdup(FILE_NAME2);
    file fileName3 = strdup(FILE_NAME3);
    initializeCounters(counter1, counter2, counter3, fileName1, fileName2, fileName3);

    // Print initial counters state
    cout << "Initial counter values: " << endl;
    printCounterValues(fileName1, fileName2, fileName3);

    // Execute increment tasks
    for (int i = 0; i < N; ++i) {
    increment(&fileName1);
    increment(&fileName2);
    increment(&fileName3);
    }

    // Print final state
    cout << "Final counter values: " << endl;
    printCounterValues(fileName1, fileName2, fileName3);

    // Stop COMPSs
    compss_off();

    return 0;
}
```

As shown in the main code, this application has 4 parameters that stand for:

1. **N:** Number of times to increase a counter
2. **counter1:** Initial value for counter 1
3. **counter2:** Initial value for counter 2
4. **counter3:** Initial value for counter 3

Next we will compile and run the Increment application with the *-g* option to be able to generate the final graph at the end of the execution.

```
compss@bsc:~$ cd ~/tutorial_apps/c/increment/
compss@bsc:~/tutorial_apps/c/increment$ compss_build_app increment
compss@bsc:~/tutorial_apps/c/increment$ runcompss --lang=c -g --project=./xml/project.xml --
→resources=./xml/resources.xml ~/tutorial_apps/c/increment/master/increment 10 1 2 3
[  INFO] Using default execution type: compss


----------------- Executing increment -------------------------


JVM_OPTIONS_FILE: /tmp/tmp.mgCheFd3kL
COMPSS_HOME: /opt/COMPSs
Args: 10 1 2 3


WARNING: COMPSs Properties file is null. Setting default values
[(655)    API]  -  Starting COMPSs Runtime v<version>
Initial counter values:
- Counter1 value is 1
- Counter2 value is 2
- Counter3 value is 3
[   BINDING]  -  @GS_register  -  Ref: 0x7ffea17719f0
[   BINDING]  -  @GS_register  -  ENTRY ADDED
[   BINDING]  -  @GS_register  -  Entry.type: 9
[   BINDING]  -  @GS_register  -  Entry.classname: File
[   BINDING]  -  @GS_register  -  Entry.filename: file1.txt
[   BINDING]  -  @GS_register  -  setting filename: file1.txt
[   BINDING]  -  @GS_register  -  Filename: file1.txt
[   BINDING]  -  @GS_register  - Result is 0
[   BINDING]  -  @GS_register  -  Ref: 0x7ffea17719f8
[   BINDING]  -  @GS_register  -  ENTRY ADDED
[   BINDING]  -  @GS_register  -  Entry.type: 9
[   BINDING]  -  @GS_register  -  Entry.classname: File
[   BINDING]  -  @GS_register  -  Entry.filename: file2.txt
[   BINDING]  -  @GS_register  -  setting filename: file2.txt
[   BINDING]  -  @GS_register  -  Filename: file2.txt
[   BINDING]  -  @GS_register  - Result is 0
[   BINDING]  -  @GS_register  -  Ref: 0x7ffea1771a00
[   BINDING]  -  @GS_register  -  ENTRY ADDED
[   BINDING]  -  @GS_register  -  Entry.type: 9
[   BINDING]  -  @GS_register  -  Entry.classname: File
[   BINDING]  -  @GS_register  -  Entry.filename: file3.txt
[   BINDING]  -  @GS_register  -  setting filename: file3.txt
[   BINDING]  -  @GS_register  -  Filename: file3.txt
[   BINDING]  -  @GS_register  - Result is 0
[   BINDING]  -  @GS_register  -  Ref: 0x7ffea17719f0
[   BINDING]  -  @GS_register  -  ENTRY FOUND
[   BINDING]  -  @GS_register  -  Entry.type: 9
[   BINDING]  -  @GS_register  -  Entry.classname: File
[   BINDING]  -  @GS_register  -  Entry.filename: file1.txt
[   BINDING]  -  @GS_register  -  setting filename: file1.txt
[   BINDING]  -  @GS_register  -  Filename: file1.txt
[   BINDING]  -  @GS_register  - Result is 0
[   BINDING]  -  @GS_register  -  Ref: 0x7ffea17719f8
[   BINDING]  -  @GS_register  -  ENTRY FOUND
[   BINDING]  -  @GS_register  -  Entry.type: 9
[   BINDING]  -  @GS_register  -  Entry.classname: File
[   BINDING]  -  @GS_register  -  Entry.filename: file2.txt
[   BINDING]  -  @GS_register  -  setting filename: file2.txt
[   BINDING]  -  @GS_register  -  Filename: file2.txt
```

(continues on next page)

```
[   BINDING]  -  @GS_register  - Result is 0
[   BINDING]  -  @GS_register  -  Ref: 0x7ffea1771a00
[   BINDING]  -  @GS_register  -  ENTRY FOUND
[   BINDING]  -  @GS_register  -  Entry.type: 9
[   BINDING]  -  @GS_register  -  Entry.classname: File
[   BINDING]  -  @GS_register  -  Entry.filename: file3.txt
[   BINDING]  -  @GS_register  -  setting filename: file3.txt
[   BINDING]  -  @GS_register  -  Filename: file3.txt
[   BINDING]  -  @GS_register  - Result is 0
[   BINDING]  -  @GS_register  -  Ref: 0x7ffea17719f0
[   BINDING]  -  @GS_register  -  ENTRY FOUND
[   BINDING]  -  @GS_register  -  Entry.type: 9
[   BINDING]  -  @GS_register  -  Entry.classname: File
[   BINDING]  -  @GS_register  -  Entry.filename: file1.txt
[   BINDING]  -  @GS_register  -  setting filename: file1.txt
[   BINDING]  -  @GS_register  -  Filename: file1.txt
[   BINDING]  -  @GS_register  - Result is 0
[   BINDING]  -  @GS_register  -  Ref: 0x7ffea17719f8
[   BINDING]  -  @GS_register  -  ENTRY FOUND
[   BINDING]  -  @GS_register  -  Entry.type: 9
[   BINDING]  -  @GS_register  -  Entry.classname: File
[   BINDING]  -  @GS_register  -  Entry.filename: file2.txt
[   BINDING]  -  @GS_register  -  setting filename: file2.txt
[   BINDING]  -  @GS_register  -  Filename: file2.txt
[   BINDING]  -  @GS_register  - Result is 0
[   BINDING]  -  @GS_register  -  Ref: 0x7ffea1771a00
[   BINDING]  -  @GS_register  -  ENTRY FOUND
[   BINDING]  -  @GS_register  -  Entry.type: 9
[   BINDING]  -  @GS_register  -  Entry.classname: File
[   BINDING]  -  @GS_register  -  Entry.filename: file3.txt
[   BINDING]  -  @GS_register  -  setting filename: file3.txt
[   BINDING]  -  @GS_register  -  Filename: file3.txt
[   BINDING]  -  @GS_register  - Result is 0
[   BINDING]  -  @GS_register  -  Ref: 0x7ffea17719f0
[   BINDING]  -  @GS_register  -  ENTRY FOUND
[   BINDING]  -  @GS_register  -  Entry.type: 9
[   BINDING]  -  @GS_register  -  Entry.classname: File
[   BINDING]  -  @GS_register  -  Entry.filename: file1.txt
[   BINDING]  -  @GS_register  -  setting filename: file1.txt
[   BINDING]  -  @GS_register  -  Filename: file1.txt
[   BINDING]  -  @GS_register  - Result is 0
[   BINDING]  -  @GS_register  -  Ref: 0x7ffea17719f8
[   BINDING]  -  @GS_register  -  ENTRY FOUND
[   BINDING]  -  @GS_register  -  Entry.type: 9
[   BINDING]  -  @GS_register  -  Entry.classname: File
[   BINDING]  -  @GS_register  -  Entry.filename: file2.txt
[   BINDING]  -  @GS_register  -  setting filename: file2.txt
[   BINDING]  -  @GS_register  -  Filename: file2.txt
[   BINDING]  -  @GS_register  - Result is 0
[   BINDING]  -  @GS_register  -  Ref: 0x7ffea1771a00
[   BINDING]  -  @GS_register  -  ENTRY FOUND
[   BINDING]  -  @GS_register  -  Entry.type: 9
[   BINDING]  -  @GS_register  -  Entry.classname: File
[   BINDING]  -  @GS_register  -  Entry.filename: file3.txt
[   BINDING]  -  @GS_register  -  setting filename: file3.txt
[   BINDING]  -  @GS_register  -  Filename: file3.txt
```

Chapter 8. Sample Applications

```
[   BINDING]   -   @GS_register   - Result is 0
[   BINDING]   -   @GS_register   -  Ref: 0x7ffea17719f0
[   BINDING]   -   @GS_register   -  ENTRY FOUND
[   BINDING]   -   @GS_register   -  Entry.type: 9
[   BINDING]   -   @GS_register   -  Entry.classname: File
[   BINDING]   -   @GS_register   -  Entry.filename: file1.txt
[   BINDING]   -   @GS_register   -  setting filename: file1.txt
[   BINDING]   -   @GS_register   -  Filename: file1.txt
[   BINDING]   -   @GS_register   - Result is 0
[   BINDING]   -   @GS_register   -  Ref: 0x7ffea17719f8
[   BINDING]   -   @GS_register   -  ENTRY FOUND
[   BINDING]   -   @GS_register   -  Entry.type: 9
[   BINDING]   -   @GS_register   -  Entry.classname: File
[   BINDING]   -   @GS_register   -  Entry.filename: file2.txt
[   BINDING]   -   @GS_register   -  setting filename: file2.txt
[   BINDING]   -   @GS_register   -  Filename: file2.txt
[   BINDING]   -   @GS_register   - Result is 0
[   BINDING]   -   @GS_register   -  Ref: 0x7ffea1771a00
[   BINDING]   -   @GS_register   -  ENTRY FOUND
[   BINDING]   -   @GS_register   -  Entry.type: 9
[   BINDING]   -   @GS_register   -  Entry.classname: File
[   BINDING]   -   @GS_register   -  Entry.filename: file3.txt
[   BINDING]   -   @GS_register   -  setting filename: file3.txt
[   BINDING]   -   @GS_register   -  Filename: file3.txt
[   BINDING]   -   @GS_register   - Result is 0
[   BINDING]   -   @GS_register   -  Ref: 0x7ffea17719f0
[   BINDING]   -   @GS_register   -  ENTRY FOUND
[   BINDING]   -   @GS_register   -  Entry.type: 9
[   BINDING]   -   @GS_register   -  Entry.classname: File
[   BINDING]   -   @GS_register   -  Entry.filename: file1.txt
[   BINDING]   -   @GS_register   -  setting filename: file1.txt
[   BINDING]   -   @GS_register   -  Filename: file1.txt
[   BINDING]   -   @GS_register   - Result is 0
[   BINDING]   -   @GS_register   -  Ref: 0x7ffea17719f8
[   BINDING]   -   @GS_register   -  ENTRY FOUND
[   BINDING]   -   @GS_register   -  Entry.type: 9
[   BINDING]   -   @GS_register   -  Entry.classname: File
[   BINDING]   -   @GS_register   -  Entry.filename: file2.txt
[   BINDING]   -   @GS_register   -  setting filename: file2.txt
[   BINDING]   -   @GS_register   -  Filename: file2.txt
[   BINDING]   -   @GS_register   - Result is 0
[   BINDING]   -   @GS_register   -  Ref: 0x7ffea1771a00
[   BINDING]   -   @GS_register   -  ENTRY FOUND
[   BINDING]   -   @GS_register   -  Entry.type: 9
[   BINDING]   -   @GS_register   -  Entry.classname: File
[   BINDING]   -   @GS_register   -  Entry.filename: file3.txt
[   BINDING]   -   @GS_register   -  setting filename: file3.txt
[   BINDING]   -   @GS_register   -  Filename: file3.txt
[   BINDING]   -   @GS_register   - Result is 0
[   BINDING]   -   @GS_register   -  Ref: 0x7ffea17719f0
[   BINDING]   -   @GS_register   -  ENTRY FOUND
[   BINDING]   -   @GS_register   -  Entry.type: 9
[   BINDING]   -   @GS_register   -  Entry.classname: File
[   BINDING]   -   @GS_register   -  Entry.filename: file1.txt
[   BINDING]   -   @GS_register   -  setting filename: file1.txt
[   BINDING]   -   @GS_register   -  Filename: file1.txt
```

```
[   BINDING]  -  @GS_register  - Result is 0
[   BINDING]  -  @GS_register  -  Ref: 0x7ffea17719f8
[   BINDING]  -  @GS_register  -  ENTRY FOUND
[   BINDING]  -  @GS_register  -  Entry.type: 9
[   BINDING]  -  @GS_register  -  Entry.classname: File
[   BINDING]  -  @GS_register  -  Entry.filename: file2.txt
[   BINDING]  -  @GS_register  -  setting filename: file2.txt
[   BINDING]  -  @GS_register  -  Filename: file2.txt
[   BINDING]  -  @GS_register  - Result is 0
[   BINDING]  -  @GS_register  -  Ref: 0x7ffea1771a00
[   BINDING]  -  @GS_register  -  ENTRY FOUND
[   BINDING]  -  @GS_register  -  Entry.type: 9
[   BINDING]  -  @GS_register  -  Entry.classname: File
[   BINDING]  -  @GS_register  -  Entry.filename: file3.txt
[   BINDING]  -  @GS_register  -  setting filename: file3.txt
[   BINDING]  -  @GS_register  -  Filename: file3.txt
[   BINDING]  -  @GS_register  - Result is 0
[   BINDING]  -  @GS_register  -  Ref: 0x7ffea17719f0
[   BINDING]  -  @GS_register  -  ENTRY FOUND
[   BINDING]  -  @GS_register  -  Entry.type: 9
[   BINDING]  -  @GS_register  -  Entry.classname: File
[   BINDING]  -  @GS_register  -  Entry.filename: file1.txt
[   BINDING]  -  @GS_register  -  setting filename: file1.txt
[   BINDING]  -  @GS_register  -  Filename: file1.txt
[   BINDING]  -  @GS_register  - Result is 0
[   BINDING]  -  @GS_register  -  Ref: 0x7ffea17719f8
[   BINDING]  -  @GS_register  -  ENTRY FOUND
[   BINDING]  -  @GS_register  -  Entry.type: 9
[   BINDING]  -  @GS_register  -  Entry.classname: File
[   BINDING]  -  @GS_register  -  Entry.filename: file2.txt
[   BINDING]  -  @GS_register  -  setting filename: file2.txt
[   BINDING]  -  @GS_register  -  Filename: file2.txt
[   BINDING]  -  @GS_register  - Result is 0
[   BINDING]  -  @GS_register  -  Ref: 0x7ffea1771a00
[   BINDING]  -  @GS_register  -  ENTRY FOUND
[   BINDING]  -  @GS_register  -  Entry.type: 9
[   BINDING]  -  @GS_register  -  Entry.classname: File
[   BINDING]  -  @GS_register  -  Entry.filename: file3.txt
[   BINDING]  -  @GS_register  -  setting filename: file3.txt
[   BINDING]  -  @GS_register  -  Filename: file3.txt
[   BINDING]  -  @GS_register  - Result is 0
[   BINDING]  -  @GS_register  -  Ref: 0x7ffea17719f0
[   BINDING]  -  @GS_register  -  ENTRY FOUND
[   BINDING]  -  @GS_register  -  Entry.type: 9
[   BINDING]  -  @GS_register  -  Entry.classname: File
[   BINDING]  -  @GS_register  -  Entry.filename: file1.txt
[   BINDING]  -  @GS_register  -  setting filename: file1.txt
[   BINDING]  -  @GS_register  -  Filename: file1.txt
[   BINDING]  -  @GS_register  - Result is 0
[   BINDING]  -  @GS_register  -  Ref: 0x7ffea17719f8
[   BINDING]  -  @GS_register  -  ENTRY FOUND
[   BINDING]  -  @GS_register  -  Entry.type: 9
[   BINDING]  -  @GS_register  -  Entry.classname: File
[   BINDING]  -  @GS_register  -  Entry.filename: file2.txt
[   BINDING]  -  @GS_register  -  setting filename: file2.txt
[   BINDING]  -  @GS_register  -  Filename: file2.txt
```

```
[  BINDING]  -  @GS_register  - Result is 0
[  BINDING]  -  @GS_register  -  Ref: 0x7ffea1771a00
[  BINDING]  -  @GS_register  -  ENTRY FOUND
[  BINDING]  -  @GS_register  -  Entry.type: 9
[  BINDING]  -  @GS_register  -  Entry.classname: File
[  BINDING]  -  @GS_register  -  Entry.filename: file3.txt
[  BINDING]  -  @GS_register  -  setting filename: file3.txt
[  BINDING]  -  @GS_register  -  Filename: file3.txt
[  BINDING]  -  @GS_register  - Result is 0
[  BINDING]  -  @GS_register  -  Ref: 0x7ffea17719f0
[  BINDING]  -  @GS_register  -  ENTRY FOUND
[  BINDING]  -  @GS_register  -  Entry.type: 9
[  BINDING]  -  @GS_register  -  Entry.classname: File
[  BINDING]  -  @GS_register  -  Entry.filename: file1.txt
[  BINDING]  -  @GS_register  -  setting filename: file1.txt
[  BINDING]  -  @GS_register  -  Filename: file1.txt
[  BINDING]  -  @GS_register  - Result is 0
[  BINDING]  -  @GS_register  -  Ref: 0x7ffea17719f8
[  BINDING]  -  @GS_register  -  ENTRY FOUND
[  BINDING]  -  @GS_register  -  Entry.type: 9
[  BINDING]  -  @GS_register  -  Entry.classname: File
[  BINDING]  -  @GS_register  -  Entry.filename: file2.txt
[  BINDING]  -  @GS_register  -  setting filename: file2.txt
[  BINDING]  -  @GS_register  -  Filename: file2.txt
[  BINDING]  -  @GS_register  - Result is 0
[  BINDING]  -  @GS_register  -  Ref: 0x7ffea1771a00
[  BINDING]  -  @GS_register  -  ENTRY FOUND
[  BINDING]  -  @GS_register  -  Entry.type: 9
[  BINDING]  -  @GS_register  -  Entry.classname: File
[  BINDING]  -  @GS_register  -  Entry.filename: file3.txt
[  BINDING]  -  @GS_register  -  setting filename: file3.txt
[  BINDING]  -  @GS_register  -  Filename: file3.txt
[  BINDING]  -  @GS_register  - Result is 0
[  BINDING]  -  @compss_wait_on  -  Entry.type: 9
[  BINDING]  -  @compss_wait_on  -  Entry.classname: File
[  BINDING]  -  @compss_wait_on  -  Entry.filename: file1.txt
[  BINDING]  -  @compss_wait_on  -  Runtime filename: /home/compss/.COMPSs/increment_01/
↪tmpFiles/d1v11_1479142004112.IT
[  BINDING]  -  @compss_wait_on  -  File renaming: /home/compss/.COMPSs/increment_01/
↪tmpFiles/d1v11_1479142004112.IT to file1.txt
[  BINDING]  -  @compss_wait_on  -  Entry.type: 9
[  BINDING]  -  @compss_wait_on  -  Entry.classname: File
[  BINDING]  -  @compss_wait_on  -  Entry.filename: file2.txt
[  BINDING]  -  @compss_wait_on  -  Runtime filename: /home/compss/.COMPSs/increment_01/
↪tmpFiles/d2v11_1479142004112.IT
[  BINDING]  -  @compss_wait_on  -  File renaming: /home/compss/.COMPSs/increment_01/
↪tmpFiles/d2v11_1479142004112.IT to file2.txt
[  BINDING]  -  @compss_wait_on  -  Entry.type: 9
[  BINDING]  -  @compss_wait_on  -  Entry.classname: File
[  BINDING]  -  @compss_wait_on  -  Entry.filename: file3.txt
[  BINDING]  -  @compss_wait_on  -  Runtime filename: /home/compss/.COMPSs/increment_01/
↪tmpFiles/d3v11_1479142004112.IT
[  BINDING]  -  @compss_wait_on  -  File renaming: /home/compss/.COMPSs/increment_01/
↪tmpFiles/d3v11_1479142004112.IT to file3.txt
Final counter values:
- Counter1 value is 2
```

```
- Counter2 value is 3
- Counter3 value is 4
[(4288)    API] - Execution Finished


-----------------------------------------------------------
```

By running the *compss_gengraph* command users can obtain the task graph of the above execution. Next we provide the set of commands to obtain the graph show in Figure 53.

```
compss@bsc:~$ cd ~/.COMPSs/increment_01/monitor/
compss@bsc:~/.COMPSs/increment_01/monitor$ compss_gengraph complete_graph.dot
compss@bsc:~/.COMPSs/increment_01/monitor$ evince complete_graph.pdf
```
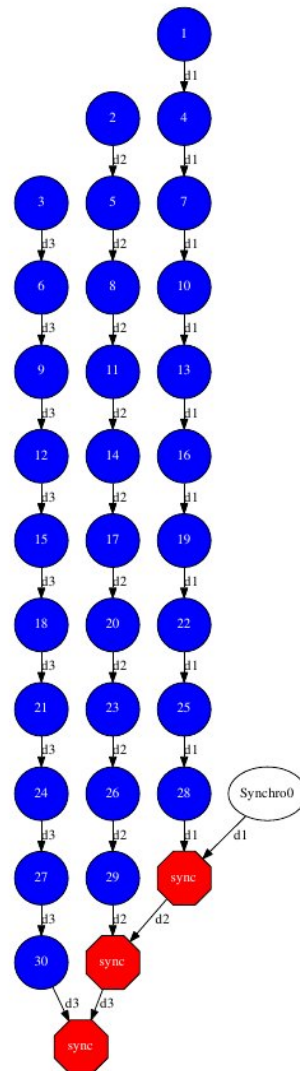

Figure 53: C increment tasks graph

Chapter 9

PyCOMPSs CLI

The PyCOMPSs CLI (`pycompss-cli`) provides a standalone tool to use PyCOMPSs interactively within docker environments, local machines and remote clusters. This tool has been implemented on top of the PyCOMPSs programming model, and it is being developed by the Workflows and Distributed Computing group of the Barcelona Supercomputing Center, and can be easily downloaded and installed from the Pypi repository.

9.1 Requirements and Installation

9.1.1 Requirements

- Python 3

9.1.1.1 Optional Requirements

- docker >= 17.12.0-ce
- docker package for Python

9.1.2 Installation

1. Install pycompss-cli:
   Since the PyCOMPSs CLI package is available in Pypi, it can be easly installed with `pip` as follows:

   ```
   $ python3 -m pip install pycompss-cli
   ```

2. Check the pycompss-cli installation:
   In order to check that it is correctly installed, check that the pycompss-cli executables (`pycompss`, `compss` and `dislib`, which can be used indiferently) are available from your command line.

   ```
   $ pycompss
   [PyCOMPSs CLI options will be shown]
   ```

Installing docker is optional and it's only required for running and deploying Docker type environments.

Unix

2. Install Docker (continue with step 3 if already installed):
   2.1. Suggested Docker installation instructions:
   - Docker for Mac. Or, if you prefer to use Homebrew.
   - Docker for Ubuntu.

- [Docker for Arch Linux](#).
  Be aware that for some distributions the Docker package has been renamed from `docker` to `docker-ce`. Make sure you install the new package.
  2.2. Add user to docker group to run the containers as a non-root user:
  - [Instructions](#)
  2.3. Check that docker is correctly installed:

```
$ docker --version
$ docker ps # this should be empty as no docker processes are yet running.
```

3. Install [docker](#) for python:

```
$ python3 -m pip install docker
```

Tip: Some Linux distributions do not include the `$HOME/.local/bin` folder in the `PATH` environment variable, preventing to access to the `pycompss-cli` commands (and any other Python packages installed in the user HOME).

If you experience that the `pycompss`| `compss`| `dislib` command is not available after the installation, you may need to include the following line into your `.bashrc` and execute it in your current session:

```
$ export PATH=${HOME}/.local/bin:${PATH}
```

Windows

1. Install Docker (continue with step 2 if already installed):
   2.1. Suggested Docker installation instructions:
   - [Docker for Windows](#).
   2.2. Check that docker is correctly installed:

```
$ docker --version
$ docker ps # this should be empty as no docker processes are yet running.
  ↪
```

2. Install [docker-py](#) for python:

```
$  conda install -c conda-forge/label/cf201901 docker-py
```

9.2 Usage

`pycompss-cli` provides the `pycompss` command line tool (`compss` and `dislib` are also alternatives to `pycompss`).

This command line tool enables to deploy and manage multiple COMPSs infrastructures from a single place and for 3 different types of environments (`docker`, `local` and `remote`)

The supported flags are:

```
$ pycompss
PyCOMPSs|COMPSS CLI:

Usage: pycompss COMMAND  |  compss COMMAND  |  dislib COMMAND

Available commands:
    init -n [NAME]:                            initialize COMPSs environment (default local).
                                               If -n is set it will initialize with NAME as␣
↪name or else with a random id.
    environment:                               lists, switch a remove COMPSs environments.
    exec CMD:                                  executes the CMD within the current COMPSs␣
↪environment.
```

(continues on next page)

```
    run [--app_name] [OPTIONS] FILE [PARAMS]:    runs FILE with COMPSs, where OPTIONS are␣
→COMPSs options and PARAMS are application parameters.
                                               --app_name parameter is only required for␣
→remote environments
    monitor [start|stop]:                      starts or stops the COMPSs monitoring.
    jupyter [--app_name] [PATH|FILE]:          starts jupyter-notebook in the given PATH or␣
→FILE.
                                               --app_name parameter is only required for␣
→remote environments
    job:                                       submits, cancel and list jobs on remote and␣
→local environments.
    app:                                       deploy, list and remove applications on␣
→remote and local environments.
    gengraph [FILE.dot]:                       converts the .dot graph into .pdf
    components list:                           lists COMPSs actives components.
    components add RESOURCE:                    adds the RESOURCE to the pool of workers of␣
→the COMPSs.
        Example given: pycompss components add worker 2 # to add 2 local workers.
        Example given: pycompss components add worker <IP>:<CORES> # to add a remote worker
                 Note: compss and dislib can be used instead of pycompss in both examples.
    components remove RESOURCE:    removes the RESOURCE to the pool of workers of the COMPSs.
        Example given: pycompss components remove worker 2 # to remove 2 local workers.
        Example given: pycompss components remove worker <IP>:<CORES> # to remove a remote␣
→worker
                 Note: compss and dislib can be used instead of pycompss in both examples.
```

9.2.1 Create a new COMPSs environment in your development directory

Docker

Local

Remote

Creates a docker type evironment and deploy a COMPSs container

```
$ pycompss init docker -w [WORK_DIR] -i [IMAGE]
```

The command initializes COMPSs in the current working dir or in WORK_DIR if -w is set. The COMPSs docker image to be used can be specified with -i (it can also be specified with the COMPSS_DOCKER_IMAGE environment variable).

Initialize the COMPSs infrastructure where your source code will be. This will allow docker to access your local code and run it inside the container.

```
$ pycompss init docker   # operates on the current directory as working directory.
```

Note: The first time needs to download the docker image from the repository, and it may take a while.

Alternatively, you can specify the working directory, the COMPSs docker image to use, or both at the same time:

```
$ # You can also provide a path
$ pycompss init docker -w /home/user/replace/path/
$
$ # Or the COMPSs docker image to use
$ pycompss init docker -i compss/compss-tutorial:3.0
```

```
$
$ # Or both
$ pycompss init docker -w /home/user/replace/path/ -i compss/compss-tutorial:3.0
```

```
$ pycompss init local -w [WORK_DIR] -m [MODULES ...]
```

Creates a local type evironment and initializes COMPSs in the current working dir or in WORK_DIR if -w is set. The modules to be loaded automatically can be specified with -m.

Initialize the COMPSs infrastructure where your source code will be.

```
$ pycompss init local   # operates on the current directory as working directory.
```

Alternatively, you can specify the working directory, the modules to automatically load or both at the same time:

```
$ # You can also provide a path
$ pycompss init local -w /home/user/replace/path/
$
$ # Or a list of modules to load automatically before every command
$ pycompss init local -m COMPSs/3.0 ANACONDA/5.1.0_py3
$
$ # Or both
$ pycompss init local -w /home/user/replace/path/ -m COMPSs/3.0 ANACONDA/5.1.0_py3
```

```
$ pycompss init remote -l [LOGIN] -m [FILE | MODULES ...]
```

Creates a remote type evironment with the credentials specified in LOGIN. The modules to be loaded automatically can be specified with -m.

Parameter LOGIN is necessary to connect to the remote host and must follow standard format i.e. [user]@[hostname]:[port]. `port` is optional and defaults to 22 for ssh.

```
$ pycompss init remote -l username@mn1.bsc.es
$
$ # Or with list of modules
$ pycompss init remote -l username@mn1.bsc.es -m COMPSs/3.0 ANACONDA/5.1.0_py3
```

Note: The SSH access to the remote should be configured to work without password. If you need to set up your machine for the first time please take a look at *Additional Configuration* Section for a detailed description of the additional configuration.

The parameter `-m` also supports passing a file containing not only modules but any kind of commands that you need to execute for the remote environment. Suppose we have a file `modules.sh` with the following content:

```
export ComputingUnits=1
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
module load COMPSs/3.0
module load ANACONDA/5.1.0_py3
```

```
$ pycompss init remote -l username@mn1.bsc.es -m /path/to/modules.sh
```

9.2.2 Managing environments

Every time command `pycompss init` is executed, a new environment is created and becomes the active environment in wich the rest of the commands will be executed. The subcommands `pycompss environment` will help inspecting, removing and switching between the environments.

You can list all the environments created with `pycompss environment list` and inspect which one is active, the types of each one and the ID.

```
$ pycompss environment list
                  ID            Type          Active
    -    5eeb858c2b10          remote            *
    -        default           local
    -   container-b54          docker
```

The ID of the environments is what you will use to switch between them.

```
$ pycompss environment change container-b54
     Environment `container-b54` is now active
```

Every environment can also be deleted, except `default` environment.

```
$ pycompss environment remove container-b54
     Deleting environment `container-b54`...
$ pycompss environment remove default
     ERROR: `default` environment is required and cannot be deleted.
```

Also every remote environment can have multiple applications deployed in remote. So if you want to delete the environment all the data associated with them will be aslo deleted.

```
$ pycompss environment remove 5eeb858c2b10        # deleting a remote env with 2 apps deployed
     WARNING: There are still applications binded to this environment
     Do you want to delete this environment and all the applications? (y/N) y    # default is no
     Deleting app1...
     Deleting app2...
     Deleting environment `5eeb858c2b10`...
```

9.2.3 Deploying applications

For a remote environment is required to deploy any application before executing it.

```
$ pycompss app deploy [APP_NAME] --source_dir [SOURCE_DIR] --destination_dir [DESTINATION_DIR]
```

APP_NAME is required and must be unique. SOURCE_DIR and DESTINATION_DIR are optional the command copies the application from the current directory or from SOURCE_DIR if –source_dir is set to the remote directory specified with DESTINATION_DIR. if DESTINATION_DIR is not set, the application will be deployed in `$HOME/.COMPSsApps`

In order to show how to deploy an application, clone the PyCOMPSs' tutorial apps repository:

```
$ git clone https://github.com/bsc-wdc/tutorial_apps.git
```

Docker

Local

Remote

This is not necessary for docker environments since the working directory is set at the initialization of the environment.

On `local` environment deploying an application wil just copy the `--source_dir` directory to another location. Let's deploy the matrix multiplication tutorial application.

```
$ pycompss app deploy matmul --source_dir tutorial_apps/python/matmul_files
```

Also you could specify the path where to copy the files.

```
$ pycompss app deploy matmul --source_dir tutorial_apps/python/matmul_files/src/ --
→destination_dir /home/user/matmul_copy
```

If the parameter `--destination_dir` is missing then the files will be copied to `~/.COMPSsApps/%env_name%/%app_name%/`

Each deployed application can be listed using the command:

```
$ pycompss app list
    Name           Source                                                          Destination
    ------------   --------------------------------------------------------------  -----------
→---------------------------
    matmul         /home/user/tutorial_apps/python/matmul_files                    /home/user/
→.COMPSsApps/default/matmul
    test_jenkins   /jenkins/tests_execution_sandbox/apps/app009/.COMPSsWorker      /tmp/test_
→jenkins
```

Also every app can be deleted using the command:

```
$ pycompss app remove matmul
    Deleting application `matmul`...
```

> **Caution:** Removing an applocation will delete the copied app directory and every valuable results generated inside.

Let's deploy the matrix multiplication tutorial application.

```
$ pycompss app deploy matmul --source_dir tutorial_apps/python/matmul_files
```

Also you could specify the path where to copy the files on the remote host.

```
$ pycompss app deploy matmul --source_dir tutorial_apps/python/matmul_files/src/ --
→destination_dir /path/cluster/my_app
```

Each deployed application within a remote environment can be listed using the command:

```
$ pycompss app list
                Name
    -           matmul
    -           app1
```

Also every app can be deleted using the command:

```
$ pycompss app remove matmul
    Deleting application `matmul`...
```

> **Caution:** Removing an applocation will delete the entire app directory and every valuable results generated inside.

9.2.4 Executing applications

Docker

Local

Remote

Run application

Submit application execution (job) to queuing system

```
$ pycompss run [COMPSS_ARGS] APP_FILE [APP_ARGS]
```

APP_FILE is required and must be a valid python file. APP_ARGS is optional and can be used to pass any argument to the application.

```
1   --graph=<bool>, --graph, -g           Generation of the complete graph (true/false)
2                                         When no value is provided it is set to true
3                                         Default: false
4   --tracing=<level>, --tracing, -t      Set generation of traces and/or tracing level ( [␣
    ↪true | basic ] | advanced | scorep | arm-map | arm-ddt | false)
5                                         True and basic levels will produce the same traces.
6                                         When no value is provided it is set to 1
7                                         Default: 0
8   --monitoring=<int>, --monitoring, -m  Period between monitoring samples (milliseconds)
9                                         When no value is provided it is set to 2000
10                                        Default: 0
11  --external_debugger=<int>,
12  --external_debugger                   Enables external debugger connection on the specified␣
    ↪port (or 9999 if empty)
13                                        Default: false
14  --jmx_port=<int>                      Enable JVM profiling on specified port
15
16  Runtime configuration options:
17  --task_execution=<compss|storage>     Task execution under COMPSs or Storage.
18                                        Default: compss
19  --storage_impl=<string>               Path to an storage implementation. Shortcut to␣
    ↪setting pypath and classpath. See Runtime/storage in your installation folder.
20  --storage_conf=<path>                 Path to the storage configuration file
21                                        Default: null
22  --project=<path>                      Path to the project XML file
23                                        Default: /opt/COMPSs//Runtime/configuration/xml/
    ↪projects/default_project.xml
24  --resources=<path>                    Path to the resources XML file
25                                        Default: /opt/COMPSs//Runtime/configuration/xml/
    ↪resources/default_resources.xml
26  --lang=<name>                         Language of the application (java/c/python)
27                                        Default: Inferred is possible. Otherwise: java
28  --summary                             Displays a task execution summary at the end of the␣
    ↪application execution
29                                        Default: false
30  --log_level=<level>, --debug, -d      Set the debug level: off | info | api | debug | trace
31                                        Warning: Off level compiles with -O2 option disabling␣
    ↪asserts and __debug__
32                                        Default: off
33
34  Advanced options:
35  --extrae_config_file=<path>           Sets a custom extrae config file. Must be in a shared␣
    ↪disk between all COMPSs workers.
```

```
36                                            Default: null
37  --extrae_config_file_python=<path>        Sets a custom extrae config file for python. Must be␣
    ↪in a shared disk between all COMPSs workers.
38                                            Default: null
39  --trace_label=<string>                    Add a label in the generated trace file. Only used in␣
    ↪the case of tracing is activated.
40                                            Default: None
41  --tracing_task_dependencies               Adds communication lines for the task dependencies (␣
    ↪[ true | false ] )
42                                            Default: false
43  --comm=<ClassName>                        Class that implements the adaptor for communications
44                                            Supported adaptors:
45                                                    ├── es.bsc.compss.nio.master.NIOAdaptor
46                                                    └── es.bsc.compss.gat.master.GATAdaptor
47                                            Default: es.bsc.compss.nio.master.NIOAdaptor
48  --conn=<className>                        Class that implements the runtime connector for the␣
    ↪cloud
49                                            Supported connectors:
50                                                    ├── es.bsc.compss.connectors.
    ↪DefaultSSHConnector
51                                                    └── es.bsc.compss.connectors.
    ↪DefaultNoSSHConnector
52                                            Default: es.bsc.compss.connectors.DefaultSSHConnector
53  --streaming=<type>                        Enable the streaming mode for the given type.
54                                            Supported types: FILES, OBJECTS, PSCOS, ALL, NONE
55                                            Default: NONE
56  --streaming_master_name=<str>             Use an specific streaming master node name.
57                                            Default: null
58  --streaming_master_port=<int>             Use an specific port for the streaming master.
59                                            Default: null
60  --scheduler=<className>                   Class that implements the Scheduler for COMPSs
61                                            Supported schedulers:
62                                                      ├── es.bsc.compss.scheduler.fifodatalocation.
    ↪FIFODataLocationScheduler
63                                                      ├── es.bsc.compss.scheduler.fifonew.
    ↪FIFOScheduler
64                                                      ├── es.bsc.compss.scheduler.fifodatanew.
    ↪FIFODataScheduler
65                                                      ├── es.bsc.compss.scheduler.lifonew.
    ↪LIFOScheduler
66                                                      ├── es.bsc.compss.components.impl.
    ↪TaskScheduler
67                                                      └── es.bsc.compss.scheduler.loadbalancing.
    ↪LoadBalancingScheduler
68                                            Default: es.bsc.compss.scheduler.loadbalancing.
    ↪LoadBalancingScheduler
69  --scheduler_config_file=<path>            Path to the file which contains the scheduler␣
    ↪configuration.
70                                            Default: Empty
71  --library_path=<path>                     Non-standard directories to search for libraries (e.g.
    ↪ Java JVM library, Python library, C binding library)
72                                            Default: Working Directory
73  --classpath=<path>                        Path for the application classes / modules
74                                            Default: Working Directory
75  --appdir=<path>                           Path for the application class folder.
76                                            Default: /home/bscuser/Documents/documentation/COMPSs_
    ↪Manuals
```

```
77  --pythonpath=<path>                      Additional folders or paths to add to the PYTHONPATH
78                                           Default: /home/bscuser/Documents/documentation/COMPSs_
    ↪Manuals
79  --env_script=<path>                      Path to the script file where the application␣
    ↪environment variables are defined.
80                                           COMPSs sources this script before running the␣
    ↪application.
81                                           Default: Empty
82  --base_log_dir=<path>                    Base directory to store COMPSs log files (a .COMPSs/␣
    ↪folder will be created inside this location)
83                                           Default: User home
84  --specific_log_dir=<path>                Use a specific directory to store COMPSs log files␣
    ↪(no sandbox is created)
85                                           Warning: Overwrites --base_log_dir option
86                                           Default: Disabled
87  --uuid=<int>                             Preset an application UUID
88                                           Default: Automatic random generation
89  --master_name=<string>                   Hostname of the node to run the COMPSs master
90                                           Default:
91  --master_port=<int>                      Port to run the COMPSs master communications.
92                                           Only for NIO adaptor
93                                           Default: [43000,44000]
94  --jvm_master_opts="<string>"             Extra options for the COMPSs Master JVM. Each option␣
    ↪separed by "," and without blank spaces (Notice the quotes)
95                                           Default:
96  --jvm_workers_opts="<string>"            Extra options for the COMPSs Workers JVMs. Each␣
    ↪option separed by "," and without blank spaces (Notice the quotes)
97                                           Default: -Xms1024m,-Xmx1024m,-Xmn400m
98  --cpu_affinity="<string>"                Sets the CPU affinity for the workers
99                                           Supported options: disabled, automatic, user defined␣
    ↪map of the form "0-8/9,10,11/12-14,15,16"
100                                          Default: automatic
101 --gpu_affinity="<string>"                Sets the GPU affinity for the workers
102                                          Supported options: disabled, automatic, user defined␣
    ↪map of the form "0-8/9,10,11/12-14,15,16"
103                                          Default: automatic
104 --fpga_affinity="<string>"               Sets the FPGA affinity for the workers
105                                          Supported options: disabled, automatic, user defined␣
    ↪map of the form "0-8/9,10,11/12-14,15,16"
106                                          Default: automatic
107 --fpga_reprogram="<string>"              Specify the full command that needs to be executed to␣
    ↪reprogram the FPGA with the desired bitstream. The location must be an absolute path.
108                                          Default:
109 --io_executors=<int>                     IO Executors per worker
110                                          Default: 0
111 --task_count=<int>                       Only for C/Python Bindings. Maximum number of␣
    ↪different functions/methods, invoked from the application, that have been selected as tasks
112                                          Default: 50
113 --input_profile=<path>                   Path to the file which stores the input application␣
    ↪profile
114                                          Default: Empty
115 --output_profile=<path>                  Path to the file to store the application profile at␣
    ↪the end of the execution
116                                          Default: Empty
117 --PyObject_serialize=<bool>              Only for Python Binding. Enable the object␣
    ↪serialization to string when possible (true/false).
```

```
118                                          Default: false
119  --persistent_worker_c=<bool>            Only for C Binding. Enable the persistent worker in c␣
     ↪(true/false).
120                                          Default: false
121  --enable_external_adaptation=<bool>     Enable external adaptation. This option will disable␣
     ↪the Resource Optimizer.
122                                          Default: false
123  --gen_coredump                          Enable master coredump generation
124                                          Default: false
125  --keep_workingdir                       Do not remove the worker working directory after the␣
     ↪execution
126                                          Default: false
127  --python_interpreter=<string>           Python interpreter to use (python/python2/python3).
128                                          Default: python Version:
129  --python_propagate_virtual_environment=<bool>  Propagate the master virtual environment to␣
     ↪the workers (true/false).
130                                               Default: true
131  --python_mpi_worker=<bool>              Use MPI to run the python worker instead of␣
     ↪multiprocessing. (true/false).
132                                          Default: false
133  --python_memory_profile                 Generate a memory profile of the master.
134                                          Default: false
135  --python_worker_cache=<string>          Python worker cache (true/size/false).
136                                          Only for NIO without mpi worker and python >= 3.8.
137                                          Default: false
138  --wall_clock_limit=<int>                Maximum duration of the application (in seconds).
139                                          Default: 0
140  --shutdown_in_node_failure=<bool>       Stop the whole execution in case of Node Failure.
141                                          Default: false
```

Init a docker environment in the root of the repository. The source files path are resolved from the init directory which sometimes can be confusing. As a rule of thumb, initialize the library in a current directory and check the paths are correct running the file with `python3 path_to/file.py` (in this case `python3 python/matmul_files/src/matmul_files.py`).

```
$ cd tutorial_apps
$ pycompss init docker
```

Now we can run the `matmul_files.py` application:

```
$ pycompss run python/matmul_files/src/matmul_files.py 4 4
```

The log files of the execution can be found at `$HOME/.COMPSs`.

You can also init the docker environment inside the examples folder. This will mount the examples directory inside the container so you can execute it without adding the path:

```
$ pycompss init docker -w python/matmul_files/src
$ pycompss run matmul_files.py 4 4
```

Not available

Not available. Submitting jobs for applications is only possible for remote and local environments.

Run application

Submit application execution (job) to queuing system

```
$ pycompss run [COMPSS_ARGS] APP_FILE [APP_ARGS]
```

APP_FILE is required and must be a valid python file. APP_ARGS is optional and can be used to pass any argument to the application.

```
1  --graph=<bool>, --graph, -g             Generation of the complete graph (true/false)
2                                          When no value is provided it is set to true
3                                          Default: false
4  --tracing=<level>, --tracing, -t        Set generation of traces and/or tracing level ( [␣
   ↪true | basic ] | advanced | scorep | arm-map | arm-ddt | false)
5                                          True and basic levels will produce the same traces.
6                                          When no value is provided it is set to 1
7                                          Default: 0
8  --monitoring=<int>, --monitoring, -m    Period between monitoring samples (milliseconds)
9                                          When no value is provided it is set to 2000
10                                         Default: 0
11 --external_debugger=<int>,
12 --external_debugger                     Enables external debugger connection on the specified␣
   ↪port (or 9999 if empty)
13                                         Default: false
14 --jmx_port=<int>                        Enable JVM profiling on specified port
15
16 Runtime configuration options:
17 --task_execution=<compss|storage>       Task execution under COMPSs or Storage.
18                                         Default: compss
19 --storage_impl=<string>                 Path to an storage implementation. Shortcut to␣
   ↪setting pypath and classpath. See Runtime/storage in your installation folder.
20 --storage_conf=<path>                   Path to the storage configuration file
21                                         Default: null
22 --project=<path>                        Path to the project XML file
23                                         Default: /opt/COMPSs//Runtime/configuration/xml/
   ↪projects/default_project.xml
24 --resources=<path>                      Path to the resources XML file
25                                         Default: /opt/COMPSs//Runtime/configuration/xml/
   ↪resources/default_resources.xml
26 --lang=<name>                           Language of the application (java/c/python)
27                                         Default: Inferred is possible. Otherwise: java
28 --summary                               Displays a task execution summary at the end of the␣
   ↪application execution
29                                         Default: false
30 --log_level=<level>, --debug, -d        Set the debug level: off | info | api | debug | trace
31                                         Warning: Off level compiles with -O2 option disabling␣
   ↪asserts and __debug__
32                                         Default: off
33
34 Advanced options:
35 --extrae_config_file=<path>             Sets a custom extrae config file. Must be in a shared␣
   ↪disk between all COMPSs workers.
36                                         Default: null
37 --extrae_config_file_python=<path>      Sets a custom extrae config file for python. Must be␣
   ↪in a shared disk between all COMPSs workers.
38                                         Default: null
39 --trace_label=<string>                  Add a label in the generated trace file. Only used in␣
   ↪the case of tracing is activated.
40                                         Default: None
41 --tracing_task_dependencies             Adds communication lines for the task dependencies (␣
   ↪[ true | false ] )
42                                         Default: false
43 --comm=<ClassName>                      Class that implements the adaptor for communications
```

```
44                                          Supported adaptors:
45                                                  ├── es.bsc.compss.nio.master.NIOAdaptor
46                                                  └── es.bsc.compss.gat.master.GATAdaptor
47                                          Default: es.bsc.compss.nio.master.NIOAdaptor
48  --conn=<className>                      Class that implements the runtime connector for the␣
    ↪cloud
49                                          Supported connectors:
50                                                  ├── es.bsc.compss.connectors.
    ↪DefaultSSHConnector
51                                                  └── es.bsc.compss.connectors.
    ↪DefaultNoSSHConnector
52                                          Default: es.bsc.compss.connectors.DefaultSSHConnector
53  --streaming=<type>                      Enable the streaming mode for the given type.
54                                          Supported types: FILES, OBJECTS, PSCOS, ALL, NONE
55                                          Default: NONE
56  --streaming_master_name=<str>           Use an specific streaming master node name.
57                                          Default: null
58  --streaming_master_port=<int>           Use an specific port for the streaming master.
59                                          Default: null
60  --scheduler=<className>                 Class that implements the Scheduler for COMPSs
61                                          Supported schedulers:
62                                                  ├── es.bsc.compss.scheduler.fifodatalocation.
    ↪FIFODataLocationScheduler
63                                                  ├── es.bsc.compss.scheduler.fifonew.
    ↪FIFOScheduler
64                                                  ├── es.bsc.compss.scheduler.fifodatanew.
    ↪FIFODataScheduler
65                                                  ├── es.bsc.compss.scheduler.lifonew.
    ↪LIFOScheduler
66                                                  ├── es.bsc.compss.components.impl.
    ↪TaskScheduler
67                                                  └── es.bsc.compss.scheduler.loadbalancing.
    ↪LoadBalancingScheduler
68                                          Default: es.bsc.compss.scheduler.loadbalancing.
    ↪LoadBalancingScheduler
69  --scheduler_config_file=<path>          Path to the file which contains the scheduler␣
    ↪configuration.
70                                          Default: Empty
71  --library_path=<path>                   Non-standard directories to search for libraries (e.g.
    ↪ Java JVM library, Python library, C binding library)
72                                          Default: Working Directory
73  --classpath=<path>                      Path for the application classes / modules
74                                          Default: Working Directory
75  --appdir=<path>                         Path for the application class folder.
76                                          Default: /home/bscuser/Documents/documentation/COMPSs_
    ↪Manuals
77  --pythonpath=<path>                     Additional folders or paths to add to the PYTHONPATH
78                                          Default: /home/bscuser/Documents/documentation/COMPSs_
    ↪Manuals
79  --env_script=<path>                     Path to the script file where the application␣
    ↪environment variables are defined.
80                                          COMPSs sources this script before running the␣
    ↪application.
81                                          Default: Empty
82  --base_log_dir=<path>                   Base directory to store COMPSs log files (a .COMPSs/␣
    ↪folder will be created inside this location)
```

```
83                                              Default: User home
84  --specific_log_dir=<path>                   Use a specific directory to store COMPSs log files␣
    ↪(no sandbox is created)
85                                              Warning: Overwrites --base_log_dir option
86                                              Default: Disabled
87  --uuid=<int>                                Preset an application UUID
88                                              Default: Automatic random generation
89  --master_name=<string>                      Hostname of the node to run the COMPSs master
90                                              Default:
91  --master_port=<int>                         Port to run the COMPSs master communications.
92                                              Only for NIO adaptor
93                                              Default: [43000,44000]
94  --jvm_master_opts="<string>"                Extra options for the COMPSs Master JVM. Each option␣
    ↪separed by "," and without blank spaces (Notice the quotes)
95                                              Default:
96  --jvm_workers_opts="<string>"               Extra options for the COMPSs Workers JVMs. Each␣
    ↪option separed by "," and without blank spaces (Notice the quotes)
97                                              Default: -Xms1024m,-Xmx1024m,-Xmn400m
98  --cpu_affinity="<string>"                   Sets the CPU affinity for the workers
99                                              Supported options: disabled, automatic, user defined␣
    ↪map of the form "0-8/9,10,11/12-14,15,16"
100                                             Default: automatic
101 --gpu_affinity="<string>"                   Sets the GPU affinity for the workers
102                                             Supported options: disabled, automatic, user defined␣
    ↪map of the form "0-8/9,10,11/12-14,15,16"
103                                             Default: automatic
104 --fpga_affinity="<string>"                  Sets the FPGA affinity for the workers
105                                             Supported options: disabled, automatic, user defined␣
    ↪map of the form "0-8/9,10,11/12-14,15,16"
106                                             Default: automatic
107 --fpga_reprogram="<string>"                 Specify the full command that needs to be executed to␣
    ↪reprogram the FPGA with the desired bitstream. The location must be an absolute path.
108                                             Default:
109 --io_executors=<int>                        IO Executors per worker
110                                             Default: 0
111 --task_count=<int>                          Only for C/Python Bindings. Maximum number of␣
    ↪different functions/methods, invoked from the application, that have been selected as tasks
112                                             Default: 50
113 --input_profile=<path>                      Path to the file which stores the input application␣
    ↪profile
114                                             Default: Empty
115 --output_profile=<path>                     Path to the file to store the application profile at␣
    ↪the end of the execution
116                                             Default: Empty
117 --PyObject_serialize=<bool>                 Only for Python Binding. Enable the object␣
    ↪serialization to string when possible (true/false).
118                                             Default: false
119 --persistent_worker_c=<bool>                Only for C Binding. Enable the persistent worker in c␣
    ↪(true/false).
120                                             Default: false
121 --enable_external_adaptation=<bool>         Enable external adaptation. This option will disable␣
    ↪the Resource Optimizer.
122                                             Default: false
123 --gen_coredump                              Enable master coredump generation
124                                             Default: false
125 --keep_workingdir                           Do not remove the worker working directory after the␣
    ↪execution
```

```
126                                               Default: false
127  --python_interpreter=<string>                Python interpreter to use (python/python2/python3).
128                                               Default: python Version:
129  --python_propagate_virtual_environment=<bool>  Propagate the master virtual environment to␣
     ↪the workers (true/false).
130                                                     Default: true
131  --python_mpi_worker=<bool>                   Use MPI to run the python worker instead of␣
     ↪multiprocessing. (true/false).
132                                               Default: false
133  --python_memory_profile                      Generate a memory profile of the master.
134                                               Default: false
135  --python_worker_cache=<string>               Python worker cache (true/size/false).
136                                               Only for NIO without mpi worker and python >= 3.8.
137                                               Default: false
138  --wall_clock_limit=<int>                     Maximum duration of the application (in seconds).
139                                               Default: 0
140  --shutdown_in_node_failure=<bool>            Stop the whole execution in case of Node Failure.
141                                               Default: false
```

Init a local environment in the root of the repository. The source files path are resolved from the init directory which sometimes can be confusing. As a rule of thumb, initialize the library in a current directory and check the paths are correct running the file with `python3 path_to/file.py` (in this case `python3 python/matmul_files/src/matmul_files.py`).

```
$ cd tutorial_apps
$ pycompss init local
```

Now we can run the `matmul_files.py` application:

```
$ pycompss run python/matmul_files/src/matmul_files.py 4 4
```

The log files of the execution can be found at `$HOME/.COMPSs`.

You can also init the local environment inside the examples folder. This will mount the examples directory inside the container so you can execute it without adding the path:

```
$ pycompss init local -w python/matmul_files/src
$ pycompss run matmul_files.py 4 4
```

Important: To be able to submit a job in a local environment you must have installed some cluster management/job scheduling system .i.e SLURM, SGE, PBS, etc.

The `pycompss job` command can be used to submit, cancel and list jobs to a remote environment. It is only available for local and remote environments.

```
$ pycompss job submit -e [ENV_VAR...] [COMPSS_ARGS] APP_FILE [APP_ARGS]
```

ENV_VAR is optional and can be used to pass any environment variable to the application. APP_FILE is required and must be a valid python file inside app directory. APP_ARGS is optional and can be used to pass any argument to the application.

```
1  Queue system configuration:
2      --sc_cfg=<name>                           SuperComputer configuration file to use. Must␣
   ↪exist inside queues/cfgs/
3                                                Default: default
4
```

```
 5   Submission configuration:
 6   General submision arguments:
 7       --exec_time=<minutes>                   Expected execution time of the application (in␣
     ↪minutes)
 8                                               Default: 10
 9       --job_name=<name>                       Job name
10                                               Default: COMPSs
11       --queue=<name>                          Queue name to submit the job. Depends on the␣
     ↪queue system.
12                                               For example (MN3): bsc_cs | bsc_debug | debug |␣
     ↪interactive
13                                               Default: default
14       --reservation=<name>                    Reservation to use when submitting the job.
15                                               Default: disabled
16       --env_script=<path/to/script>           Script to source the required environment for the␣
     ↪application.
17                                               Default: Empty
18       --extra_submit_flag=<flag>              Flag to pass queue system flags not supported by␣
     ↪default command flags.
19                                               Spaces must be added as '#'
20                                               Default: Empty
21       --cpus_per_task                         Number of cpus per task the queue system must␣
     ↪allocate per task.
22                                               Note that this will be equal to the cpus_per_node␣
     ↪in a worker node and
23                                               equal to the worker_in_master_cpus in a master␣
     ↪node respectively.
24                                               Default: false
25       --job_dependency=<jobID>                Postpone job execution until the job dependency␣
     ↪has ended.
26                                               Default: None
27       --forward_time_limit=<true|false>       Forward the queue system time limit to the␣
     ↪runtime.
28                                               It will stop the application in a controlled way.
29                                               Default: true
30       --storage_home=<string>                 Root installation dir of the storage␣
     ↪implementation
31                                               Default: null
32       --storage_props=<string>                Absolute path of the storage properties file
33                                               Mandatory if storage_home is defined
34   Agents deployment arguments:
35       --agents=<string>                       Hierarchy of agents for the deployment. Accepted␣
     ↪values: plain|tree
36                                               Default: tree
37       --agents                                Deploys the runtime as agents instead of the␣
     ↪classic Master-Worker deployment.
38                                               Default: disabled
39
40   Homogeneous submission arguments:
41       --num_nodes=<int>                       Number of nodes to use
42                                               Default: 2
43       --num_switches=<int>                    Maximum number of different switches. Select 0␣
     ↪for no restrictions.
44                                               Maximum nodes per switch: 18
45                                               Only available for at least 4 nodes.
46                                               Default: 0
```

```
47  Heterogeneous submission arguments:
48      --type_cfg=<file_location>                  Location of the file with the descriptions of␣
    ↪node type requests
49                                                  File should follow the following format:
50                                                  type_X(){
51                                                  cpus_per_node=24
52                                                  node_memory=96
53                                                  ...
54                                                  }
55                                                  type_Y(){
56                                                  ...
57                                                  }
58      --master=<master_node_type>                 Node type for the master
59                                                  (Node type descriptions are provided in the --
    ↪type_cfg flag)
60      --workers=type_X:nodes,type_Y:nodes         Node type and number of nodes per type for the␣
    ↪workers
61                                                  (Node type descriptions are provided in the --
    ↪type_cfg flag)
62  Launch configuration:
63      --cpus_per_node=<int>                       Available CPU computing units on each node
64                                                  Default: 32
65      --gpus_per_node=<int>                       Available GPU computing units on each node
66                                                  Default: 0
67      --fpgas_per_node=<int>                      Available FPGA computing units on each node
68                                                  Default:
69      --io_executors=<int>                        Number of IO executors on each node
70                                                  Default: 0
71      --fpga_reprogram="<string>                  Specify the full command that needs to be␣
    ↪executed to reprogram the FPGA with
72                                                  the desired bitstream. The location must be an␣
    ↪absolute path.
73                                                  Default:
74      --max_tasks_per_node=<int>                  Maximum number of simultaneous tasks running on a␣
    ↪node
75                                                  Default: -1
76      --node_memory=<MB>                          Maximum node memory: disabled | <int> (MB)
77                                                  Default: disabled
78      --node_storage_bandwidth=<MB>               Maximum node storage bandwidth: <int> (MB)
79                                                  Default:
80
81      --network=<name>                            Communication network for transfers: default |␣
    ↪ethernet | infiniband | data.
82                                                  Default: ethernet
83
84      --prolog="<string>"                         Task to execute before launching COMPSs (Notice␣
    ↪the quotes)
85                                                  If the task has arguments split them by ","␣
    ↪rather than spaces.
86                                                  This argument can appear multiple times for more␣
    ↪than one prolog action
87                                                  Default: Empty
88      --epilog="<string>"                         Task to execute after executing the COMPSs␣
    ↪application (Notice the quotes)
89                                                  If the task has arguments split them by ","␣
    ↪rather than spaces.
```

```
90                                                 This argument can appear multiple times for more␣
   ↪than one epilog action
91                                                 Default: Empty
92
93      --master_working_dir=<path>                Working directory of the application
94                                                 Default: .
95      --worker_working_dir=<name | path>         Worker directory. Use: local_disk | shared_disk |
   ↪<path>
96                                                 Default: local_disk
97
98      --worker_in_master_cpus=<int>              Maximum number of CPU computing units that the␣
   ↪master node can run as worker. Cannot exceed cpus_per_node.
99                                                 Default: 0
100     --worker_in_master_memory=<int> MB         Maximum memory in master node assigned to the␣
   ↪worker. Cannot exceed the node_memory.
101                                                Mandatory if worker_in_master_cpus is specified.
102                                                Default: disabled
103     --worker_port_range=<min>,<max>            Port range used by the NIO adaptor at the worker␣
   ↪side
104                                                Default: 43001,43005
105     --jvm_worker_in_master_opts="<string>"     Extra options for the JVM of the COMPSs Worker in␣
   ↪the Master Node.
106                                                Each option separed by "," and without blank␣
   ↪spaces (Notice the quotes)
107                                                Default:
108     --container_image=<path>                   Runs the application by means of a container␣
   ↪engine image
109                                                Default: Empty
110     --container_compss_path=<path>             Path where compss is installed in the container␣
   ↪image
111                                                Default: /opt/COMPSs
112     --container_opts="<string>"                Options to pass to the container engine
113                                                Default: empty
114     --elasticity=<max_extra_nodes>             Activate elasticity specifiying the maximum extra␣
   ↪nodes (ONLY AVAILABLE FORM SLURM CLUSTERS WITH NIO ADAPTOR)
115                                                Default: 0
116     --automatic_scaling=<bool>                 Enable or disable the runtime automatic scaling␣
   ↪(for elasticity)
117                                                Default: true
118     --jupyter_notebook=<path>,                 Swap the COMPSs master initialization with␣
   ↪jupyter notebook from the specified path.
119     --jupyter_notebook                         Default: false
120     --ipython                                  Swap the COMPSs master initialization with␣
   ↪ipython.
121                                                Default: empty
122
123
124  Runcompss configuration:
125
126
127  Tools enablers:
128     --graph=<bool>, --graph, -g                Generation of the complete graph (true/false)
129                                                When no value is provided it is set to true
130                                                Default: false
131     --tracing=<level>, --tracing, -t           Set generation of traces and/or tracing level ( [␣
   ↪true | basic ] | advanced | scorep | arm-map | arm-ddt | false)
```

```
132                                                 True and basic levels will produce the same␣
      ↪traces.
133                                                 When no value is provided it is set to 1
134                                                 Default: 0
135     --monitoring=<int>, --monitoring, -m        Period between monitoring samples (milliseconds)
136                                                 When no value is provided it is set to 2000
137                                                 Default: 0
138     --external_debugger=<int>,
139     --external_debugger                         Enables external debugger connection on the␣
      ↪specified port (or 9999 if empty)
140                                                 Default: false
141     --jmx_port=<int>                            Enable JVM profiling on specified port
142
143 Runtime configuration options:
144     --task_execution=<compss|storage>           Task execution under COMPSs or Storage.
145                                                 Default: compss
146     --storage_impl=<string>                     Path to an storage implementation. Shortcut to␣
      ↪setting pypath and classpath. See Runtime/storage in your installation folder.
147     --storage_conf=<path>                       Path to the storage configuration file
148                                                 Default: null
149     --project=<path>                            Path to the project XML file
150                                                 Default: /opt/COMPSs//Runtime/configuration/xml/
      ↪projects/default_project.xml
151     --resources=<path>                          Path to the resources XML file
152                                                 Default: /opt/COMPSs//Runtime/configuration/xml/
      ↪resources/default_resources.xml
153     --lang=<name>                               Language of the application (java/c/python)
154                                                 Default: Inferred is possible. Otherwise: java
155     --summary                                   Displays a task execution summary at the end of␣
      ↪the application execution
156                                                 Default: false
157     --log_level=<level>, --debug, -d            Set the debug level: off | info | api | debug |␣
      ↪trace
158                                                 Warning: Off level compiles with -O2 option␣
      ↪disabling asserts and __debug__
159                                                 Default: off
160
161 Advanced options:
162     --extrae_config_file=<path>                 Sets a custom extrae config file. Must be in a␣
      ↪shared disk between all COMPSs workers.
163                                                 Default: null
164     --extrae_config_file_python=<path>          Sets a custom extrae config file for python. Must␣
      ↪be in a shared disk between all COMPSs workers.
165                                                 Default: null
166     --trace_label=<string>                      Add a label in the generated trace file. Only␣
      ↪used in the case of tracing is activated.
167                                                 Default: None
168     --tracing_task_dependencies                 Adds communication lines for the task␣
      ↪dependencies ( [ true | false ] )
169                                                 Default: false
170     --comm=<ClassName>                          Class that implements the adaptor for␣
      ↪communications
171                                                 Supported adaptors:
172                                                         ┌── es.bsc.compss.nio.master.NIOAdaptor
173                                                         └── es.bsc.compss.gat.master.GATAdaptor
174                                                 Default: es.bsc.compss.nio.master.NIOAdaptor
```

```
175      --conn=<className>                    Class that implements the runtime connector for␣
     ↪the cloud
176                                            Supported connectors:
177                                                ├── es.bsc.compss.connectors.
     ↪DefaultSSHConnector
178                                                └── es.bsc.compss.connectors.
     ↪DefaultNoSSHConnector
179                                            Default: es.bsc.compss.connectors.
     ↪DefaultSSHConnector
180      --streaming=<type>                    Enable the streaming mode for the given type.
181                                            Supported types: FILES, OBJECTS, PSCOS, ALL, NONE
182                                            Default: NONE
183      --streaming_master_name=<str>         Use an specific streaming master node name.
184                                            Default: null
185      --streaming_master_port=<int>         Use an specific port for the streaming master.
186                                            Default: null
187      --scheduler=<className>               Class that implements the Scheduler for COMPSs
188                                            Supported schedulers:
189                                                ├── es.bsc.compss.scheduler.fifodatalocation.
     ↪FIFODataLocationScheduler
190                                                ├── es.bsc.compss.scheduler.fifonew.
     ↪FIFOScheduler
191                                                ├── es.bsc.compss.scheduler.fifodatanew.
     ↪FIFODataScheduler
192                                                ├── es.bsc.compss.scheduler.lifonew.
     ↪LIFOScheduler
193                                                ├── es.bsc.compss.components.impl.
     ↪TaskScheduler
194                                                └── es.bsc.compss.scheduler.loadbalancing.
     ↪LoadBalancingScheduler
195                                            Default: es.bsc.compss.scheduler.loadbalancing.
     ↪LoadBalancingScheduler
196      --scheduler_config_file=<path>        Path to the file which contains the scheduler␣
     ↪configuration.
197                                            Default: Empty
198      --library_path=<path>                 Non-standard directories to search for libraries␣
     ↪(e.g. Java JVM library, Python library, C binding library)
199                                            Default: Working Directory
200      --classpath=<path>                    Path for the application classes / modules
201                                            Default: Working Directory
202      --appdir=<path>                       Path for the application class folder.
203                                            Default: /home/bscuser/Documents/framework/
     ↪builders/specs/cli/pyCOMPSsCLIResources
204      --pythonpath=<path>                   Additional folders or paths to add to the␣
     ↪PYTHONPATH
205                                            Default: /home/bscuser/Documents/framework/
     ↪builders/specs/cli/pyCOMPSsCLIResources
206      --env_script=<path>                   Path to the script file where the application␣
     ↪environment variables are defined.
207                                            COMPSs sources this script before running the␣
     ↪application.
208                                            Default: Empty
209      --base_log_dir=<path>                 Base directory to store COMPSs log files (a .
     ↪COMPSs/ folder will be created inside this location)
210                                            Default: User home
211      --specific_log_dir=<path>             Use a specific directory to store COMPSs log␣
     ↪files (no sandbox is created)
```

```
212                                             Warning: Overwrites --base_log_dir option
213                                             Default: Disabled
214     --uuid=<int>                            Preset an application UUID
215                                             Default: Automatic random generation
216     --master_name=<string>                  Hostname of the node to run the COMPSs master
217                                             Default:
218     --master_port=<int>                     Port to run the COMPSs master communications.
219                                             Only for NIO adaptor
220                                             Default: [43000,44000]
221     --jvm_master_opts="<string>"            Extra options for the COMPSs Master JVM. Each␣
 ↪option separed by "," and without blank spaces (Notice the quotes)
222                                             Default:
223     --jvm_workers_opts="<string>"           Extra options for the COMPSs Workers JVMs. Each␣
 ↪option separed by "," and without blank spaces (Notice the quotes)
224                                             Default: -Xms1024m,-Xmx1024m,-Xmn400m
225     --cpu_affinity="<string>"               Sets the CPU affinity for the workers
226                                             Supported options: disabled, automatic, user␣
 ↪defined map of the form "0-8/9,10,11/12-14,15,16"
227                                             Default: automatic
228     --gpu_affinity="<string>"               Sets the GPU affinity for the workers
229                                             Supported options: disabled, automatic, user␣
 ↪defined map of the form "0-8/9,10,11/12-14,15,16"
230                                             Default: automatic
231     --fpga_affinity="<string>"              Sets the FPGA affinity for the workers
232                                             Supported options: disabled, automatic, user␣
 ↪defined map of the form "0-8/9,10,11/12-14,15,16"
233                                             Default: automatic
234     --fpga_reprogram="<string>"             Specify the full command that needs to be␣
 ↪executed to reprogram the FPGA with the desired bitstream. The location must be an absolute␣
 ↪path.
235                                             Default:
236     --io_executors=<int>                    IO Executors per worker
237                                             Default: 0
238     --task_count=<int>                      Only for C/Python Bindings. Maximum number of␣
 ↪different functions/methods, invoked from the application, that have been selected as tasks
239                                             Default: 50
240     --input_profile=<path>                  Path to the file which stores the input␣
 ↪application profile
241                                             Default: Empty
242     --output_profile=<path>                 Path to the file to store the application profile␣
 ↪at the end of the execution
243                                             Default: Empty
244     --PyObject_serialize=<bool>             Only for Python Binding. Enable the object␣
 ↪serialization to string when possible (true/false).
245                                             Default: false
246     --persistent_worker_c=<bool>            Only for C Binding. Enable the persistent worker␣
 ↪in c (true/false).
247                                             Default: false
248     --enable_external_adaptation=<bool>     Enable external adaptation. This option will␣
 ↪disable the Resource Optimizer.
249                                             Default: false
250     --gen_coredump                          Enable master coredump generation
251                                             Default: false
252     --keep_workingdir                       Do not remove the worker working directory after␣
 ↪the execution
253                                             Default: false
```

```
254      --python_interpreter=<string>            Python interpreter to use (python/python2/
    ↪python3).
255                                               Default: python Version:
256      --python_propagate_virtual_environment=<bool>  Propagate the master virtual environment␣
    ↪to the workers (true/false).
257                                                  Default: true
258      --python_mpi_worker=<bool>               Use MPI to run the python worker instead of␣
    ↪multiprocessing. (true/false).
259                                               Default: false
260      --python_memory_profile                  Generate a memory profile of the master.
261                                               Default: false
262      --python_worker_cache=<string>           Python worker cache (true/size/false).
263                                               Only for NIO without mpi worker and python >= 3.8.
264                                               Default: false
265      --wall_clock_limit=<int>                 Maximum duration of the application (in seconds).
266                                               Default: 0
267      --shutdown_in_node_failure=<bool>        Stop the whole execution in case of Node Failure.
268                                               Default: false
```

The command will submit a job and return the Job ID. In order to run a COMPSs program on the local machine
we can use the command:

```
$ cd tutorial_apps/python/matmul_files/src
$ pycompss job submit -e ComputingUnits=1 --num_nodes=2 --exec_time=10 --worker_working_
    ↪dir=local_disk --tracing=false --lang=python --qos=debug matmul_files.py 4 4
```

Submit application execution (job) to queuing system

Run application

The `pycompss job` command can be used to submit, cancel and list jobs to a remote environment. It is only
available for local and remote environments.

```
$ pycompss job submit -e [ENV_VAR...] -app APP_NAME [COMPSS_ARGS] APP_FILE [APP_ARGS]
```

ENV_VAR is optional and can be used to pass any environment variable to the application. APP_NAME is
required and must be a valid application name previously deployed. APP_FILE is required and must be a valid
python file inside app directory. APP_ARGS is optional and can be used to pass any argument to the application.

```
1  Queue system configuration:
2      --sc_cfg=<name>                          SuperComputer configuration file to use. Must␣
    ↪exist inside queues/cfgs/
3                                               Default: default
4
5  Submission configuration:
6  General submision arguments:
7      --exec_time=<minutes>                    Expected execution time of the application (in␣
    ↪minutes)
8                                               Default: 10
9      --job_name=<name>                        Job name
10                                              Default: COMPSs
11     --queue=<name>                           Queue name to submit the job. Depends on the␣
    ↪queue system.
12                                              For example (MN3): bsc_cs | bsc_debug | debug |␣
    ↪interactive
13                                              Default: default
14     --reservation=<name>                     Reservation to use when submitting the job.
15                                              Default: disabled
```

```
16      --env_script=<path/to/script>        Script to source the required environment for the␣
    ↪application.
17                                           Default: Empty
18      --extra_submit_flag=<flag>           Flag to pass queue system flags not supported by␣
    ↪default command flags.
19                                           Spaces must be added as '#'
20                                           Default: Empty
21      --cpus_per_task                      Number of cpus per task the queue system must␣
    ↪allocate per task.
22                                           Note that this will be equal to the cpus_per_node␣
    ↪in a worker node and
23                                           equal to the worker_in_master_cpus in a master␣
    ↪node respectively.
24                                           Default: false
25      --job_dependency=<jobID>             Postpone job execution until the job dependency␣
    ↪has ended.
26                                           Default: None
27      --forward_time_limit=<true|false>    Forward the queue system time limit to the␣
    ↪runtime.
28                                           It will stop the application in a controlled way.
29                                           Default: true
30      --storage_home=<string>              Root installation dir of the storage␣
    ↪implementation
31                                           Default: null
32      --storage_props=<string>             Absolute path of the storage properties file
33                                           Mandatory if storage_home is defined
34  Agents deployment arguments:
35      --agents=<string>                    Hierarchy of agents for the deployment. Accepted␣
    ↪values: plain|tree
36                                           Default: tree
37      --agents                             Deploys the runtime as agents instead of the␣
    ↪classic Master-Worker deployment.
38                                           Default: disabled
39
40  Homogeneous submission arguments:
41      --num_nodes=<int>                    Number of nodes to use
42                                           Default: 2
43      --num_switches=<int>                 Maximum number of different switches. Select 0␣
    ↪for no restrictions.
44                                           Maximum nodes per switch: 18
45                                           Only available for at least 4 nodes.
46                                           Default: 0
47  Heterogeneous submission arguments:
48      --type_cfg=<file_location>           Location of the file with the descriptions of␣
    ↪node type requests
49                                           File should follow the following format:
50                                           type_X(){
51                                           cpus_per_node=24
52                                           node_memory=96
53                                           ...
54                                           }
55                                           type_Y(){
56                                           ...
57                                           }
58      --master=<master_node_type>          Node type for the master
59                                           (Node type descriptions are provided in the --
    ↪type_cfg flag)
```

```
60      --workers=type_X:nodes,type_Y:nodes      Node type and number of nodes per type for the␣
→workers
61                                               (Node type descriptions are provided in the --␣
→type_cfg flag)
62  Launch configuration:
63      --cpus_per_node=<int>                    Available CPU computing units on each node
64                                               Default: 32
65      --gpus_per_node=<int>                    Available GPU computing units on each node
66                                               Default: 0
67      --fpgas_per_node=<int>                   Available FPGA computing units on each node
68                                               Default:
69      --io_executors=<int>                     Number of IO executors on each node
70                                               Default: 0
71      --fpga_reprogram="<string>               Specify the full command that needs to be␣
→executed to reprogram the FPGA with
72                                               the desired bitstream. The location must be an␣
→absolute path.
73                                               Default:
74      --max_tasks_per_node=<int>               Maximum number of simultaneous tasks running on a␣
→node
75                                               Default: -1
76      --node_memory=<MB>                       Maximum node memory: disabled | <int> (MB)
77                                               Default: disabled
78      --node_storage_bandwidth=<MB>            Maximum node storage bandwidth: <int> (MB)
79                                               Default:
80
81      --network=<name>                         Communication network for transfers: default |␣
→ethernet | infiniband | data.
82                                               Default: ethernet
83
84      --prolog="<string>"                      Task to execute before launching COMPSs (Notice␣
→the quotes)
85                                               If the task has arguments split them by ","␣
→rather than spaces.
86                                               This argument can appear multiple times for more␣
→than one prolog action
87                                               Default: Empty
88      --epilog="<string>"                      Task to execute after executing the COMPSs␣
→application (Notice the quotes)
89                                               If the task has arguments split them by ","␣
→rather than spaces.
90                                               This argument can appear multiple times for more␣
→than one epilog action
91                                               Default: Empty
92
93      --master_working_dir=<path>              Working directory of the application
94                                               Default: .
95      --worker_working_dir=<name | path>       Worker directory. Use: local_disk | shared_disk |
→<path>
96                                               Default: local_disk
97
98      --worker_in_master_cpus=<int>            Maximum number of CPU computing units that the␣
→master node can run as worker. Cannot exceed cpus_per_node.
99                                               Default: 0
100     --worker_in_master_memory=<int> MB       Maximum memory in master node assigned to the␣
→worker. Cannot exceed the node_memory.
```

```
101 │                                              Mandatory if worker_in_master_cpus is specified.
102 │                                              Default: disabled
103 │     --worker_port_range=<min>,<max>          Port range used by the NIO adaptor at the worker␣
    │ ↪side
104 │                                              Default: 43001,43005
105 │     --jvm_worker_in_master_opts="<string>"   Extra options for the JVM of the COMPSs Worker in␣
    │ ↪the Master Node.
106 │                                              Each option separed by "," and without blank␣
    │ ↪spaces (Notice the quotes)
107 │                                              Default:
108 │     --container_image=<path>                 Runs the application by means of a container␣
    │ ↪engine image
109 │                                              Default: Empty
110 │     --container_compss_path=<path>           Path where compss is installed in the container␣
    │ ↪image
111 │                                              Default: /opt/COMPSs
112 │     --container_opts="<string>"              Options to pass to the container engine
113 │                                              Default: empty
114 │     --elasticity=<max_extra_nodes>           Activate elasticity specifiying the maximum extra␣
    │ ↪nodes (ONLY AVAILABLE FORM SLURM CLUSTERS WITH NIO ADAPTOR)
115 │                                              Default: 0
116 │     --automatic_scaling=<bool>               Enable or disable the runtime automatic scaling␣
    │ ↪(for elasticity)
117 │                                              Default: true
118 │     --jupyter_notebook=<path>,               Swap the COMPSs master initialization with␣
    │ ↪jupyter notebook from the specified path.
119 │     --jupyter_notebook                       Default: false
120 │     --ipython                                Swap the COMPSs master initialization with␣
    │ ↪ipython.
121 │                                              Default: empty
122 │
123 │
124 │ Runcompss configuration:
125 │
126 │
127 │ Tools enablers:
128 │     --graph=<bool>, --graph, -g              Generation of the complete graph (true/false)
129 │                                              When no value is provided it is set to true
130 │                                              Default: false
131 │     --tracing=<level>, --tracing, -t         Set generation of traces and/or tracing level ( [␣
    │ ↪true | basic ] | advanced | scorep | arm-map | arm-ddt | false)
132 │                                              True and basic levels will produce the same␣
    │ ↪traces.
133 │                                              When no value is provided it is set to 1
134 │                                              Default: 0
135 │     --monitoring=<int>, --monitoring, -m     Period between monitoring samples (milliseconds)
136 │                                              When no value is provided it is set to 2000
137 │                                              Default: 0
138 │     --external_debugger=<int>,
139 │     --external_debugger                      Enables external debugger connection on the␣
    │ ↪specified port (or 9999 if empty)
140 │                                              Default: false
141 │     --jmx_port=<int>                         Enable JVM profiling on specified port
142 │
143 │ Runtime configuration options:
144 │     --task_execution=<compss|storage>        Task execution under COMPSs or Storage.
```

```
145  │                                                Default: compss
146  │      --storage_impl=<string>                   Path to an storage implementation. Shortcut to␣
     →setting pypath and classpath. See Runtime/storage in your installation folder.
147  │      --storage_conf=<path>                     Path to the storage configuration file
148  │                                                Default: null
149  │      --project=<path>                          Path to the project XML file
150  │                                                Default: /opt/COMPSs//Runtime/configuration/xml/
     →projects/default_project.xml
151  │      --resources=<path>                        Path to the resources XML file
152  │                                                Default: /opt/COMPSs//Runtime/configuration/xml/
     →resources/default_resources.xml
153  │      --lang=<name>                             Language of the application (java/c/python)
154  │                                                Default: Inferred is possible. Otherwise: java
155  │      --summary                                 Displays a task execution summary at the end of␣
     →the application execution
156  │                                                Default: false
157  │      --log_level=<level>, --debug, -d          Set the debug level: off | info | api | debug |␣
     →trace
158  │                                                Warning: Off level compiles with -O2 option␣
     →disabling asserts and __debug__
159  │                                                Default: off
160  │
161  │Advanced options:
162  │      --extrae_config_file=<path>               Sets a custom extrae config file. Must be in a␣
     →shared disk between all COMPSs workers.
163  │                                                Default: null
164  │      --extrae_config_file_python=<path>        Sets a custom extrae config file for python. Must␣
     →be in a shared disk between all COMPSs workers.
165  │                                                Default: null
166  │      --trace_label=<string>                    Add a label in the generated trace file. Only␣
     →used in the case of tracing is activated.
167  │                                                Default: None
168  │      --tracing_task_dependencies               Adds communication lines for the task␣
     →dependencies ( [ true | false ] )
169  │                                                Default: false
170  │      --comm=<ClassName>                        Class that implements the adaptor for␣
     →communications
171  │                                                Supported adaptors:
172  │                                                      ├── es.bsc.compss.nio.master.NIOAdaptor
173  │                                                      └── es.bsc.compss.gat.master.GATAdaptor
174  │                                                Default: es.bsc.compss.nio.master.NIOAdaptor
175  │      --conn=<className>                        Class that implements the runtime connector for␣
     →the cloud
176  │                                                Supported connectors:
177  │                                                      ├── es.bsc.compss.connectors.
     →DefaultSSHConnector
178  │                                                      └── es.bsc.compss.connectors.
     →DefaultNoSSHConnector
179  │                                                Default: es.bsc.compss.connectors.
     →DefaultSSHConnector
180  │      --streaming=<type>                        Enable the streaming mode for the given type.
181  │                                                Supported types: FILES, OBJECTS, PSCOS, ALL, NONE
182  │                                                Default: NONE
183  │      --streaming_master_name=<str>             Use an specific streaming master node name.
184  │                                                Default: null
185  │      --streaming_master_port=<int>             Use an specific port for the streaming master.
```

```
186                                              Default: null
187     --scheduler=<className>                  Class that implements the Scheduler for COMPSs
188                                              Supported schedulers:
189                                                  ├── es.bsc.compss.scheduler.fifodatalocation.
→FIFODataLocationScheduler
190                                                  ├── es.bsc.compss.scheduler.fifonew.
→FIFOScheduler
191                                                  ├── es.bsc.compss.scheduler.fifodatanew.
→FIFODataScheduler
192                                                  ├── es.bsc.compss.scheduler.lifonew.
→LIFOScheduler
193                                                  ├── es.bsc.compss.components.impl.
→TaskScheduler
194                                                  └── es.bsc.compss.scheduler.loadbalancing.
→LoadBalancingScheduler
195                                              Default: es.bsc.compss.scheduler.loadbalancing.
→LoadBalancingScheduler
196     --scheduler_config_file=<path>           Path to the file which contains the scheduler␣
→configuration.
197                                              Default: Empty
198     --library_path=<path>                    Non-standard directories to search for libraries␣
→(e.g. Java JVM library, Python library, C binding library)
199                                              Default: Working Directory
200     --classpath=<path>                       Path for the application classes / modules
201                                              Default: Working Directory
202     --appdir=<path>                          Path for the application class folder.
203                                              Default: /home/bscuser/Documents/framework/
→builders/specs/cli/pyCOMPSsCLIResources
204     --pythonpath=<path>                      Additional folders or paths to add to the␣
→PYTHONPATH
205                                              Default: /home/bscuser/Documents/framework/
→builders/specs/cli/pyCOMPSsCLIResources
206     --env_script=<path>                      Path to the script file where the application␣
→environment variables are defined.
207                                              COMPSs sources this script before running the␣
→application.
208                                              Default: Empty
209     --base_log_dir=<path>                    Base directory to store COMPSs log files (a .
→COMPSs/ folder will be created inside this location)
210                                              Default: User home
211     --specific_log_dir=<path>                Use a specific directory to store COMPSs log␣
→files (no sandbox is created)
212                                              Warning: Overwrites --base_log_dir option
213                                              Default: Disabled
214   --uuid=<int>                               Preset an application UUID
215                                              Default: Automatic random generation
216   --master_name=<string>                     Hostname of the node to run the COMPSs master
217                                              Default:
218   --master_port=<int>                        Port to run the COMPSs master communications.
219                                              Only for NIO adaptor
220                                              Default: [43000,44000]
221     --jvm_master_opts="<string>"             Extra options for the COMPSs Master JVM. Each␣
→option separed by "," and without blank spaces (Notice the quotes)
222                                              Default:
223     --jvm_workers_opts="<string>"            Extra options for the COMPSs Workers JVMs. Each␣
→option separed by "," and without blank spaces (Notice the quotes)
```

```
224                                              Default: -Xms1024m,-Xmx1024m,-Xmn400m
225      --cpu_affinity="<string>"               Sets the CPU affinity for the workers
226                                              Supported options: disabled, automatic, user␣
     ↪defined map of the form "0-8/9,10,11/12-14,15,16"
227                                              Default: automatic
228      --gpu_affinity="<string>"               Sets the GPU affinity for the workers
229                                              Supported options: disabled, automatic, user␣
     ↪defined map of the form "0-8/9,10,11/12-14,15,16"
230                                              Default: automatic
231      --fpga_affinity="<string>"              Sets the FPGA affinity for the workers
232                                              Supported options: disabled, automatic, user␣
     ↪defined map of the form "0-8/9,10,11/12-14,15,16"
233                                              Default: automatic
234      --fpga_reprogram="<string>"             Specify the full command that needs to be␣
     ↪executed to reprogram the FPGA with the desired bitstream. The location must be an absolute␣
     ↪path.
235                                              Default:
236      --io_executors=<int>                    IO Executors per worker
237                                              Default: 0
238      --task_count=<int>                      Only for C/Python Bindings. Maximum number of␣
     ↪different functions/methods, invoked from the application, that have been selected as tasks
239                                              Default: 50
240      --input_profile=<path>                  Path to the file which stores the input␣
     ↪application profile
241                                              Default: Empty
242      --output_profile=<path>                 Path to the file to store the application profile␣
     ↪at the end of the execution
243                                              Default: Empty
244      --PyObject_serialize=<bool>             Only for Python Binding. Enable the object␣
     ↪serialization to string when possible (true/false).
245                                              Default: false
246      --persistent_worker_c=<bool>            Only for C Binding. Enable the persistent worker␣
     ↪in c (true/false).
247                                              Default: false
248      --enable_external_adaptation=<bool>     Enable external adaptation. This option will␣
     ↪disable the Resource Optimizer.
249                                              Default: false
250      --gen_coredump                          Enable master coredump generation
251                                              Default: false
252      --keep_workingdir                       Do not remove the worker working directory after␣
     ↪the execution
253                                              Default: false
254      --python_interpreter=<string>           Python interpreter to use (python/python2/
     ↪python3).
255                                              Default: python Version:
256      --python_propagate_virtual_environment=<bool>  Propagate the master virtual environment␣
     ↪to the workers (true/false).
257                                                Default: true
258      --python_mpi_worker=<bool>              Use MPI to run the python worker instead of␣
     ↪multiprocessing. (true/false).
259                                              Default: false
260      --python_memory_profile                 Generate a memory profile of the master.
261                                              Default: false
262      --python_worker_cache=<string>          Python worker cache (true/size/false).
263                                              Only for NIO without mpi worker and python >= 3.8.
264                                              Default: false
```

```
265        --wall_clock_limit=<int>                 Maximum duration of the application (in seconds).
266                                                  Default: 0
267        --shutdown_in_node_failure=<bool>        Stop the whole execution in case of Node Failure.
268                                                  Default: false
```

Set environment variables (-e, –env_var)

```
$ pycompss job submit -e MYVAR1 --env MYVAR2=foo APPNAME EXECFILE ARGS
```

Use the -e, –env_var flags to set simple (non-array) environment variables in the remote environment. Or overwrite variables that are defined in the *init* command of the environment.

Submitting Jobs

The command will submit a job and return the Job ID. In order to run a COMPSs program on the local machine we can use the command:

```
$ pycompss job submit -e ComputingUnits=1 -app matmul --num_nodes=2 --exec_time=10 --master_
↪working_dir={COMPS_APP_PATH} --worker_working_dir=local_disk --tracing=false --pythonpath=
↪{COMPS_APP_PATH}/src --lang=python --qos=debug {COMPS_APP_PATH}/src/matmul_files.py 4 4
```

Note: We can also use a macro specific to this CLI in order to use absolute paths: `{COMPS_APP_PATH}` will be resolved by the CLI and replaced with the /absolute/path/to/app on the remote cluster.

Not available

Not available. A remote type environment only accepts submitting jobs for deployed applications. See `Job` tab for more information.

9.2.5 Managing jobs

Once the job is submitted, it can be inspected using the `pycompss job list` command.

The command will list all pending/running jobs subbmited in this environment.

```
$ pycompss job list
        SUCCESS
        19152612        - RUNNING        - COMPSs
```

Every subbmited job that didn't finish yet can be cancelled using the `pycompss job cancel` command.

```
$ pycompss job cancel 19152612 # JOBID
    Job `19152612` cancelled
```

You can also check the status of a particular job with the `pycompss job status` command.

```
$ pycompss job status 19152612 # JOBID
    SUCCESS:RUNNING
```

Also we can query the history of past jobs and we'll get the app name, the environment variables and the enqueue_-compss arguments used to submit the job.

```
$ pycompss job history --job_id 19152612
    Environment Variables: ComputingUnits=1
    Enqueue Args:    --num_nodes=2
                     --exec_time=10
                     --worker_working_dir=local_disk
```

```
                    --tracing=false
                    --lang=python
                    --qos=debug
                    matmul_files.py 4 4
```

9.2.6 Running the COMPSs monitor

Docker

Local

Remote

The COMPSs monitor can be started using the `pycompss monitor start` command. This will start the COMPSs monitoring facility which enables to check the application status while running. Once started, it will show the url to open the monitor in your web browser (i.e. http://127.0.0.1:8080/compss-monitor)

Important: Include the `--monitor=<REFRESH_RATE_MS>` flag in the execution before the binary to be executed.

```
$ pycompss monitor start
$ pycompss run --monitor=1000 -g matmul_files.py 4 4
$ # During the execution, go to the URL in your web browser
$ pycompss monitor stop
```

If running a notebook, just add the monitoring parameter into the COMPSs runtime start call.

Once finished, it is possible to stop the monitoring facility by using the `pycompss monitor stop` command.

The COMPSs monitor can be started using the `pycompss monitor start` command. This will start the COMPSs monitoring facility which enables to check the application status while running. Once started, it will show the url to open the monitor in your web browser (i.e. http://127.0.0.1:8080/compss-monitor)

Important: Include the `--monitor=<REFRESH_RATE_MS>` flag in the execution before the binary to be executed.

```
$ pycompss monitor start
$ pycompss run --monitor=1000 -g matmul_files.py 4 4
$ # During the execution, go to the URL in your web browser
$ pycompss monitor stop
```

If running a notebook, just add the monitoring parameter into the `pycompss jupyter` call.

Once finished, it is possible to stop the monitoring facility by using the `pycompss monitor stop` command.

Not implemented yet.

9.2.7 Running Jupyter notebooks

Docker

Local

Remote

Notebooks can be run using the `pycompss jupyter` command. Run the following snippet from the root of the project:

```
$ cd tutorial_apps/python
$ pycompss jupyter ./notebooks
```

And access interactively to your notebook by opening following the http://127.0.0.1:8888/ URL in your web browser.

Notebooks can be run using the `pycompss jupyter` command. Run the following snippet from the root of the project:

```
$ cd tutorial_apps/python
$ pycompss jupyter ./notebooks
```

A web browser will opened automatically with the notebook.

You could also add any jupyter argument to the command, like for example the port number:

```
$ pycompss jupyter --port 9999 ./notebooks
```

In order to run a jupyter notebook in remote, it must be bound to an already deployed app

Let's deploy another application that contains jupyter notebooks:

```
$ pycompss app deploy synchronization --source_dir tutorial_apps/python/notebooks/syntax/
```

The command will be executed inside the remote directory specified at deployment. The path for the selected application will be automatically resolved and the jupyter server will be started and you'll be promted with the URL of the jupyter web page.

```
$ pycompss jupyter -app synchronization --port 9999
    Job submitted: 19320191
    Waiting for jupyter to start...
    Connecting to jupyter server...
    Connection established. Please use the following URL to connect to the job.
    http://localhost:9999/?token=35199bb8917a97ef2ed0e7a79fbfb6e4c727983bb3a87483
    Ready to work!
    To force quit: CTRL + C
```

How to use Jupyter in MN4 from local machine with PyCOMPSs CLI?

1st Step (to be done in your laptop)

Create the MN4 environment in the PyCOMPSs CLI:

```
pycompss init -n mn4 cluster -l <MN4_USER>@mn1.bsc.es
```

By default, `pycompss` creates the local environment, and since the objective is to run in MN4, this command will create the MN4 environment and set it by default.

Important: This environment will use the `mn1.bsc.es` login node to submit the job, and the notebook will be started within a MN4 compute node.

2nd Step (to be done in your laptop)

Go to the folder where your notebook is in your local machine.

```
cd /path/to/notebook/
```

3rd Step (to be done in your laptop)

Deploy the current folder to MN4 with the following command:

```
pycompss app deploy mynotebook
```

This command will copy the whole current folder into your `$HOME/.COMPSsApps/` folder, and will be used from jupyter notebook.

It will register `mynotebook` name (choose the name that you want), so that it can be used in the next step.

4th Step (to be done in your laptop)

Launch a jupyter job into MN4 using the deployed folder with name `mynotebook` (or the name defined in previous step):

```
pycompss jupyter -app mynotebook --qos=debug --exec_time=20
```

A job will be submitted to MN4 queueing system within the `debug` queue and with a `20 minutes` walltime. **Please, wait for it to start**. It can be checked with `squeue` from MN4 while waiting, and its expected start time with `squeue --start` command.

This job **will deploy the PyCOMPSs infrastructure** in the given nodes.

Once started, the URL to open jupyter from your web browser will automatically appear a few seconds after the job started. Output example:

```
Job submitted: 20480430
Waiting for jupyter to start...
Jupyter started
Connecting to jupyter server...
Connection established. Please use the following URL to connect to the job.
http://localhost:8888/?token=c653b02a899265ad6c9cf075d4882f91d9d372b06132d1fe
Ready to work!
To force quit: CTRL + C
```

5th Step (to be done in your laptop)

Open the given URL (*in some consoles with CTRL + left click*) in your local web browser and you can start working with the notebook.

Inside the notebook, PyCOMPSs must be imported, its runtime started, tasks defined, etc.

Please, check the documentation to get help and examples:

- PyCOMPSs programming model
- Jupyter API (pycompss start, stop, etc.)
- Sample notebooks
- Tutorial

Caution: If the walltime of the job is reached, the job will be killed by the queuing system and the notebook will stop working.

6th Step (to be done in your laptop)

Once finished working with the notebook, press `CTRL+C` in the console where you launched the `pycompss jupyter` command. This will trigger the job cancellation.

9.2.8 Generating the task graph

COMPSs is able to produce the task graph showing the dependencies that have been respected. In order to producee it, include the `--graph` flag in the execution command:

Docker

Local

Remote

```
$ cd tutorial_apps/python/simple/src
$ pycompss init docker
$ pycompss run --graph simple.py 1
```

Once the application finishes, the graph will be stored into the `.COMPSs\app_name_XX\monitor\complete_graph.dot` file. This dot file can be converted to pdf for easier visualilzation through the use of the `gengraph` parameter:

```
$ pycompss gengraph .COMPSs/simple.py_01/monitor/complete_graph.dot
```

The resulting pdf file will be stored into the `.COMPSs\app_name_XX\monitor\complete_graph.pdf` file, that is, the same folder where the dot file is.

```
$ cd tutorial_apps/python/simple/src
$ pycompss run --graph simple.py 1
```

Once the application finishes, the graph will be stored into the `~\.COMPSs\app_name_XX\monitor\complete_-graph.dot` file. This dot file can be converted to pdf for easier visualilzation through the use of the `gengraph` parameter:

```
$ pycompss gengraph ~/.COMPSs/simple.py_01/monitor/complete_graph.dot
```

The resulting pdf file will be stored into the `~\.COMPSs\app_name_XX\monitor\complete_graph.pdf` file, that is, the same folder where the dot file is.

Not implemented yet!

9.2.9 Tracing applications or notebooks

COMPSs is able to produce tracing profiles of the application execution through the use of EXTRAE. In order to enable it, include the `--tracing` flag in the execution command:

```
$ cd python/matmul_files/src
$ pycompss run --tracing matmul_files.py 4 4
```

If running a notebook, just add the tracing parameter into `pycompss jupyter` call.

Once the application finishes, the trace will be stored into the `~\.COMPSs\app_name_XX\trace` folder. It can then be analysed with Paraver.

9.2.10 Adding more nodes

Docker

Local

Remote

Note: Adding more nodes is still in beta phase. Please report issues, suggestions, or feature requests on Github.

To add more computing nodes, you can either let docker create more workers for you or manually create and config a custom node.

For docker just issue the desired number of workers to be added. For example, to add 2 docker workers:

```
$ pycompss components add worker 2
```

You can check that both new computing nodes are up with:

```
$ pycompss components list
```

If you want to add a custom node it needs to be reachable through ssh without user. Moreover, pycompss will try to copy the `working_dir` there, so it needs write permissions for the scp.

For example, to add the local machine as a worker node:

```
$ pycompss components add worker '127.0.0.1:6'
```

- '127.0.0.1': is the IP used for ssh (can also be a hostname like 'localhost' as long as it can be resolved).
- '6': desired number of available computing units for the new node.

Important: Please be aware** that `pycompss components` will not list your custom nodes because they are not docker processes and thus it can't be verified if they are up and running.

Environment not compatible with this feature.

Environment not compatible with this feature.

9.2.11 Removing existing nodes

Docker

Local

Remote

Note: Removing nodes is still in beta phase. Please report issues, suggestions, or feature requests on Github.

For docker just issue the desired number of workers to be removed. For example, to remove 2 docker workers:

```
$ pycompss components remove worker 2
```

You can check that the workers have been removed with:

```
$ pycompss components list
```

If you want to remove a custom node, you just need to specify its IP and number of computing units used when defined.

```
$ pycompss components remove worker '127.0.0.1:6'
```

Environment not compatible with this feature.

Environment not compatible with this feature.

Chapter 10

PyCOMPSs Notebooks

This section contains all PyCOMPSs related tutorial notebooks (sources available in https://github.com/bsc-wdc/notebooks).

It is divided into three main folders:

1. **Syntax**: Contains the main tutorial notebooks. They cover the syntax and main functionalities of Py-COMPSs.
2. **Hands-On**: Contains example applications and hands-on exercises.
3. **Demos**: Contains demonstration notebooks.

10.1 Syntax

Here you will find the syntax notebooks used in the tutorials.

10.1.1 Basics of programming with PyCOMPSs

In this example we will see basics of programming with PyCOMPSs: - Runtime start - Task definition - Task invocation - Runtime stop

10.1.1.1 Let's get started with a simple example

First step

- Import the PyCOMPSs library

```
[1]: import pycompss.interactive as ipycompss
```

Second step

- Initialize COMPSs runtime. Parameters indicates if the execution will generate task graph, tracefile, monitor interval and debug information.

```
[2]: import os
     if 'BINDER_SERVICE_HOST' in os.environ:
         ipycompss.start(graph=True,
                         project_xml='../xml/project.xml',
                         resources_xml='../xml/resources.xml')
```

(continues on next page)

```
else:
    ipycompss.start(graph=True, monitor=1000)  # debug=True, trace=True
```

```
**********************************************************
*************** PyCOMPSs Interactive ******************
**********************************************************
*         .-~~-.--.              _____          ___      *
*        :         )            |____  \        /   |      *
*  .~ ~ -.\       /.- ~~ .        __) |        /_/| |      *
*  >       `.   .'       <       |__  |          | |       *
*  (         .- -.         )   ___) |    _       | |       *
*   `- -.-~  `- -'  ~-.- -'    |_____/   |_|     |_|       *
*     (        :        )          _ _ .-:               *
*      ~--.    :    .--~        .-~  .-~  }              *
*          ~-.-^-.-~ \_      .~  .-~   .~               *
*                   \ \ '     \ '_ _ -~                 *
*                    \`.\`.    //                       *
*           . - ~ ~-.__\`.\`-.//                        *
*       .-~   . - ~  }~ ~ ~-.~-.                        *
*     .' .-~      .-~       :/~-.~-./:                  *
*    /_~_ _ . - ~                 ~-.~-._               *
*                                     ~-.<              *
**********************************************************
* - Starting COMPSs runtime...                           *
* - Log path : /home/user/.COMPSs/Interactive_01/
* - PyCOMPSs Runtime started... Have fun!                *
**********************************************************
```

Third step

- Import task module before annotating functions or methods

```
[3]: from pycompss.api.task import task
```

Fourth step

- Declare functions and decorate with @task those that should be tasks

```
[4]: @task(returns=int)
     def square(val1):
         return val1 * val1
```

```
[5]: @task(returns=int)
     def add(val2, val3):
         return val2 + val3
```

```
[6]: @task(returns=int)
     def multiply(val1, val2):
         return val1 * val2
```

Fifth step

- Invoke tasks

```
[7]: a = square(2)

     Found task: square
```

```
[8]: b = add(a, 4)

     Found task: add
```

```
[9]: c = multiply(b, 5)

     Found task: multiply
```

Sixth step (last)

- Stop COMPSs runtime. All data can be synchronized in the main program .

```
[10]: ipycompss.stop(sync=True)

      *********************************************************
      *************** STOPPING PyCOMPSs ******************
      *********************************************************
      Checking if any issue happened.
      Synchronizing all future objects left on the user scope.
      Found a future object: a
      Found a future object: b
      Found a future object: c
      *********************************************************
```

```
[11]: print("Results after stopping PyCOMPSs: ")
      print("a: %d" % a)
      print("b: %d" % b)
      print("c: %d" % c)

      Results after stopping PyCOMPSs:
      a: 4
      b: 8
      c: 40
```

10.1.2 PyCOMPSs: Synchronization

In this example we will see how to synchronize with PyCOMPSs.

10.1.2.1 Import the PyCOMPSs library

```
[1]: import pycompss.interactive as ipycompss
```

10.1.2.2 Start the runtime

Initialize COMPSs runtime Parameters indicates if the execution will generate task graph, tracefile, monitor interval and debug information.

```
[2]: import os
     if 'BINDER_SERVICE_HOST' in os.environ:
         ipycompss.start(graph=True, debug=False,
                         project_xml='../xml/project.xml',
                         resources_xml='../xml/resources.xml')
     else:
         ipycompss.start(graph=True, monitor=1000, trace=False)
```

```
********************************************************
**************** PyCOMPSs Interactive ******************
********************************************************
*           .-~~-.--.                    _____        ___        *
*          :         )                  |____   \      /   |       *
*    .~ ~ -.\       /.- ~~ .            __) |        /_/| |        *
*    >            `.   .'          <      |__  |          | |       *
*   (              .- -.            )    ___) |    _       | |      *
*    `- -.-~  `- -'  ~-.- -'      |_____/   |_|      |_|      *
*     (           :          )           _ _ .-:            *
*      ~--.    :    .--~            .-~  .-~  }          *
*          ~-.-^-.-~ \_          .~  .-~    .~          *
*                   \ \ '      \ '_ _ -~            *
*                    \`.\`.    //                    *
*            . - ~ ~-.__\`.\`-.//                    *
*        .-~    . - ~  }~ ~ ~-.~-.                    *
*       .' .-~       .-~       :/~-.~-./:            *
*      /_~_ _ . - ~                 ~-.~-._            *
*                                         ~-.<        *
********************************************************
* - Starting COMPSs runtime...                          *
* - Log path : /home/user/.COMPSs/Interactive_02/       *
* - PyCOMPSs Runtime started... Have fun!               *
********************************************************
```

Importing task and parameter modules

Import task module before annotating functions or methods

```
[3]: from pycompss.api.task import task
     from pycompss.api.parameter import *
     from pycompss.api.api import compss_wait_on
```

10.1.2.3 Declaring tasks

Declare functions and decorate with @task those that should be tasks

```
[4]: @task(returns=int)
     def square(val1):
         return val1 * val1
```

```
[5]: @task(returns=int)
     def add(val2, val3):
         return val2 + val3
```

```
[6]: @task(returns=int)
     def multiply(val1, val2):
         return val1 * val2
```

10.1.2.4 Invoking tasks

```
[7]: a = square(2)
```

```
Found task: square
```

```
[8]: b = add(a, 4)
```

```
Found task: add
```

```
[9]: c = multiply (b, 5)
```

```
Found task: multiply
```

Accessing data outside tasks requires synchronization

```
[10]: c = compss_wait_on(c)
```

```
[11]: c = c + 1
```

```
[12]: print("a: %s" % a)
      print("b: %s" % b)
      print("c: %d" % c)
```

```
a: <pycompss.runtime.management.classes.Future object at 0x7f20490ba650>
b: <pycompss.runtime.management.classes.Future object at 0x7f20490ba2c0>
c: 41
```

```
[13]: a = compss_wait_on(a)
```

```
[14]: print("a: %d" % a)
```

```
a: 4
```

10.1.2.5 Stop the runtime

```
[15]: ipycompss.stop(sync=True)
```

```
*********************************************************
*************** STOPPING PyCOMPSs ******************
*********************************************************
Checking if any issue happened.
Synchronizing all future objects left on the user scope.
Found a future object: b
*********************************************************
```

```
[16]: print("Results after stopping PyCOMPSs: ")
      print("a: %d" % a)
      print("b: %d" % b)
      print("c: %d" % c)
```

```
Results after stopping PyCOMPSs:
a: 4
b: 8
c: 41
```

10.1.3 PyCOMPSs: Using objects, lists, and synchronization

In this example we will see how classes and objects can be used from PyCOMPSs, and that class methods can become tasks.

10.1.3.1 Import the PyCOMPSs library

```
[1]: import pycompss.interactive as ipycompss
```

10.1.3.2 Start the runtime

Initialize COMPSs runtime Parameters indicates if the execution will generate task graph, tracefile, monitor interval and debug information.

```
[2]: import os
     if 'BINDER_SERVICE_HOST' in os.environ:
         ipycompss.start(graph=True, debug=True,
                         project_xml='../xml/project.xml',
                         resources_xml='../xml/resources.xml')
     else:
         ipycompss.start(graph=True, monitor=1000, debug=True)
```

```
*********************************************************
*************** PyCOMPSs Interactive ******************
*********************************************************
*          .-~~-.--.              _____          ---        *
*         :         )           |_____ \        /   |      *
*   .~ ~ -.\       /.- ~~ .         __) |      /_/| |      *
*   >       `.   .'       <        |__   |          | |      *
*  (         .- -.         )      ____) |    _      | |      *
*   `- -.-~  `- -'  ~-.- -'      |_____/   |_|    |_|      *
*      (         :         )            _ _ .-:            *
*       ~--.    :    .--~         .-~  .-~  }            *
```

```
*           ~-.-^-.-~ \_          .~   .-~    .~
*                  \ \ '        \ '_ _ -~
*                   \`.\`.     //
*          . - ~ ~-.__\`.\`-.//
*      .-~    . - ~  }~ ~ ~-.~-.
*    .' .-~      .-~      :/~-.~-./:
*   /_~_ _ . - ~             ~-.~-._
*                                ~-.<
**********************************************************
* - Starting COMPSs runtime...                          *
* - Log path : /home/user/.COMPSs/Interactive_03/
* - PyCOMPSs Runtime started... Have fun!               *
**********************************************************
```

10.1.3.3 Importing task and arguments directionality modules

Import task module before annotating functions or methods

```
[3]: from pycompss.api.api import compss_barrier
     from pycompss.api.api import compss_wait_on
```

10.1.3.4 Declaring a class

```
[4]: %%writefile my_shaper.py

     from pycompss.api.task import task
     from pycompss.api.parameter import IN

     class Shape(object):
         def __init__(self,x,y):
             self.x = x
             self.y = y

         @task(returns=int)
         def area(self):
             return self.x * self.y

         @task(returns=int)
         def perimeter(self):
             return 2 * self.x + 2 * self.y

         def describe(self,text):
             self.description = text

         @task()
         def scaleSize(self,scale):
             self.x = self.x * scale
             self.y = self.y * scale

         @task(target_direction=IN)
         def infoShape(self):
             print('Shape x=', self.x, 'y= ', self.y)
```

```
Writing my_shaper.py
```

10.1.3.5 Invoking tasks

```
[5]: from my_shaper import Shape
```

```
[6]: my_shapes = []
     my_shapes.append(Shape(100,45))
     my_shapes.append(Shape(50,50))
```

```
[7]: all_areas = []
```

```
[8]: for this_shape in my_shapes:
         all_areas.append(this_shape.area())
```

```
[9]: # Need it if we want to synchonize nested objects
     all_areas = compss_wait_on(all_areas)
     print(all_areas)
```

```
[4500, 2500]
```

```
[10]: rectangle = Shape(200,25)
      rectangle.scaleSize(5)
      area_rectangle = rectangle.area()
      rectangle = compss_wait_on(rectangle)
      print('X = %d' % rectangle.x)
      area_rectangle = compss_wait_on(area_rectangle)
      print('Area = %d' % area_rectangle)
```

```
X = 1000
Area = 125000
```

```
[11]: all_perimeters=[]
      my_shapes.append(rectangle)
      for this_shape in my_shapes:
          this_shape.infoShape()
          all_perimeters.append(this_shape.perimeter())
```

```
[12]: all_perimeters = compss_wait_on(all_perimeters)
      print(all_perimeters)
```

```
[290, 200, 2250]
```

10.1.3.6 Stop the runtime

```
[13]: ipycompss.stop(sync=True)
```

```
********************************************************
*************** STOPPING PyCOMPSs ******************
********************************************************
Checking if any issue happened.
Synchronizing all future objects left on the user scope.
Found a list to synchronize: my_shapes
Found a list to synchronize: all_areas
Found a list to synchronize: all_perimeters
********************************************************
```

10.1.4 PyCOMPSs: Using objects, lists, and synchronization

In this example we will see how classes and objects can be used from PyCOMPSs, and that class methods can become tasks.

10.1.4.1 Import the PyCOMPSs library

```
[1]: import pycompss.interactive as ipycompss
```

10.1.4.2 Start the runtime

Initialize COMPSs runtime Parameters indicates if the execution will generate task graph, tracefile, monitor interval and debug information.

```
[2]: import os
     if 'BINDER_SERVICE_HOST' in os.environ:
         ipycompss.start(graph=True, debug=True,
                         project_xml='../xml/project.xml',
                         resources_xml='../xml/resources.xml')
     else:
         ipycompss.start(graph=True, monitor=1000, debug=True, trace=False)
```

```
********************************************************
**************** PyCOMPSs Interactive ******************
********************************************************
*          .-~~-.--.                 _____        ___       *
*         :         )              |____   \      /   |       *
*   .~ ~ -.\       /.- ~~ .           __) |      /_/| |       *
*   >       `.   .'       <          |__  |         | |       *
*  (         .- -.         )       ___) |     _     | |       *
*   `- -.-~  `- -'  ~-.- -'       |_____/   |_|    |_|       *
*     (        :        )           _ _ .-:                   *
*      ~--.    :    .--~         .-~  .-~  }                  *
*          ~-.-^-.-~ \_      .~  .-~   .~                     *
*                   \ \ '     \ '_ _ -~                       *
*                    \`.\`.    //                             *
*            . - ~ ~-.__\`.\`-.//                             *
*        .-~   . - ~  }~ ~ ~-.~-.                             *
*      .' .-~      .-~       :/~-.~-./:                       *
*     /_~_ _ . - ~                 ~-.~-._                    *
*                                       ~-.<                  *
********************************************************
* - Starting COMPSs runtime...                         *
* - Log path : /home/user/.COMPSs/Interactive_04/
* - PyCOMPSs Runtime started... Have fun!              *
********************************************************
```

10.1.4.3 Importing task and arguments directionality modules

Import task module before annotating functions or methods

```
[3]: from pycompss.api.api import compss_barrier
     from pycompss.api.api import compss_wait_on
     from pycompss.api.task import task
```

10.1.4.4 Declaring a class

```
[4]: %%writefile my_shaper.py

     from pycompss.api.task import task
     from pycompss.api.parameter import IN

     class Shape(object):
         def __init__(self,x,y):
             self.x = x
             self.y = y
             description = "This shape has not been described yet"

         @task(returns=int)
         def area(self):
             return self.x * self.y

         @task(returns=int)
         def perimeter(self):
             return 2 * self.x + 2 * self.y

         def describe(self,text):
             self.description = text

         @task()
         def scaleSize(self,scale):
             self.x = self.x * scale
             self.y = self.y * scale

         @task(target_direction=IN)
         def infoShape(self):
             print('Shape x=', self.x, 'y= ', self.y)
```

```
Overwriting my_shaper.py
```

```
[5]: @task(returns=int)
     def addAll(*mylist):
         sum = 0
         for ll in mylist:
             sum = sum + ll
         return sum
```

10.1.4.5 Invoking tasks

```
[6]: from my_shaper import Shape
```

```
[7]: my_shapes = []
     my_shapes.append(Shape(100,45))
     my_shapes.append(Shape(50,50))
     my_shapes.append(Shape(10,100))
     my_shapes.append(Shape(20,30))
```

```
[8]: all_areas = []
```

```
[9]: for this_shape in my_shapes:
         all_areas.append(this_shape.area())
```

```
[10]: # Need it if we want to synchonize nested objects
      all_areas = compss_wait_on(all_areas)
      print(all_areas)
```

```
[4500, 2500, 1000, 600]
```

```
[11]: rectangle = Shape(200,25)
      rectangle.scaleSize(5)
      area_rectangle = rectangle.area()
      rectangle = compss_wait_on(rectangle)
      print('X = %d' % rectangle.x)
      area_rectangle = compss_wait_on(area_rectangle)
      print('Area = %d' % area_rectangle)
```

```
X = 1000
Area = 125000
```

```
[12]: all_perimeters=[]
      my_shapes.append(rectangle)
      for this_shape in my_shapes:
          this_shape.infoShape()
          all_perimeters.append(this_shape.perimeter())
```

```
[13]: # all_perimeters = compss_wait_on(all_perimeters)
      # print all_perimeters
```

```
[14]: mysum = addAll(*all_perimeters)
      mysum = compss_wait_on(mysum)
      print(mysum)
```

```
Task definition detected.
Found task: addAll
3060
```

10.1.4.6 Stop the runtime

```
[15]: ipycompss.stop(sync=True)
```

```
*********************************************************
*************** STOPPING PyCOMPSs ******************
*********************************************************
Checking if any issue happened.
Synchronizing all future objects left on the user scope.
Found a list to synchronize: my_shapes
Found a list to synchronize: all_areas
Found a list to synchronize: all_perimeters
*********************************************************
```

10.1.5 PyCOMPSs: Using objects, lists, and synchronization. Using collections.

In this example we will see how classes and objects can be used from PyCOMPSs, and that class methods can become tasks. The example also illustrates the use of collections

10.1.5.1 Import the PyCOMPSs library

```
[1]: import pycompss.interactive as ipycompss
```

10.1.5.2 Start the runtime

Initialize COMPSs runtime Parameters indicates if the execution will generate task graph, tracefile, monitor interval and debug information.

```
[2]: import os
     if 'BINDER_SERVICE_HOST' in os.environ:
         ipycompss.start(graph=True, debug=True,
                         project_xml='../xml/project.xml',
                         resources_xml='../xml/resources.xml')
     else:
         ipycompss.start(graph=True, monitor=1000, debug=True, trace=False)
```

```
*********************************************************
*************** PyCOMPSs Interactive ******************
*********************************************************
*          .-~~-.--.              _____          ___        *
*         :         )           |____ \        /   |       *
*   .~ ~ -.\       /.- ~~ .        __) |      /_/| |       *
*   >       `.   .'       <       |__  |          | |       *
*   (         .- -.         )    ___) |    _      | |       *
*   `- -.-~  `- -'  ~-.- -'     |_____/   |_|    |_|       *
*     (        :       )            _ _ .-:           *
*      ~--.    :    .--~         .-~ .-~  }          *
*          ~-.-^-.-~ \_        .~  .-~   .~          *
*                   \ \ '     \ '_ _ -~          *
*                    \`.\`.    //                   *
*            . - ~ ~-.__\`.\`-.//                    *
*        .-~   . - ~  }~ ~ ~-.~-.                    *
*      .' .-~      .-~       :/~-.~-./:              *
*     /_~_ _ . - ~                 ~-.~-._           *
*                                      ~-.<          *
```

```
*******************************************************
* - Starting COMPSs runtime...                        *
* - Log path : /home/user/.COMPSs/Interactive_05/
* - PyCOMPSs Runtime started... Have fun!             *
*******************************************************
```

10.1.5.3 Importing task and arguments directionality modules

Import task module before annotating functions or methods

```
[3]: from pycompss.api.api import compss_barrier
     from pycompss.api.api import compss_wait_on
     from pycompss.api.task import task
     from pycompss.api.parameter import *
```

10.1.5.4 Declaring a class

```
[4]: %%writefile my_shaper.py

     from pycompss.api.task import task
     from pycompss.api.parameter import IN

     class Shape(object):
         def __init__(self,x,y):
             self.x = x
             self.y = y
             description = "This shape has not been described yet"

         @task(returns=int, target_direction=IN)
         def area(self):
             import time
             time.sleep(4)
             return self.x * self.y

         @task()
         def scaleSize(self,scale):
             import time
             time.sleep(4)
             self.x = self.x * scale
             self.y = self.y * scale

         @task(returns=int, target_direction=IN)
         def perimeter(self):
             import time
             time.sleep(4)
             return 2 * self.x + 2 * self.y

         def describe(self,text):
             self.description = text

         @task(target_direction=IN)
         def infoShape(self):
             import time
```

```
        time.sleep(1)
        print('Shape x=', self.x, 'y= ', self.y)
```

Overwriting my_shaper.py

```python
[5]: #Operations with collections: previous to release 2.5
@task(returns=1)
def addAll(*mylist):
    import time
    time.sleep(1)
    sum = 0
    for ll in mylist:
        sum = sum + ll
    return sum
```

```python
[6]: @task(returns=int, mylist=COLLECTION_IN)
def addAll_C(mylist):
    import time
    time.sleep(4)
    sum = 0
    for ll in mylist:
        sum = sum + ll
    return sum
```

```python
[7]: @task(returns=2, mylist=COLLECTION_IN, my_otherlist=COLLECTION_IN)
def addAll_C2(mylist, my_otherlist):
    import time
    time.sleep(4)
    sum = 0
    sum2 = 0
    for ll in mylist:
        sum = sum + ll
    for jj in my_otherlist:
        sum2 = sum2 + jj
    return sum, sum2
```

```python
[8]: @task(mylist=COLLECTION_INOUT)
def scale_all(mylist, scale):
    import time
    time.sleep(4)
    for ll in mylist:
        ll.x = ll.x * scale
        ll.y = ll.y * scale
```

10.1.5.5 Invoking tasks

```python
[9]: from my_shaper import Shape
```

```python
[10]: my_shapes = []
my_shapes.append(Shape(100,45))
my_shapes.append(Shape(50,50))
my_shapes.append(Shape(10,100))
my_shapes.append(Shape(20,30))
```

```
[11]: all_areas = []
```

```
[12]: for this_shape in my_shapes:
          all_areas.append(this_shape.area())
```

10.1.5.6 Synchronizing results from tasks

```
[13]: all_areas = compss_wait_on(all_areas)
      print(all_areas)
```

```
[4500, 2500, 1000, 600]
```

```
[14]: rectangle = Shape(200,25)
      rectangle.scaleSize(5)
      area_rectangle = rectangle.area()
      rectangle = compss_wait_on(rectangle)
      print('X =', rectangle.x)
      area_rectangle = compss_wait_on(area_rectangle)
      print('Area =', area_rectangle)
```

```
X = 1000
Area = 125000
```

10.1.5.7 Accessing data in collections

```
[15]: all_perimeters = []
      my_shapes.append(rectangle)
      for this_shape in my_shapes:
          all_perimeters.append(this_shape.perimeter())
```

```
[16]: mysum = addAll_C(all_perimeters)
      mysum = compss_wait_on(mysum)
      print(mysum)
```

```
Task definition detected.
Found task: addAll_C
3060
```

```
[17]: # Previous version without collections
      # mysum = addAll(*all_perimeters)
      # mysum = compss_wait_on(mysum)
      # print(mysum)
```

10.1.5.8 Accessing two collections

```
[18]: all_perimeters = []
      all_areas = []
      for this_shape in my_shapes:
          all_perimeters.append(this_shape.perimeter())
          all_areas.append(this_shape.area())
```

```
[19]: [my_per, my_area] = addAll_C2(all_perimeters, all_areas)
      [my_per, my_area] = compss_wait_on([my_per, my_area])
      print([my_per, my_area])
```

```
Task definition detected.
Found task: addAll_C2
[3060, 133600]
```

10.1.5.9 Scattering data from a collection

```
[20]: scale_all(my_shapes,2)
      scaled_areas=[]
      for this_shape in my_shapes:
          scaled_areas.append(this_shape.area())

      scaled_areas = compss_wait_on(scaled_areas)
      print(scaled_areas)
```

```
Task definition detected.
Found task: scale_all
[18000, 10000, 4000, 2400, 500000]
```

10.1.5.10 Stop the runtime

```
[21]: ipycompss.stop(sync=True)
```

```
*********************************************************
*************** STOPPING PyCOMPSs *******************
*********************************************************
Checking if any issue happened.
Synchronizing all future objects left on the user scope.
Found a list to synchronize: my_shapes
Found a list to synchronize: all_areas
Found a list to synchronize: all_perimeters
Found a list to synchronize: scaled_areas
*********************************************************
```

10.1.6 PyCOMPSs: Using objects, lists, and synchronization. Using dictionary.

In this example we will see how classes and objects can be used from PyCOMPSs, and that class methods can become tasks. The example also illustrates the use of dictionary

10.1.6.1 Import the PyCOMPSs library

```
[1]: import pycompss.interactive as ipycompss
```

10.1.6.2 Start the runtime

Initialize COMPSs runtime Parameters indicates if the execution will generate task graph, tracefile, monitor interval and debug information.

```
[2]: import os
     if 'BINDER_SERVICE_HOST' in os.environ:
         ipycompss.start(graph=True, debug=True,
                         project_xml='../xml/project.xml',
                         resources_xml='../xml/resources.xml')
     else:
         ipycompss.start(graph=True, monitor=1000, debug=True, trace=False)
```

```
********************************************************
*************** PyCOMPSs Interactive ******************
********************************************************
*          .-~~-.--.                _____      ___       *
*         :         )              |____  \    /  |      *
*   .~ ~ -.\       /.- ~~ .          __) |      /_/| |     *
*   >       `.   .'       <         |__  |          | |     *
*  (         .- -.         )       ____) |    _    | |     *
*   `- -.-~  `- -'  ~-.- -'        |_____/   |_|    |_|     *
*     (        :        )             _ _ .-:      *
*      ~--.    :    .--~          .-~  .-~  }     *
*          ~-.-^-.-~ \_        .~  .-~   .~     *
*                   \ \ '     \ '_ _ -~        *
*                    \`.\`.    //              *
*           . - ~ ~-.__\`.\`-.//               *
*       .-~   . - ~  }~ ~ ~-.-.                *
*     .' .-~      .-~       :/~-.~-./:         *
*    /_~_ _ . - ~                 ~-.~-._     *
*                                    ~-.<     *
********************************************************
* - Starting COMPSs runtime...                          *
* - Log path : /home/user/.COMPSs/Interactive_06/
* - PyCOMPSs Runtime started... Have fun!               *
********************************************************
```

10.1.6.3 Importing task and arguments directionality modules

Import task module before annotating functions or methods

```
[3]: from pycompss.api.api import compss_barrier
     from pycompss.api.api import compss_wait_on
     from pycompss.api.task import task
     from pycompss.api.parameter import *
```

10.1.6.4 Declaring a class

```
[4]: %%writefile my_shaper.py

     from pycompss.api.task import task
     from pycompss.api.parameter import IN

     class Shape(object):
         def __init__(self,x,y):
             self.x = x
             self.y = y
             description = "This shape has not been described yet"

         @task(returns=int, target_direction=IN)
         def area(self):
             import time
             time.sleep(4)
             return self.x * self.y

         @task()
         def scaleSize(self,scale):
             import time
             time.sleep(4)
             self.x = self.x * scale
             self.y = self.y * scale

         @task(returns=int, target_direction=IN)
         def perimeter(self):
             import time
             time.sleep(4)
             return 2 * self.x + 2 * self.y

         def describe(self,text):
             self.description = text

         @task(target_direction=IN)
         def infoShape(self):
             import time
             time.sleep(1)
             print('Shape x=', self.x, 'y= ', self.y)
```

```
Overwriting my_shaper.py
```

```
[5]: @task(returns=int, mydict = DICTIONARY_IN)
     def addAll(mydict):
         import time
         time.sleep(4)
         sum = 0
         for key, value in mydict.items():
             sum = sum + value
         return sum
```

```
[6]: @task(returns=2, mydict=DICTIONARY_IN, my_otherdict=DICTIONARY_IN)
     def addAll_2(mydict, my_otherdict):
         import time
         time.sleep(4)
         sum = 0
```

```
        sum2 = 0
        for key, value in mydict.items():
            sum = sum + value
        for key2, value2 in my_otherdict.items():
            sum2 = sum2 + value2
        return sum, sum2
```

```
[7]: @task(mydict=DICTIONARY_INOUT)
     def scale_all(mydict, scale):
         import time
         time.sleep(4)
         for key, value in mydict.items():
             mydict[key].x = value.x * scale
             mydict[key].y = value.y * scale
```

10.1.6.5 Invoking tasks

```
[8]: from my_shaper import Shape
```

```
[9]: my_shapes = {}
     my_shapes["rectangle"] = Shape(100,45)
     my_shapes["square"] = Shape(50,50)
     my_shapes["long_rectangle"] = Shape(10,100)
     my_shapes["small_rectangle"] = Shape(20,30)
```

```
[10]: all_areas = {}
```

```
[11]: for key, value in my_shapes.items():
          all_areas[key] = value.area()
```

10.1.6.6 Synchronizing results from tasks

```
[12]: all_areas = compss_wait_on(all_areas)
      print(all_areas)
```

```
{'rectangle': 4500, 'square': 2500, 'long_rectangle': 1000, 'small_rectangle': 600}
```

```
[13]: rectangle = Shape(200,25)
      rectangle.scaleSize(5)
      area_rectangle = rectangle.area()
      rectangle = compss_wait_on(rectangle)
      print('X =', rectangle.x)
      area_rectangle = compss_wait_on(area_rectangle)
      print('Area =', area_rectangle)
```

```
X = 1000
Area = 125000
```

10.1.6.7 Accessing data in collections

```
[14]: all_perimeters = {}
      my_shapes["new_shape"] = rectangle
      for key, value in my_shapes.items():
          all_perimeters[key] = value.perimeter()
```

```
[15]: mysum = addAll(all_perimeters)
      mysum = compss_wait_on(mysum)
      print(mysum)
```

```
Task definition detected.
Found task: addAll
3060
```

10.1.6.8 Accessing two collections

```
[16]: all_perimeters = {}
      all_areas = {}
      for key, value in my_shapes.items():
          all_perimeters[key] = value.perimeter()
          all_areas[key] = value.area()
```

```
[17]: [my_per, my_area] = addAll_2(all_perimeters, all_areas)
      [my_per, my_area] = compss_wait_on([my_per, my_area])
      print([my_per, my_area])
```

```
Task definition detected.
Found task: addAll_2
[3060, 133600]
```

10.1.6.9 Scattering data from a collection

```
[18]: scale_all(my_shapes, 2)
      scaled_areas = {}
      for key, value in my_shapes.items():
          scaled_areas[key] = value.area()

      scaled_areas = compss_wait_on(scaled_areas)
      print(scaled_areas)
```

```
Task definition detected.
Found task: scale_all
{'rectangle': 18000, 'square': 10000, 'long_rectangle': 4000, 'small_rectangle': 2400, 'new_
→shape': 500000}
```

10.1.6.10 Stop the runtime

```
[19]: ipycompss.stop(sync=True)
```

```
*********************************************************
*************** STOPPING PyCOMPSs ******************
*********************************************************
Checking if any issue happened.
Synchronizing all future objects left on the user scope.
*********************************************************
```

10.1.7 PyCOMPSs: Using objects, lists, and synchronization. Managing fault-tolerance.

In this example we will see how classes and objects can be used from PyCOMPSs, and that class methods can become tasks. The example also illustrates the current fault-tolerance management provided by the runtime.

10.1.7.1 Import the PyCOMPSs library

```
[1]: import pycompss.interactive as ipycompss
```

10.1.7.2 Start the runtime

Initialize COMPSs runtime Parameters indicates if the execution will generate task graph, tracefile, monitor interval and debug information.

```
[2]: import os
     if 'BINDER_SERVICE_HOST' in os.environ:
         ipycompss.start(graph=True, debug=False,
                         project_xml='../xml/project.xml',
                         resources_xml='../xml/resources.xml')
     else:
         ipycompss.start(graph=True, monitor=1000, trace=False, debug=False)
```

```
*********************************************************
*************** PyCOMPSs Interactive ******************
*********************************************************
*          .-~~-.--.                _____          ___       *
*         :         )              |____ \        /   |       *
*   .~ ~ -.\       /.- ~~ .           __) |      /_/| |       *
*   >       `.   .'       <          |__  |         | |       *
*   (         .- -.         )        ___) |   _      | |       *
*    `- -.-~  `- -'  ~-.- -'        |_____/  |_|     |_|      *
*      (         :         )           _ _ .-:                 *
*       ~--.    :    .--~          .-~  .-~  }                 *
*           ~-.-^-.-~ \_         .~  .-~   .~                  *
*                    \ \ '     \ '_ _ -~                       *
*                     \`.\`.    //                             *
*            . - ~ ~-.__\`.\`-.//                              *
*        .-~   . - ~  }~ ~ ~-.~-.                              *
*      .' .-~      .-~       :/~-.~-./:                        *
*     /_~_ _ . - ~                 ~-.~-._                     *
*                                      ~-.<                    *
*********************************************************
```

```
* - Starting COMPSs runtime...                        *
* - Log path : /home/user/.COMPSs/Interactive_07/
* - PyCOMPSs Runtime started... Have fun!             *
******************************************************
```

10.1.7.3 Importing task and arguments directionality modules

Import task module before annotating functions or methods

```
[3]: from pycompss.api.api import compss_barrier
     from pycompss.api.api import compss_wait_on
     from pycompss.api.task import task
     from pycompss.api.parameter import *
```

10.1.7.4 Declaring a class

```
[4]: %%writefile my_shaper.py

     from pycompss.api.task import task
     from pycompss.api.on_failure import on_failure
     from pycompss.api.parameter import IN
     import sys

     class Shape(object):
         def __init__(self,x,y):
             self.x = x
             self.y = y
             description = "This shape has not been described yet"

         @task(returns=int, target_direction=IN)
         def area(self):
             return self.x * self.y

         @task()
         def scaleSize(self,scale):
             self.x = self.x * scale
             self.y = self.y * scale

         # management='IGNORE' | 'RETRY' | 'FAIL' | 'CANCEL_SUCCESSORS'
         @on_failure(management="CANCEL_SUCCESSORS")
         @task()
         def downScale(self,scale):
             if (scale <= 0):
                 sys.exit(1)
             else:
                 self.x = self.x/scale
                 self.y = self.y/scale

         @task(returns=int, target_direction=IN)
         def perimeter(self):
             return 2 * self.x + 2 * self.y

         def describe(self,text):
```

```
        self.description = text

    @task(target_direction=IN)
    def infoShape(self):
        print('Shape x=', self.x, 'y= ', self.y)
```

```
Overwriting my_shaper.py
```

10.1.7.5 Invoking tasks

```
[5]: from my_shaper import Shape
```

```
[6]: my_shapes = []
     my_shapes.append(Shape(100,45))
     my_shapes.append(Shape(50,50))
     my_shapes.append(Shape(10,100))
     my_shapes.append(Shape(20,30))
     my_shapes.append(Shape(200,25))
```

```
[7]: all_perimeters = []
```

```
[8]: i=4
     for this_shape in my_shapes:
         this_shape.scaleSize(2)
         this_shape.area()
         i = i - 1
         this_shape.downScale(i)
         all_perimeters.append(this_shape.perimeter())
```

10.1.7.6 Synchronizing results from tasks

```
[9]: all_perimeters = compss_wait_on(all_perimeters)
     print(all_perimeters)
```

```
WARNING: Could not retrieve the object /home/user/.COMPSs/Interactive_07/tmpFiles/
↪pycompssa5pbusyt/1e8234bc-5e80-11ed-96d1-a86daaac2cd1-12 since the task that produces it␣
↪may have been IGNORED or CANCELLED. Please, check the logs. Returning None.
WARNING: Could not retrieve the object /home/user/.COMPSs/Interactive_07/tmpFiles/
↪pycompssa5pbusyt/1e8234bc-5e80-11ed-96d1-a86daaac2cd1-15 since the task that produces it␣
↪may have been IGNORED or CANCELLED. Please, check the logs. Returning None.
[193.33333333333334, 200.0, 440.0, None, None]
```

```
<IPython.core.display.Javascript object>
```

```
INFO: The ERRMGR displayed some error or warnings.
```

10.1.7.7 Stop the runtime

```
[10]: ipycompss.stop(sync=False)
```

```
********************************************************
*************** STOPPING PyCOMPSs ******************
********************************************************
Checking if any issue happened.
Warning: some of the variables used with PyCOMPSs may
        have not been brought to the master.
********************************************************
```

10.1.8 PyCOMPSs: Using files

In this example we will how files can be used with PyCOMPSs.

10.1.8.1 Import the PyCOMPSs library

```
[1]: import pycompss.interactive as ipycompss
```

10.1.8.2 Start the runtime

Initialize COMPSs runtime Parameters indicates if the execution will generate task graph, tracefile, monitor interval and debug information.

```
[2]: import os
     if 'BINDER_SERVICE_HOST' in os.environ:
         ipycompss.start(graph=True, debug=False,
                         project_xml='../xml/project.xml',
                         resources_xml='../xml/resources.xml')
     else:
         ipycompss.start(graph=True, monitor=1000, trace=False, debug=False)
```

```
********************************************************
*************** PyCOMPSs Interactive ******************
********************************************************
*          .-~~-.--.                _____          ___         *
*        :         )              |____ \        /   |         *
*     .~ ~ -.\       /.- ~~ .         __) |        /_/| |         *
*     >       `.   .'       <         |__ |           | |         *
*    (         .- -.         )     ____) |    _      | |         *
*     `- -.-~  `- -'  ~-.- -'     |_____/   |_|     |_|         *
*       (        :         )           _ _ .-:               *
*        ~--.    :    .--~          .-~ .-~ }            *
*            ~-.-^-.-~ \_        .~  .-~   .~          *
*                     \ \ '     \ '_ _ -~            *
*                      \`.\`.    //                 *
*              . - ~ ~-.__\`.\`-.//                 *
*          .-~   . - ~  }~ ~ ~-.~-.              *
*        .' .-~      .-~       :/~-.~-./:             *
*       /_~_ _ . - ~                 ~-.~-._            *
*                                    ~-.<           *
********************************************************
* - Starting COMPSs runtime...                          *
* - Log path : /home/user/.COMPSs/Interactive_08/
```

```
* - PyCOMPSs Runtime started... Have fun!                *
**********************************************************
```

10.1.8.3 Importing task and parameter modules

Import task module before annotating functions or methods

```
[3]: from pycompss.api.task import task
     from pycompss.api.parameter import FILE_IN, FILE_OUT, FILE_INOUT
     from pycompss.api.api import compss_wait_on, compss_open
```

10.1.8.4 Declaring tasks

Declare functions and decorate with @task those that should be tasks

```
[4]: @task(fout=FILE_OUT)
     def write(fout, content):
         with open(fout, 'w') as fout_d:
             fout_d.write(content)
```

```
[5]: @task(finout=FILE_INOUT)
     def append(finout):
         finout_d = open(finout, 'a')
         finout_d.write("\n===> INOUT FILE ADDED CONTENT")
         finout_d.close()
```

```
[6]: @task(fin=FILE_IN, returns=str)
     def readFile(fin):
         fin_d = open(fin, 'r')
         content = fin_d.read()
         fin_d.close()
         return content
```

10.1.8.5 Invoking tasks

```
[7]: f = "myFile.txt"
     content = "OUT FILE CONTENT"
     write(f, content)
```

```
Found task: write
```

```
[8]: append(f)
```

```
Found task: append
```

```
[9]: readed = readFile(f)
```

```
Found task: readFile
```

```
[10]: append(f)
```

Accessing data outside tasks requires synchronization

```
[11]: readed = compss_wait_on(readed)
      print(readed)
```

```
OUT FILE CONTENT
===> INOUT FILE ADDED CONTENT
```

```
[12]: with compss_open(f) as fd:
          f_content = fd.read()
      print(f_content)
```

```
OUT FILE CONTENT
===> INOUT FILE ADDED CONTENT
===> INOUT FILE ADDED CONTENT
```

10.1.8.6 Stop the runtime

```
[13]: ipycompss.stop(sync=True)
```

```
********************************************************
*************** STOPPING PyCOMPSs ******************
********************************************************
Checking if any issue happened.
Synchronizing all future objects left on the user scope.
********************************************************
```

10.1.9 PyCOMPSs: Using constraints

In this example we will how to define task constraints with PyCOMPSs.

10.1.9.1 Import the PyCOMPSs library

```
[1]: import pycompss.interactive as ipycompss
```

10.1.9.2 Starting runtime

Initialize COMPSs runtime Parameters indicates if the execution will generate task graph, tracefile, monitor interval and debug information.

```
[2]: import os
     if 'BINDER_SERVICE_HOST' in os.environ:
         ipycompss.start(graph=True, debug=False,
                         project_xml='../xml/project.xml',
                         resources_xml='../xml/resources.xml')
     else:
         ipycompss.start(graph=True, monitor=1000, trace=True, debug=False)
```

```
********************************************************
*************** PyCOMPSs Interactive ******************
********************************************************
*             .-~~-.--.              _____        ___        *
*            :         )            |____ \      / | |        *
```

(continues on next page)

```
*    .~ ~ -.\         /.- ~~ .       __) |      /_/| |     *
*    >          `.   .'         <      |__ |          | |     *
*    (          .- -.          )    ____) |    _      | |     *
*    `- -.-~  `- -'  ~-.- -'   |_____/  |_|    |_|     *
*      (        :        )            _ _ .-:          *
*       ~--.    :    .--~          .-~  .-~  }        *
*          ~-.-^-.-~ \_        .~   .-~   .~          *
*               \ \ '      \ '_ _ -~          *
*               \`.\`.     //                *
*       . - ~ ~-.__\`.\`-.//                *
*      .-~   . - ~  }~ ~ ~-.~-.              *
*    .' .-~      .-~       :/~-.~-./:          *
*   /_~_ _ . - ~                 ~-.~-._          *
*                                ~-.<          *
**********************************************************
* - Starting COMPSs runtime...                          *
* - Log path : /home/user/.COMPSs/Interactive_09/
* - PyCOMPSs Runtime started... Have fun!               *
**********************************************************
```

10.1.9.3 Importing task and arguments directionality modules

Import task module before annotating functions or methods

```
[3]: from pycompss.api.task import task
     from pycompss.api.parameter import *
     from pycompss.api.api import compss_barrier
     from pycompss.api.constraint import constraint
     from pycompss.api.implement import implement
```

10.1.9.4 Declaring tasks

Declare functions and decorate with @task those that should be tasks

```
[4]: @constraint(computing_units="2")
     @task(returns=int)
     def square(val1):
         return val1 * val1
```

```
[5]: @constraint(computing_units="1")
     @task(returns=int)
     def add(val2, val3):
         return val2 + val3
```

```
[6]: @constraint(computing_units="4")
     @task(returns=int)
     def multiply(val1, val2):
         return val1 * val2
```

10.1.9.5 Invoking tasks

```
[7]: for i in range(20):
         r1 = square(i)
         r2 = add(r1,i)
         r3 = multiply(r2,r1)

     compss_barrier()
```

```
Found task: square
Found task: add
Found task: multiply
```

10.1.9.6 Stop the runtime

```
[8]: ipycompss.stop(sync=True)
```

```
********************************************************
*************** STOPPING PyCOMPSs ******************
********************************************************
Checking if any issue happened.
Synchronizing all future objects left on the user scope.
Found a future object: r1
Found a future object: r2
Found a future object: r3
********************************************************
```

```
[9]: print(r1)
     print(r2)
     print(r3)
```

```
361
380
137180
```

10.1.10 PyCOMPSs: Polymorphism

In this example we will how to use polimorphism with PyCOMPSs.

10.1.10.1 Import the PyCOMPSs library

```
[1]: import pycompss.interactive as ipycompss
```

10.1.10.2 Start the runtime

Initialize COMPSs runtime Parameters indicates if the execution will generate task graph, tracefile, monitor interval and debug information.

```
[2]: import os
     if 'BINDER_SERVICE_HOST' in os.environ:
         ipycompss.start(graph=True, debug=False,
                         project_xml='../xml/project.xml',
                         resources_xml='../xml/resources.xml')
```

```
else:
    ipycompss.start(graph=True, monitor=1000, trace=False, debug=False)
```

```
*********************************************************
*************** PyCOMPSs Interactive *******************
*********************************************************
*          .-~~-.--.              _____        ___      *
*        :         )            |____   \      /   |      *
*  .~ ~ -.\       /.- ~~ .         __) |      /_/| |      *
*  >       `.   .'       <        |__  |        | |       *
*  (         .- -.         )    ____) |    _    | |       *
*   `- -.-~  `- -'  ~-.- -'    |_____/   |_|   |_|       *
*     (        :        )            _ _ .-:              *
*      ~--.    :    .--~         .-~  .-~  }              *
*          ~-.-^-.-~ \_        .~  .-~    .~              *
*                   \ \ '     \ '_ _ -~                   *
*                    \`.\`.    //                         *
*           . - ~ ~-.__\`.\`-.//                          *
*       .-~   . - ~  }~ ~ ~-.~-.                          *
*     .' .-~      .-~       :/~-.~-./:                    *
*    /_~_ _ . - ~                 ~-.~-._                 *
*                                     ~-.<                *
*********************************************************
* - Starting COMPSs runtime...                          *
* - Log path : /home/user/.COMPSs/Interactive_10/        *
* - PyCOMPSs Runtime started... Have fun!                *
*********************************************************
```

10.1.10.3 Create a file to define the tasks

Importing task, implement and constraint modules

```
[3]: %%writefile module.py

from pycompss.api.task import task
from pycompss.api.implement import implement
from pycompss.api.constraint import constraint
```

```
Writing module.py
```

10.1.10.4 Declaring tasks into the file

Declare functions and decorate with @task those that should be tasks

```
[4]: %%writefile -a module.py

@constraint(computing_units='1')
@task(returns=list)
def addtwovectors(list1, list2):
    for i in range(len(list1)):
        list1[i] += list2[i]
    return list1
```

```
Appending to module.py
```

```
[5]: %%writefile -a module.py

@implement(source_class="module", method="addtwovectors")
@constraint(computing_units='4')
@task(returns=list)
def addtwovectorsWithNumpy(list1, list2):
    import numpy as np
    x = np.array(list1)
    y = np.array(list2)
    z = x + y
    return z.tolist()
```

```
Appending to module.py
```

10.1.10.5 Invoking tasks

```
[6]: from pycompss.api.api import compss_wait_on
     from module import addtwovectors  # Just import and use addtwovectors
     from random import random

     vectors = 100
     vector_length = 5000
     vectors_a = [[random() for i in range(vector_length)] for i in range(vectors)]
     vectors_b = [[random() for i in range(vector_length)] for i in range(vectors)]

     results = []
     for i in range(vectors):
         results.append(addtwovectors(vectors_a[i], vectors_b[i]))
```

Accessing data outside tasks requires synchronization

```
[7]: results = compss_wait_on(results)
     print(len(results))
     print(results[0])
```

```
100
[1.6295444543818425, 0.7682641581840065, 0.5688501346398221, 1.7349915887940313, 0.
↪34240668721599665, 1.8129727228214119, 0.44144609567927784, 1.1844437821450766, 1.
↪0676591360333771, 0.9209777590260219, 1.1522776481300805, 1.046955888210573, 0.
↪12786767509627306, 0.6160455573963768, 0.5108724058925447, 0.09282839671001641, 1.
↪2172796926835634, 1.9061469247953493, 0.9759651959687146, 1.0288654795103946, 0.
↪9803601379420879, 1.3912172238119989, 1.2327713474074518, 1.3442337056109936, 1.
↪4255732215902637, 1.3253522443836894, 0.8268610135391908, 1.1043292364488035, 0.
↪6070547295693902, 0.8709726887247944, 1.210055777428964, 0.21242236230457634, 1.
↪511500027411685, 0.35122900518106226, 1.1508781907620516, 0.5745176650754502, 1.
↪0994146946983876, 0.9257678376773317, 1.448200657177663, 1.5967988695819608, 1.
↪3295550985689313, 1.1029941058575488, 0.3431109981027236, 0.6990268473145599, 0.
↪7714242518230537, 1.5097782642046513, 0.9604620968006221, 1.342106362983846, 0.
↪9761505665478536, 0.31123141111131825, 1.0250820473538282, 0.5420588027427826, 0.
↪8547036648744836, 0.9754340464879068, 0.7869146832666167, 1.8254421672359267, 0.
↪9634842555255763, 1.6251687012712992, 0.5217391478037386, 1.1195882321444568, 0.
↪6802969925140138, 1.314514393107192, 1.6768405114334626, 0.48249403936583246, 0.
↪8360394799329087, 0.917702088754649, 0.6304975754084398, 1.6613439228633697, 1.
↪1091993111720373, 0.7165387567236119, 1.3969871332946617, 0.6651065454440385, 1.
↪08351715914467, 0.8253300426051241, 1.3473327507790946, 1.2293175856957155, 0.
↪2783173036656482, 0.4809001409913112, 1.1744839107586837, 1.280072766065557
↪3974369188100224, 1.3516651795839523, 1.564275359847209, 1.034627766474135, 1.
↪430627396661035, 1.2892791883462118, 0.3956435208777903, 0.8579659206295474, 1.
↪4650996364529476, 1.137436696482654, 1.0891576168518322, 1.4097814952793513, 0.
↪9190060648874148, 1.051779157825969, 0.8850654233418775, 1.2933265285160087, 0.
↪8756672396284963, 0.6472448647348019, 0.5128851024940668, 0.5918081816854716, 0.
↪9033431850958179, 0.863086320933653, 1.945584453260528, 0.9471607661909363, 0.
```

10.1.10.6 Stop the runtime

```
[8]: ipycompss.stop(sync=True)
```

```
********************************************************
*************** STOPPING PyCOMPSs ******************
********************************************************
Checking if any issue happened.
Synchronizing all future objects left on the user scope.
Found a list to synchronize: vectors_a
Found a list to synchronize: vectors_b
Found a list to synchronize: results
********************************************************
```

10.1.11 PyCOMPSs: Other decorators - *Binary*

In this example we will how to invoke binaries as tasks with PyCOMPSs.

10.1.11.1 Import the PyCOMPSs library

```
[1]: import pycompss.interactive as ipycompss
```

10.1.11.2 Start the runtime

Initialize COMPSs runtime Parameters indicates if the execution will generate task graph, tracefile, monitor interval and debug information.

```
[2]: import os
     if 'BINDER_SERVICE_HOST' in os.environ:
         ipycompss.start(debug=False,
                         project_xml='../xml/project.xml',
                         resources_xml='../xml/resources.xml')
     else:
         ipycompss.start(graph=True, monitor=1000, trace=True, debug=True)
```

```
********************************************************
*************** PyCOMPSs Interactive ******************
********************************************************
*          .-~~-.--.                _____          ___          *
*         :         )              |____ \        /   |         *
*   .~ ~ -.\        /.- ~~ .          __) |       /_/| |         *
*   >       `.     .'       <       |__  |            | |         *
*   (         .- -.         )     ____) |    _        | |         *
*   `- -.-~  `- -'  ~-.- -'      |_____/   |_|      |_|         *
*     (          :            )          _ _ .-:              *
*      ~--.       :       .--~         .-~  .-~  }            *
*          ~-.-^-.-~ \_        .~    .-~    .~            *
*               \ \ '      \ '_ _ -~            *
*                \`.\`.      //                            *
*        . - ~ ~-.__\`.\`-.//                            *
*          .-~     . - ~  }~ ~ ~-.~-.                      *
********************************************************
```

```
*      .' .-~       .-~        :/~-.~-./:                    *
*     /_~_ _ . - ~                    ~-.~-._               *
*                                        ~-.<              *
********************************************************
* - Starting COMPSs runtime...                           *
* - Log path : /home/user/.COMPSs/Interactive_11/
* - PyCOMPSs Runtime started... Have fun!                *
********************************************************
```

10.1.11.3 Importing task and binary modules

Import task module before annotating functions or methods

```python
[3]: from pycompss.api.task import task
     from pycompss.api.binary import binary
     from pycompss.api.parameter import *
```

10.1.11.4 Declaring tasks

Declare functions and decorate with @task those that should be tasks and with @binary the ones that execute a binary file

```python
[4]: @binary(binary="sed")
     @task(file=FILE_INOUT)
     def sed(flag, expression, file):
         # Equivalent to: $ sed flag expression file
         pass
```

```python
[5]: @binary(binary="grep")
     @task(infile={Type:FILE_IN, StdIOStream:STDIN}, result={Type:FILE_OUT, StdIOStream:STDOUT})
     def grep(keyword, infile, result):
         # Equivalent to: $ grep keyword < infile > result
         pass
```

10.1.11.5 Invoking tasks

```python
[6]: from pycompss.api.api import compss_open

     finout = "inoutfile.txt"
     with open(finout, 'w') as finout_d:
         finout_d.write("Hi, this a simple test!")
         finout_d.write("\nHow are you?")

     sed('-i', 's/Hi/Hello/g', finout)
     fout = "outfile.txt"
     grep("Hello", finout, fout)
```

```
Task definition detected.
Found task: sed
Task definition detected.
Found task: grep
```

Accessing data outside tasks requires synchronization

```
[7]: # Check the result of 'sed'
     with compss_open(finout, "r") as finout_r:
         sedresult = finout_r.read()
     print(sedresult)
```

```
Hello, this a simple test!
How are you?
```

```
[8]: # Check the result of 'grep'
     with compss_open(fout, "r") as fout_r:
         grepresult = fout_r.read()
     print(grepresult)
```

```
Hello, this a simple test!
```

10.1.11.6 Stop the runtime

```
[9]: ipycompss.stop(sync=True)
```

```
*********************************************************
**************** STOPPING PyCOMPSs ******************
*********************************************************
Checking if any issue happened.
Synchronizing all future objects left on the user scope.
*********************************************************
```

10.1.12 PyCOMPSs: Integration with Numba

In this example we will how to use Numba with PyCOMPSs.

10.1.12.1 Import the PyCOMPSs library

```
[1]: import pycompss.interactive as ipycompss
```

10.1.12.2 Starting runtime

Initialize COMPSs runtime Parameters indicates if the execution will generate task graph, tracefile, monitor interval and debug information.

```
[2]: import os
     if 'BINDER_SERVICE_HOST' in os.environ:
         ipycompss.start(graph=True, debug=False,
                         project_xml='../xml/project.xml',
                         resources_xml='../xml/resources.xml')
     else:
         ipycompss.start(graph=True, monitor=1000, trace=True, debug=False)
```

```
*********************************************************
**************** PyCOMPSs Interactive ******************
*********************************************************
```

```
*            .-~~-.--.                  _____           ___       *
*         :          )                |____  \        /   |       *
*   .~ ~ -.\         /.- ~~ .             __) |       /_/| |       *
*   >        `.    .'        <            |__  |          | |       *
*  (         .- -.           )         ___) |       _        | |       *
*   `- -.-~  `- -'  ~-.- -'         |_____/      |_|      |_|       *
*     (        :        )               _ _ .-:            *
*      ~--.    :    .--~            .-~  .-~  }          *
*          ~-.-^-.-~ \_          .~  .-~     .~          *
*                  \ \ '      \ '_ _ -~          *
*                  \`.\`.      //                   *
*         . - ~ ~-.__\`.\`-.//                       *
*       .-~    . - ~  }~ ~ ~-.~-.                      *
*      .' .-~        .-~       :/~-.~-./:              *
*     /_~_ _ . - ~                 ~-.~-._            *
*                                  ~-.<            *
**********************************************************
* - Starting COMPSs runtime...                            *
* - Log path : /home/user/.COMPSs/Interactive_12/        *
* - PyCOMPSs Runtime started... Have fun!                 *
**********************************************************
```

10.1.12.3 Importing task and arguments directionality modules

Import task module before annotating functions or methods

```python
[3]: from pycompss.api.task import task
     from pycompss.api.parameter import *
     from pycompss.api.api import compss_barrier
     from pycompss.api.api import compss_wait_on
```

10.1.12.4 Importing other modules

Import the time and numpy modules

```python
[4]: import time
     import numpy as np
```

10.1.12.5 Declaring tasks

Declare functions and decorate with @task those that should be tasks – Note that they are exactly the same but the "numba" parameter in the @task decorator

```python
[5]: @task(returns=1, numba=False)   # Default: numba=False
     def ident_loops(x):
         r = np.empty_like(x)
         n = len(x)
         for i in range(n):
             r[i] = np.cos(x[i]) ** 2 + np.sin(x[i]) ** 2
         return r
```

```
[6]: @task(returns=1, numba=True)
     def ident_loops_jit(x):
         r = np.empty_like(x)
         n = len(x)
         for i in range(n):
             r[i] = np.cos(x[i]) ** 2 + np.sin(x[i]) ** 2
         return r
```

10.1.12.6 Invoking tasks

```
[7]: size = 1000000
     ntasks = 8

     # Run some tasks without numba jit
     start = time.time()
     for i in range(ntasks):
         out = ident_loops(np.arange(size))
     compss_barrier()
     end = time.time()

     # Run some tasks with numba jit
     start_jit = time.time()
     for i in range(ntasks):
         out_jit = ident_loops_jit(np.arange(size))
     compss_barrier()
     end_jit = time.time()

     # Get the last result of each run to compare that the results are ok
     out = compss_wait_on(out)
     out_jit = compss_wait_on(out_jit)

     print("TIMING RESULTS:")
     print("* ident_loops      : %s seconds" % str(end - start))
     print("* ident_loops_jit  : %s seconds" % str(end_jit - start_jit))
     if len(out) == len(out_jit) and list(out) == list(out_jit):
         print("* SUCCESS: Results match.")
     else:
         print("* FAILURE: Results are different!!!")
```

```
Found task: ident_loops
Found task: ident_loops_jit
TIMING RESULTS:
* ident_loops      : 10.02781057357788 seconds
* ident_loops_jit  : 1.8261973857879639 seconds
* SUCCESS: Results match.
```

10.1.12.7 Stop the runtime

```
[8]: ipycompss.stop(sync=False)
```

```
********************************************************
*************** STOPPING PyCOMPSs ******************
********************************************************
Checking if any issue happened.
Warning: some of the variables used with PyCOMPSs may
        have not been brought to the master.
********************************************************
```

10.1.13 Dislib tutorial

This tutorial will show the basics of using dislib.

10.1.13.1 Setup

First, we need to start an interactive PyCOMPSs session:

```
[1]: import os
     os.environ["ComputingUnits"] = "1"

     import pycompss.interactive as ipycompss
     if 'BINDER_SERVICE_HOST' in os.environ:
         ipycompss.start(graph=True,
                         project_xml='../xml/project.xml',
                         resources_xml='../xml/resources.xml')
     else:
         ipycompss.start(graph=True, monitor=1000)
```

```
********************************************************
*************** PyCOMPSs Interactive ******************
********************************************************
*           .-~~-.--.                  _____        ___        *
*          :         )              |____   \      /   |        *
*    .~ ~ -.\       /.- ~~ .          __) |      /_/| |        *
*    >       `.   .'       <         |__   |           | |        *
*    (         .- -.         )       ____) |    _       | |        *
*    `- -.-~  `- -'  ~-.- -'        |_____/   |_|     |_|        *
*      (         :           )              _ _ .-:             *
*       ~--.    :    .--~          .-~  .-~  }          *
*           ~-.-^-.-~ \_         .~  .-~   .~            *
*                    \ \ '     \ '_ _ -~             *
*                     \`.\`.    //                    *
*            . - ~ ~-.__\`.\`-.//                      *
*        .-~   . - ~  }~ ~ ~-.~-.                       *
*      .' .-~      .-~       :/~-.~-./:                  *
*     /_~_ _ . - ~                 ~-.~-._            *
*                                       ~-.<          *
********************************************************
* - Starting COMPSs runtime...                        *
* - Log path : /home/user/.COMPSs/Interactive_13/     *
* - PyCOMPSs Runtime started... Have fun!             *
********************************************************
```

Next, we import dislib and we are all set to start working!

```
[2]: import dislib as ds
```

10.1.13.2 Distributed arrays

The main data structure in dislib is the distributed array (or ds-array). These arrays are a distributed representation of a 2-dimensional array that can be operated as a regular Python object. Usually, rows in the array represent samples, while columns represent features.

To create a random array we can run the following NumPy-like command:

```
[3]: x = ds.random_array(shape=(500, 500), block_size=(100, 100))
     print(x.shape)
     x
```

```
(500, 500)
```

```
[3]: ds-array(blocks=(...), top_left_shape=(100, 100), reg_shape=(100, 100), shape=(500, 500),␣
     ↪sparse=False)
```

Now `x` is a 500x500 ds-array of random numbers stored in blocks of 100x100 elements. Note that `x` is not stored in memory. Instead, `random_array` generates the contents of the array in tasks that are usually executed remotely. This allows the creation of really big arrays.

The content of `x` is a list of `Futures` that represent the actual data (wherever it is stored).

To see this, we can access the `_blocks` field of `x`:

```
[4]: x._blocks[0][0]
```

```
[4]: <pycompss.runtime.management.classes.Future at 0x7f0428791240>
```

`block_size` is useful to control the granularity of dislib algorithms.

To retrieve the actual contents of `x`, we use `collect`, which synchronizes the data and returns the equivalent NumPy array:

```
[5]: x.collect()
```

```
[5]: array([[0.48604732, 0.68571232, 0.98557605, ..., 0.51530027, 0.39511585,
             0.42942001],
            [0.03398195, 0.40964073, 0.5437061 , ..., 0.16162333, 0.79046618,
             0.71677277],
            [0.82399233, 0.80869154, 0.16965568, ..., 0.79380114, 0.31004525,
             0.51511589],
            ...,
            [0.57630698, 0.72028925, 0.11842501, ..., 0.92236462, 0.5837854 ,
             0.92114111],
            [0.84521256, 0.17909749, 0.42140394, ..., 0.95331429, 0.01587735,
             0.58532187],
            [0.81065273, 0.5666422 , 0.65635218, ..., 0.58820423, 0.42493203,
             0.84351429]])
```

Another way of creating ds-arrays is using array-like structures like NumPy arrays or lists:

```
[6]: x1 = ds.array([[1, 2, 3], [4, 5, 6]], block_size=(1, 3))
     x1
```

```
[6]: ds-array(blocks=(...), top_left_shape=(1, 3), reg_shape=(1, 3), shape=(2, 3), sparse=False)
```

Distributed arrays can also store sparse data in CSR format:

```
[7]: from scipy.sparse import csr_matrix

     sp = csr_matrix([[0, 0, 1], [1, 0, 1]])
     x_sp = ds.array(sp, block_size=(1, 3))
     x_sp
```

```
[7]: ds-array(blocks=(...), top_left_shape=(1, 3), reg_shape=(1, 3), shape=(2, 3), sparse=True)
```

In this case, `collect` returns a CSR matrix as well:

```
[8]: x_sp.collect()
```

```
[8]: <2x3 sparse matrix of type '<class 'numpy.int64'>'
          with 3 stored elements in Compressed Sparse Row format>
```

Loading data

A typical way of creating ds-arrays is to load data from disk. Dislib currently supports reading data in CSV and SVMLight formats like this:

```
[9]: x, y = ds.load_svmlight_file("./files/libsvm/1", block_size=(20, 100), n_features=780, store_
     ↪sparse=True)

     print(x)

     csv = ds.load_txt_file("./files/csv/1", block_size=(500, 122))

     print(csv)
```
```
     ds-array(blocks=(...), top_left_shape=(20, 100), reg_shape=(20, 100), shape=(61, 780),␣
     ↪sparse=True)
     ds-array(blocks=(...), top_left_shape=(500, 122), reg_shape=(500, 122), shape=(4235, 122),␣
     ↪sparse=False)
```

Slicing

Similar to NumPy, ds-arrays support the following types of slicing:

(Note that slicing a ds-array creates a new ds-array)

```
[10]: x = ds.random_array((50, 50), (10, 10))
```

Get a single row:

```
[11]: x[4]
```

```
[11]: ds-array(blocks=(...), top_left_shape=(1, 10), reg_shape=(10, 10), shape=(1, 50),␣
      ↪sparse=False)
```

Get a single element:

```
[12]: x[2, 3]
```

```
[12]: ds-array(blocks=(...), top_left_shape=(1, 1), reg_shape=(1, 1), shape=(1, 1), sparse=False)
```

Get a set of rows or a set of columns:

```
[13]:  # Consecutive rows
       print(x[10:20])

       # Consecutive columns
       print(x[:, 10:20])

       # Non consecutive rows
       print(x[[3, 7, 22]])

       # Non consecutive columns
       print(x[:, [5, 9, 48]])
```

```
ds-array(blocks=(...), top_left_shape=(10, 10), reg_shape=(10, 10), shape=(10, 50),␣
↪sparse=False)
ds-array(blocks=(...), top_left_shape=(10, 10), reg_shape=(10, 10), shape=(50, 10),␣
↪sparse=False)
ds-array(blocks=(...), top_left_shape=(3, 10), reg_shape=(10, 10), shape=(3, 50),␣
↪sparse=False)
ds-array(blocks=(...), top_left_shape=(10, 3), reg_shape=(10, 10), shape=(50, 3),␣
↪sparse=False)
```

Get any set of elements:

```
[14]:  x[0:5, 40:45]
```

```
[14]:  ds-array(blocks=(...), top_left_shape=(5, 5), reg_shape=(10, 10), shape=(5, 5), sparse=False)
```

Other functions

Apart from this, ds-arrays also provide other useful operations like `transpose` and `mean`:

```
[15]:  x.mean(axis=0).collect()
```

```
[15]:  array([0.51352356, 0.49396794, 0.4661033 , 0.48026991, 0.50136143,
               0.49323405, 0.51248831, 0.51658519, 0.4904544 , 0.47166468,
               0.50245676, 0.49936659, 0.47499634, 0.52566765, 0.53676456,
               0.59127036, 0.50947458, 0.47320677, 0.42695456, 0.54335201,
               0.51780756, 0.49855486, 0.53845333, 0.37299501, 0.51229418,
               0.43110043, 0.47262688, 0.41698864, 0.54994596, 0.46676007,
               0.46070067, 0.48861301, 0.45868291, 0.53380687, 0.50555055,
               0.53453463, 0.43711111, 0.52115681, 0.48152436, 0.49215593,
               0.41552034, 0.47669533, 0.5610678 , 0.43511911, 0.49611885,
               0.44116871, 0.42241364, 0.48626255, 0.51636529, 0.44251849])
```

```
[16]:  x.transpose().collect()
```

```
[16]:  array([[0.02733543, 0.65891797, 0.36654465, ..., 0.52109164, 0.86395718,
                0.93593907],
               [0.41462264, 0.97419918, 0.14124931, ..., 0.15893453, 0.49486474,
                0.14138483],
               [0.91312707, 0.53860404, 0.96686988, ..., 0.78763956, 0.18268972,
                0.20551984],
               ...,
               [0.19468602, 0.62184611, 0.81007025, ..., 0.88719987, 0.55132466,
                0.32694948],
               [0.19221646, 0.64678511, 0.98416872, ..., 0.18736269, 0.51392039,
                0.59614856],
```

(continues on next page)

```
        [0.49591758, 0.17913008, 0.11419029, ..., 0.02701779, 0.22316829,
         0.78426262]])
```

10.1.13.3 Machine learning with dislib

Dislib provides an estimator-based API very similar to scikit-learn. To run an algorithm, we first create an estimator. For example, a K-means estimator:

```
[17]: from dislib.cluster import KMeans

      km = KMeans(n_clusters=3)
```

Now, we create a ds-array with some blob data, and fit the estimator:

```
[18]: from sklearn.datasets import make_blobs

      # create ds-array
      x, y = make_blobs(n_samples=1500)
      x_ds = ds.array(x, block_size=(500, 2))

      km.fit(x_ds)
```

```
[18]: KMeans(n_clusters=3, random_state=RandomState(MT19937) at 0x7F040C5D6440)
```

Finally, we can make predictions on new (or the same) data:

```
[19]: y_pred = km.predict(x_ds)
      y_pred
```

```
[19]: ds-array(blocks=(...), top_left_shape=(500, 1), reg_shape=(500, 1), shape=(1500, 1),␣
      ↪sparse=False)
```

y_pred is a ds-array of predicted labels for x_ds

Let's plot the results

```
[20]: %matplotlib inline
      import matplotlib.pyplot as plt


      centers = km.centers

      # set the color of each sample to the predicted label
      plt.scatter(x[:, 0], x[:, 1], c=y_pred.collect())

      # plot the computed centers in red
      plt.scatter(centers[:, 0], centers[:, 1], c='red')
```

```
[20]: <matplotlib.collections.PathCollection at 0x7f03cb313e80>
```
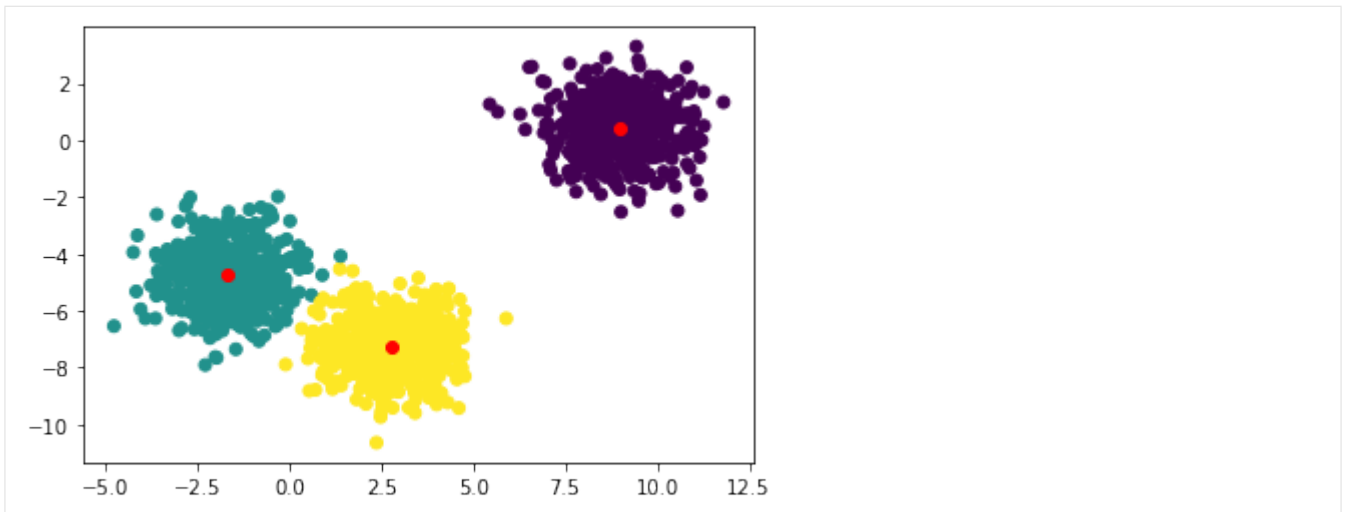
Note that we need to call `y_pred.collect()` to retrieve the actual labels and plot them. The rest is the same as if we were using scikit-learn.

Now let's try a more complex example that uses some preprocessing tools.

First, we load a classification data set from scikit-learn into ds-arrays.

Note that this step is only necessary for demonstration purposes. Ideally, your data should be already loaded in ds-arrays.

```
[21]: from sklearn.datasets import load_breast_cancer
      from sklearn.model_selection import train_test_split

      x, y = load_breast_cancer(return_X_y=True)

      x_train, x_test, y_train, y_test = train_test_split(x, y)

      x_train = ds.array(x_train, block_size=(100, 10))
      y_train = ds.array(y_train.reshape(-1, 1), block_size=(100, 1))

      x_test = ds.array(x_test, block_size=(100, 10))
      y_test = ds.array(y_test.reshape(-1, 1), block_size=(100, 1))
```

Next, we can see how support vector machines perform in classifying the data. We first fit the model (ignore any warnings in this step):

```
[22]: from dislib.classification import CascadeSVM

      csvm = CascadeSVM()

      csvm.fit(x_train, y_train)
```

```
/home/user/github/dislib/dislib/classification/csvm/base.py:395: RuntimeWarning: overflow␣
→encountered in exp
  k = np.exp(k)
/home/user/github/dislib/dislib/classification/csvm/base.py:363: RuntimeWarning: invalid␣
→value encountered in double_scalars
  delta = np.abs((w - self._last_w) / self._last_w)
```

```
[22]: CascadeSVM()
```

and now we can make predictions on new data using `csvm.predict()`, or we can get the model accuracy on the test set with:

```
[23]:  score = csvm.score(x_test, y_test)
```

`score` represents the classifier accuracy, however, it is returned as a `Future`. We need to synchronize to get the actual value:

```
[24]:  from pycompss.api.api import compss_wait_on

       print(compss_wait_on(score))
```
```
       0.6503496503496503
```

The accuracy should be around 0.6, which is not very good. We can scale the data before classification to improve accuracy. This can be achieved using dislib's `StandardScaler`.

The `StandardScaler` provides the same API as other estimators. In this case, however, instead of making predictions on new data, we transform it:

```
[25]:  from dislib.preprocessing import StandardScaler

       sc = StandardScaler()

       # fit the scaler with train data and transform it
       scaled_train = sc.fit_transform(x_train)

       # transform test data
       scaled_test = sc.transform(x_test)
```

Now `scaled_train` and `scaled_test` are the scaled samples. Let's see how SVM perfroms now.

```
[26]:  csvm.fit(scaled_train, y_train)
       score = csvm.score(scaled_test, y_test)
       print(compss_wait_on(score))
```
```
       0.993006993006993
```

The new accuracy should be around 0.9, which is a great improvement!

Close the session

To finish the session, we need to stop PyCOMPSs:

```
[27]:  ipycompss.stop()
```
```
       *******************************************************
       *************** STOPPING PyCOMPSs ******************
       *******************************************************
       Checking if any issue happened.
       Warning: some of the variables used with PyCOMPSs may
                have not been brought to the master.
       *******************************************************
```

10.1.14 Machine Learning with dislib

This tutorial will show the different algorithms available in dislib.

10.1.14.1 Setup

First, we need to start an interactive PyCOMPSs session:

```
[1]: import os
     os.environ["ComputingUnits"] = "1"

     import pycompss.interactive as ipycompss
     if 'BINDER_SERVICE_HOST' in os.environ:
         ipycompss.start(graph=True,
                         project_xml='../xml/project.xml',
                         resources_xml='../xml/resources.xml')
     else:
         ipycompss.start(graph=True, monitor=1000)
```

```
********************************************************
**************** PyCOMPSs Interactive ******************
********************************************************
*          .-~~-.--.                _____        ___       *
*         :         )              |____   \      /   |      *
*   .~ ~ -.\       /.- ~~ .         __) |      /_/| |       *
*   >       `.   .'       <        |__  |          | |       *
*   (         .- -.         )    ____) |      _      | |       *
*   `- -.-~  `- -'  ~-.- -'    |_____/    |_|     |_|       *
*     (           :           )          _ _ .-:            *
*      ~--.    :    .--~        .-~  .-~  }            *
*          ~-.-^-.-~ \_        .~   .-~   .~             *
*                   \ \ '      \ '_   _ -~              *
*                    \`.\`.     //                      *
*           . - ~ ~-.__\`.\`-.//                        *
*        .-~      . - ~  }~ ~ ~-.~-.                     *
*       .' .-~       .-~        :/~-.~-./:               *
*      /_~_ _ . - ~                ~-.~-._              *
*                                      ~-.<            *
********************************************************
* - Starting COMPSs runtime...                         *
* - Log path : /home/user/.COMPSs/Interactive_15/
* - PyCOMPSs Runtime started... Have fun!              *
********************************************************
```

Next, we import dislib and we are all set to start working!

```
[2]: import dislib as ds
```

10.1.14.2 Load the MNIST dataset

```
[3]: x, y = ds.load_svmlight_file('/tmp/mnist/mnist', # Download the dataset
                                   block_size=(10000, 784), n_features=784, store_sparse=False)
```

```
[4]: x.shape
```

```
[4]: (60000, 784)
```

```
[5]: y.shape
```

```
[5]: (60000, 1)
```

```
[6]: y_array = y.collect()
     y_array
```

```
[6]: array([5., 0., 4., ..., 5., 6., 8.])
```

```
[7]: img = x[0].collect().reshape(28,28)
```

```
[8]: %matplotlib inline
     import matplotlib.pyplot as plt
     plt.imshow(img)
```

```
[8]: <matplotlib.image.AxesImage at 0x7ff5c1fbfee0>
```
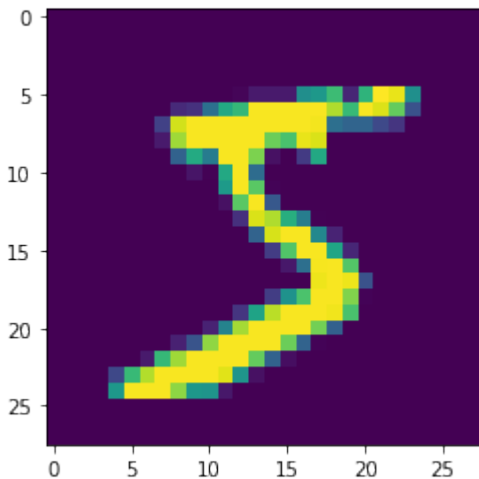


```
[9]: int(y[0].collect())
```

```
[9]: 5
```

10.1.14.3 dislib algorithms

Preprocessing

```
[10]: from dislib.preprocessing import StandardScaler
      from dislib.decomposition import PCA
```

Clustering

```
[11]: from dislib.cluster import KMeans
      from dislib.cluster import DBSCAN
      from dislib.cluster import GaussianMixture
```

Classification

```
[12]: from dislib.classification import CascadeSVM
      from dislib.classification import RandomForestClassifier
```

Recommendation

```
[13]: from dislib.recommendation import ALS
```

Model selection

```
[14]: from dislib.model_selection import GridSearchCV
```

Others

```
[15]: from dislib.regression import LinearRegression
      from dislib.neighbors import NearestNeighbors
```

10.1.14.4 Examples

KMeans

```
[16]: kmeans = KMeans(n_clusters=10)
      pred_clusters = kmeans.fit_predict(x).collect()
```

Get the number of images of each class in the cluster 0:

```
[17]: from collections import Counter
      Counter(y_array[pred_clusters==0])
```

```
[17]: Counter({5.0: 1209,
               3.0: 1058,
               8.0: 3499,
               2.0: 323,
               1.0: 9,
               9.0: 54,
               0.0: 121,
               4.0: 16,
               6.0: 45,
               7.0: 21})
```

GaussianMixture

Fit the GaussianMixture with the painted pixels of a single image:

```
[18]: import numpy as np
      img_filtered_pixels = np.stack([np.array([i, j]) for i in range(28) for j in range(28) if␣
      ↪img[i,j] > 10])
      img_pixels = ds.array(img_filtered_pixels, block_size=(50,2))
      gm = GaussianMixture(n_components=7, random_state=0)
      gm.fit(img_pixels)
```

Get the parameters that define the Gaussian components:

```
[19]: from pycompss.api.api import compss_wait_on
      means = compss_wait_on(gm.means_)
      covariances = compss_wait_on(gm.covariances_)
      weights = compss_wait_on(gm.weights_)
```

Use the Gaussian mixture model to sample random pixels replicating the original distribution:

```
[20]: samples = np.concatenate([np.random.multivariate_normal(means[i], covariances[i],␣
      ↪int(weights[i]*1000))
                              for i in range(7)])
      plt.scatter(samples[:,1], samples[:,0])
      plt.gca().set_aspect('equal', adjustable='box')
      plt.gca().invert_yaxis()
      plt.draw()
```
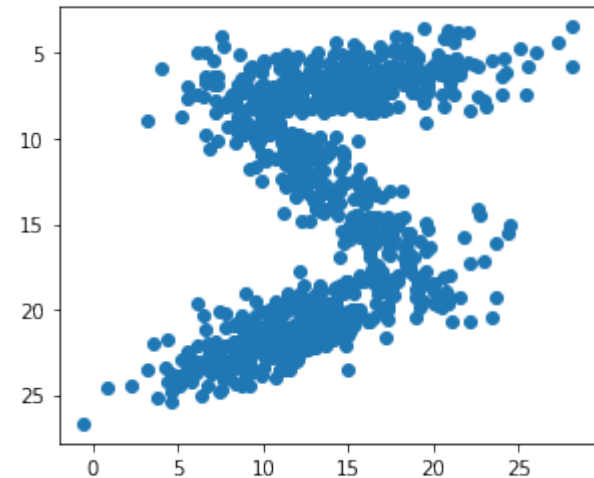

PCA

```
[21]: pca = PCA()
      pca.fit(x)
```

```
[21]: PCA()
```

Calculate the explained variance of the 10 first eigenvectors:

```
[22]: explained_variance = pca.explained_variance_.collect()
      sum(explained_variance[0:10])/sum(explained_variance)
```

```
[22]: 0.48814980354934023
```

Show the weights of the first eigenvector:

```
[23]: plt.imshow(np.abs(pca.components_.collect()[0]).reshape(28,28))
```

```
[23]: <matplotlib.image.AxesImage at 0x7ff5b53b87f0>
```
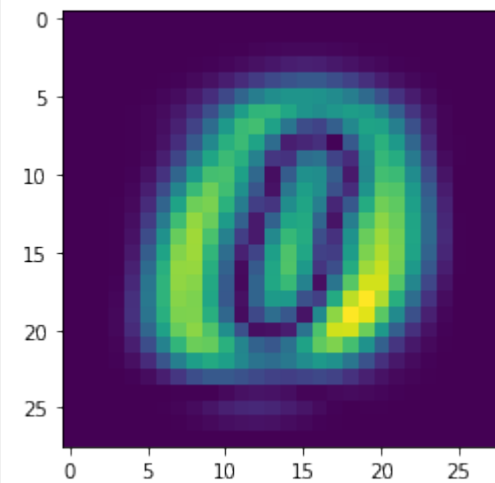

RandomForestClassifier

```
[24]: rf = RandomForestClassifier(n_estimators=5, max_depth=3)
      rf.fit(x, y)
```

```
[24]: RandomForestClassifier(max_depth=3, n_estimators=5)
```

Use the test dataset to get an accuracy score:

```
[25]: x_test, y_test = ds.load_svmlight_file('/tmp/mnist/mnist.test', block_size=(10000, 784), n_
      →features=784, store_sparse=False)
      score = rf.score(x_test, y_test)
      print(compss_wait_on(score))
```

```
0.6152
```

Close the session

To finish the session, we need to stop PyCOMPSs:

```
[26]: ipycompss.stop()
```

```
*****************************************************
*************** STOPPING PyCOMPSs *******************
*****************************************************
Checking if any issue happened.
Warning: some of the variables used with PyCOMPSs may
         have not been brought to the master.
*****************************************************
```

10.2 Hands-on

Here you will find the hands on notebooks used in the tutorials.

10.2.1 Sort by Key

Algorithm that sorts the elements of a set of files and merges the partial results respecting the order.

10.2.1.1 First of all - Create a dataset

This step can be avoided if the dataset already exists.

If not, this code snipped creates a set of files with dictionary on each one generated randomly. Uses pickle.

```python
[1]: def datasetGenerator(directory, numFiles, numPairs):
         import random
         import pickle
         import os
         if os.path.exists(directory):
             print("Dataset directory already exists... Removing")
             import shutil
             shutil.rmtree(directory)
         os.makedirs(directory)
         for f in range(numFiles):
             fragment = {}
             while len(fragment) < numPairs:
                 fragment[random.random()] = random.randint(0, 1000)
             filename = 'file_' + str(f) + '.data'
             with open(directory + '/' + filename, 'wb') as fd:
                 pickle.dump(fragment, fd)
             print('File ' + filename + ' has been created.')
```

```python
[2]: numFiles = 2
     numPairs = 10
     directoryName = 'mydataset'
     datasetGenerator(directoryName, numFiles, numPairs)
```

```
Dataset directory already exists... Removing
File file_0.data has been created.
File file_1.data has been created.
```

```python
[3]: # Show the files that have been created
     %ls -l $directoryName
```

```
total 8
-rw-r--r-- 1 user users 135 nov  7 11:09 file_0.data
-rw-r--r-- 1 user users 133 nov  7 11:09 file_1.data
```

10.2.1.2 Algorithm definition

```
[4]: import pycompss.interactive as ipycompss
```

```
[5]: import os
     if 'BINDER_SERVICE_HOST' in os.environ:
         ipycompss.start(graph=True,
                         project_xml='../xml/project.xml',
                         resources_xml='../xml/resources.xml')
     else:
         ipycompss.start(graph=True, monitor=1000)
```

```
********************************************************
**************** PyCOMPSs Interactive ******************
********************************************************
*          .-~~-.--.              _____         ___       *
*         :         )            |____ \       /   |       *
*   .~ ~ -.\       /.- ~~ .        __) |     /_/| |       *
*   >       `.   .'       <       |__  |        | |       *
*   (         .- -.         )    ____) |    _     | |       *
*   `- -.-~  `- -'  ~-.- -'    |_____/   |_|     |_|       *
*       (         :        )          _ _ .-:       *
*        ~--.    :    .--~         .-~  .-~  }       *
*           ~-.-^-.-~ \_       .~   .-~   .~       *
*                    \ \ '     \ '_ _ -~       *
*                     \`.\`.    //       *
*            . - ~ ~-.__\`.\`-.//       *
*          .-~   . - ~  }~ ~ ~-.~-.       *
*          .' .-~      .-~       :/~-.~-./:       *
*         /_~_ _ . - ~                 ~-.~-._       *
*                                       ~-.<       *
********************************************************
* - Starting COMPSs runtime...                         *
* - Log path : /home/user/.COMPSs/Interactive_17/
* - PyCOMPSs Runtime started... Have fun!              *
********************************************************
```

```
[6]: from pycompss.api.task import task
     from pycompss.api.parameter import FILE_IN
```

```
[7]: @task(returns=list, dataFile=FILE_IN)
     def sortPartition(dataFile):
         '''
         Reads the dataFile and sorts its content which is assumed to be a dictionary {K: V}
         :param path: file that contains the data
         :return: a list of (K, V) pairs sorted.
         '''
         import pickle
         import operator
         with open(dataFile, 'rb') as f:
             data = pickle.load(f)
         # res = sorted(data, key=lambda (k, v): k, reverse=not ascending)
         partition_result = sorted(data.items(), key=operator.itemgetter(0), reverse=False)
         return partition_result
```

```
[8]: @task(returns=list, priority=True)
```

```python
def reducetask(a, b):
    '''
    Merges two partial results (lists of (K, V) pairs) respecting the order
    :param a: Partial result a
    :param b: Partial result b
    :return: The merging result sorted
    '''
    partial_result = []
    i = 0
    j = 0
    while i < len(a) and j < len(b):
        if a[i] < b[j]:
            partial_result.append(a[i])
            i += 1
        else:
            partial_result.append(b[j])
            j += 1
    if i < len(a):
        partial_result + a[i:]
    elif j < len(b):
        partial_result + b[j:]
    return partial_result
```

```python
[9]: def merge_reduce(function, data):
    import sys
    if sys.version_info[0] >= 3:
        import queue as Queue
    else:
        import Queue
    q = Queue.Queue()
    for i in data:
        q.put(i)
    while not q.empty():
        x = q.get()
        if not q.empty():
            y = q.get()
            q.put(function(x, y))
        else:
            return x
```

MAIN

Parameters (that can be configured in the following cell): * datasetPath: The path where the dataset is (default: the same as created previously).

```python
[10]: import os
import time
from pycompss.api.api import compss_wait_on

datasetPath = directoryName   # Where the dataset is
files = []
for f in os.listdir(datasetPath):
    files.append(datasetPath + '/' + f)
```

```python
startTime = time.time()

partialSorted = []
for f in files:
    partialSorted.append(sortPartition(f))
result = merge_reduce(reducetask, partialSorted)

result = compss_wait_on(result)

print("Elapsed Time(s)")
print(time.time() - startTime)
import pprint
pprint.pprint(result)
```

```
Found task: sortPartition
Found task: reducetask
Elapsed Time(s)
3.8698995113372803
[(0.041892723435254076, 357),
 (0.06359546469536281, 14),
 (0.08331557020368874, 878),
 (0.10278403432902072, 880),
 (0.12426682077744233, 630),
 (0.1517902880938985, 337),
 (0.16148801796133683, 356),
 (0.3738492069799837, 194),
 (0.3936177040756046, 529),
 (0.40150631691932415, 258),
 (0.4314880044253141, 176),
 (0.4397589460812781, 894),
 (0.5282304183357669, 391),
 (0.5342789370009435, 971),
 (0.8090653537847396, 621),
 (0.8430013017668287, 166),
 (0.8726915372538349, 908),
 (0.9031628077963247, 744)]
```

```
[11]: ipycompss.stop()
```

```
*****************************************************
*************** STOPPING PyCOMPSs ******************
*****************************************************
Checking if any issue happened.
Warning: some of the variables used with PyCOMPSs may
        have not been brought to the master.
*****************************************************
```

10.2.2 KMeans

KMeans is machine-learning algorithm (NP-hard), popularly employed for cluster analysis in data mining, and interesting for benchmarking and performance evaluation.

The objective of the Kmeans algorithm to group a set of multidimensional points into a predefined number of clusters, in which each point belongs to the closest cluster (with the nearest mean distance), in an iterative process.

```
[1]: import pycompss.interactive as ipycompss
```

```
[2]: import os
     if 'BINDER_SERVICE_HOST' in os.environ:
         ipycompss.start(graph=True,                    # trace=True
                         project_xml='../xml/project.xml',
                         resources_xml='../xml/resources.xml')
     else:
         ipycompss.start(graph=True, monitor=1000)   # trace=True
```

```
********************************************************
**************** PyCOMPSs Interactive ******************
********************************************************
*          .-~~-.--.                _____        ___       *
*         :         )              |____   \      /   |      *
*   .~ ~ -.\       /.- ~~ .           __) |     /_/| |      *
*   >       `.   .'       <          |__  |          | |      *
*   (         .- -.         )       ____) |    _     | |      *
*    `- -.-~  `- -'  ~-.- -'        |_____/   |_|    |_|      *
*      (        :        )            _ _ .-:            *
*       ~--.    :    .--~          .-~  .-~  }           *
*           ~-.-^-.-~ \_       .~   .-~    .~           *
*                    \ \ '      \ '_ _ -~             *
*                     \`.\`.    //                     *
*           . - ~ ~-.__\`.\`-.//                        *
*       .-~   . - ~  }~ ~ ~-.~-.                        *
*     .' .-~      .-~       :/~-.~-./:                   *
*    /_~_ _ . - ~                 ~-.~-._                *
*                                     ~-.<              *
********************************************************
* - Starting COMPSs runtime...                          *
* - Log path : /home/user/.COMPSs/Interactive_18/
* - PyCOMPSs Runtime started... Have fun!               *
********************************************************
```

```
[3]: from pycompss.api.task import task
```

```
[4]: import numpy as np
```

```
[5]: def init_random(numV, dim, seed):
         np.random.seed(seed)
         c = [np.random.uniform(-3.5, 3.5, dim)]
         while len(c) < numV:
             p = np.random.uniform(-3.5, 3.5, dim)
             distance = [np.linalg.norm(p-i) for i in c]
             if min(distance) > 2:
                 c.append(p)
         return c
```

```
[6]:  #@task(returns=list)   # Not a task for plotting
      def genFragment(numV, K, c, dim, mode='gauss'):
          if mode == "gauss":
              n = int(float(numV) / K)
              r = numV % K
              data = []
              for k in range(K):
                  s = np.random.uniform(0.05, 0.75)
                  for i in range(n+r):
                      d = np.array([np.random.normal(c[k][j], s) for j in range(dim)])
                      data.append(d)
              return np.array(data)[:numV]
          else:
              return [np.random.random(dim) for _ in range(numV)]
```

```
[7]:  @task(returns=dict)
      def cluster_points_partial(XP, mu, ind):
          dic = {}
          for x in enumerate(XP):
              bestmukey = min([(i[0], np.linalg.norm(x[1] - mu[i[0]])) for i in enumerate(mu)],␣
      →key=lambda t: t[1])[0]
              if bestmukey not in dic:
                  dic[bestmukey] = [x[0] + ind]
              else:
                  dic[bestmukey].append(x[0] + ind)
          return dic
```

```
[8]:  @task(returns=dict)
      def partial_sum(XP, clusters, ind):
          p = [(i, [(XP[j - ind]) for j in clusters[i]]) for i in clusters]
          dic = {}
          for i, l in p:
              dic[i] = (len(l), np.sum(l, axis=0))
          return dic
```

```
[9]:  @task(returns=dict, priority=True)
      def reduceCentersTask(a, b):
          for key in b:
              if key not in a:
                  a[key] = b[key]
              else:
                  a[key] = (a[key][0] + b[key][0], a[key][1] + b[key][1])
          return a
```

```
[10]:  def mergeReduce(function, data):
           from collections import deque
           q = deque(list(range(len(data))))
           while len(q):
               x = q.popleft()
               if len(q):
                   y = q.popleft()
                   data[x] = function(data[x], data[y])
                   q.append(x)
               else:
                   return data[x]
```

```
[11]:   def has_converged(mu, oldmu, epsilon, iter, maxIterations):
            print("iter: " + str(iter))
            print("maxIterations: " + str(maxIterations))
            if oldmu != []:
                if iter < maxIterations:
                    aux = [np.linalg.norm(oldmu[i] - mu[i]) for i in range(len(mu))]
                    distancia = sum(aux)
                    if distancia < epsilon * epsilon:
                        print("Distance_T: " + str(distancia))
                        return True
                    else:
                        print("Distance_F: " + str(distancia))
                        return False
                else:
                    # Reached the max amount of iterations
                    return True
```

```
[12]:   def plotKMEANS(dim, mu, clusters, data):
            import pylab as plt
            colors = ['b','g','r','c','m','y','k']
            if dim == 2 and len(mu) <= len(colors):
                from matplotlib.patches import Circle
                from matplotlib.collections import PatchCollection
                fig, ax = plt.subplots(figsize=(10,10))
                patches = []
                pcolors = []
                for i in range(len(clusters)):
                    for key in clusters[i].keys():
                        d = clusters[i][key]
                        for j in d:
                            j = j - i * len(data[0])
                            C = Circle((data[i][j][0], data[i][j][1]), .05)
                            pcolors.append(colors[key])
                            patches.append(C)
                collection = PatchCollection(patches)
                collection.set_facecolor(pcolors)
                ax.add_collection(collection)
                x, y = zip(*mu)
                plt.plot(x, y, '*', c='y', markersize=20)
                plt.autoscale(enable=True, axis='both', tight=False)
                plt.show()
            elif dim == 3 and len(mu) <= len(colors):
                from mpl_toolkits.mplot3d import Axes3D
                fig = plt.figure()
                ax = fig.add_subplot(111, projection='3d')
                for i in range(len(clusters)):
                    for key in clusters[i].keys():
                        d = clusters[i][key]
                        for j in d:
                            j = j - i * len(data[0])
                            ax.scatter(data[i][j][0], data[i][j][1], data[i][j][2], 'o',␣
        →c=colors[key])
                x, y, z = zip(*mu)
                for i in range(len(mu)):
                    ax.scatter(x[i], y[i], z[i], s=80, c='y', marker='D')
                plt.show()
```

(continues on next page)

```
    else:
        print("No representable dim or not enough colours")
```

10.2.2.1 MAIN

Parameters (that can be configured in the following cell): * numV: number of vectors (default: 10.000) * dim: dimension of the points (default: 2) * k: number of centers (default: 4) * numFrag: number of fragments (default: 16) * epsilon: convergence condition (default: 1e-10) * maxIterations: Maximum number of iterations (default: 20)

```
[13]: %matplotlib inline
      import ipywidgets as widgets
      from pycompss.api.api import compss_wait_on

      w_numV = widgets.IntText(value=10000)         # Number of Vectors - with 1000 it is feasible␣
      ↪to see the evolution across iterations
      w_dim = widgets.IntText(value=2)              # Number of Dimensions
      w_k = widgets.IntText(value=4)                # Centers
      w_numFrag = widgets.IntText(value=16)         # Fragments
      w_epsilon = widgets.FloatText(value=1e-10)    # Convergence condition
      w_maxIterations = widgets.IntText(value=20)   # Max number of iterations
      w_seed = widgets.IntText(value=8)             # Random seed

      def kmeans(numV, dim, k, numFrag, epsilon, maxIterations, seed):
          size = int(numV / numFrag)
          cloudCenters = init_random(k, dim, seed) # centers to create data groups
          X = [genFragment(size, k, cloudCenters, dim, mode='gauss') for _ in range(numFrag)]
          mu = init_random(k, dim, seed - 1)        # First centers
          oldmu = []
          n = 0
          while not has_converged(mu, oldmu, epsilon, n, maxIterations):
              oldmu = mu
              clusters = [cluster_points_partial(X[f], mu, f * size) for f in range(numFrag)]
              partialResult = [partial_sum(X[f], clusters[f], f * size) for f in range(numFrag)]
              mu = mergeReduce(reduceCentersTask, partialResult)
              mu = compss_wait_on(mu)
              mu = [mu[c][1] / mu[c][0] for c in mu]
              while len(mu) < k:
                  # Add new random center if one of the centers has no points.
                  indP = np.random.randint(0, size)
                  indF = np.random.randint(0, numFrag)
                  mu.append(X[indF][indP])
              n += 1
          clusters = compss_wait_on(clusters)
          plotKMEANS(dim, mu, clusters, X)
          print("--------------------")
          print("Result:")
          print("Iterations: ", n)
          print("Centers: ", mu)
          print("--------------------")

      widgets.interact_manual(kmeans, numV=w_numV, dim=w_dim, k=w_k, numFrag=w_numFrag, epsilon=w_
      ↪epsilon, maxIterations=w_maxIterations, seed=w_seed)
```

```
interactive(children=(IntText(value=10000, description='numV'), IntText(value=2, description=
↪'dim'), IntText(v...
```

```
[13]: <function __main__.kmeans(numV, dim, k, numFrag, epsilon, maxIterations, seed)>
```

```
[14]: ipycompss.stop()
```

```
********************************************************
*************** STOPPING PyCOMPSs ******************
********************************************************
Checking if any issue happened.
Warning: some of the variables used with PyCOMPSs may
        have not been brought to the master.
********************************************************
```

10.2.3 KMeans with Reduce

KMeans is machine-learning algorithm (NP-hard), popularly employed for cluster analysis in data mining, and interesting for benchmarking and performance evaluation.

The objective of the Kmeans algorithm to group a set of multidimensional points into a predefined number of clusters, in which each point belongs to the closest cluster (with the nearest mean distance), in an iterative process.

```
[1]: import pycompss.interactive as ipycompss
```

```
[2]: import os
     if 'BINDER_SERVICE_HOST' in os.environ:
         ipycompss.start(graph=True,                    # trace=True
                         project_xml='../xml/project.xml',
                         resources_xml='../xml/resources.xml')
     else:
         ipycompss.start(graph=True, monitor=1000)  # trace=True
```

```
********************************************************
*************** PyCOMPSs Interactive ******************
********************************************************
*          .-~~-.--.                _____         ___        *
*         :         )              |____   \       /   |       *
*   .~ ~ -.\       /.- ~~ .          __) |     /_/| |       *
*   >       `.   .'       <         |__  |         | |        *
*  (         .- -.         )     ___) |    _      | |        *
*   `- -.-~  `- -'  ~-.- -'     |_____/   |_|     |_|        *
*     (        :         )          _ _ .-:               *
*      ~--.    :    .--~        .-~  .-~  }               *
*          ~-.-^-.-~ \_        .~   .-~    .~             *
*                   \ \ '     \ '_ _ -~             *
*                    \`.\`.    //                     *
*           . - ~ ~-.__\`.\`-.//                     *
*       .-~   . - ~  }~ ~ ~-.~-.                      *
*     .' .-~      .-~       :/~-.~-./:                   *
*    /_~_ _ . - ~                 ~-.~-._                 *
*                                    ~-.<           *
********************************************************
* - Starting COMPSs runtime...                        *
* - Log path : /home/user/.COMPSs/Interactive_19/
* - PyCOMPSs Runtime started... Have fun!             *
********************************************************
```

```
[3]: from pycompss.api.task import task
```

```
[4]: import numpy as np
```

```
[5]: def init_random(numV, dim, seed):
         np.random.seed(seed)
         c = [np.random.uniform(-3.5, 3.5, dim)]
         while len(c) < numV:
             p = np.random.uniform(-3.5, 3.5, dim)
             distance = [np.linalg.norm(p-i) for i in c]
             if min(distance) > 2:
                 c.append(p)
         return c
```

```
[6]: #@task(returns=list)  # Not a task for plotting
     def genFragment(numV, K, c, dim, mode='gauss'):
         if mode == "gauss":
             n = int(float(numV) / K)
             r = numV % K
             data = []
             for k in range(K):
                 s = np.random.uniform(0.05, 0.75)
                 for i in range(n+r):
                     d = np.array([np.random.normal(c[k][j], s) for j in range(dim)])
                     data.append(d)
             return np.array(data)[:numV]
         else:
             return [np.random.random(dim) for _ in range(numV)]
```

```
[7]: @task(returns=dict)
     def cluster_points_partial(XP, mu, ind):
         dic = {}
         for x in enumerate(XP):
             bestmukey = min([(i[0], np.linalg.norm(x[1] - mu[i[0]])) for i in enumerate(mu)],␣
     ↪key=lambda t: t[1])[0]
             if bestmukey not in dic:
                 dic[bestmukey] = [x[0] + ind]
             else:
                 dic[bestmukey].append(x[0] + ind)
         return dic
```

```
[8]: @task(returns=dict)
     def partial_sum(XP, clusters, ind):
         p = [(i, [(XP[j - ind]) for j in clusters[i]]) for i in clusters]
         dic = {}
         for i, l in p:
             dic[i] = (len(l), np.sum(l, axis=0))
         return dic
```

```
[9]: def reduceCenters(a, b):
         """
         Reduce method to sum the result of two partial_sum methods
         :param a: partial_sum {cluster_ind: (#points_a, sum(points_a))}
         :param b: partial_sum {cluster_ind: (#points_b, sum(points_b))}
         :return: {cluster_ind: (#points_a+#points_b, sum(points_a+points_b))}
```

(continues on next page)

```
    """
    for key in b:
        if key not in a:
            a[key] = b[key]
        else:
            a[key] = (a[key][0] + b[key][0], a[key][1] + b[key][1])
    return a
```

```
[10]: @task(returns=dict)
      def reduceCentersTask(*data):
          reduce_value = data[0]
          for i in range(1, len(data)):
              reduce_value = reduceCenters(reduce_value, data[i])
          return reduce_value
```

```
[11]: def mergeReduce(function, data, chunk=50):
          """ Apply function cumulatively to the items of data,
              from left to right in binary tree structure, so as to
              reduce the data to a single value.
          :param function: function to apply to reduce data
          :param data: List of items to be reduced
          :return: result of reduce the data to a single value
          """
          while(len(data)) > 1:
              dataToReduce = data[:chunk]
              data = data[chunk:]
              data.append(function(*dataToReduce))
          return data[0]
```

```
[12]: def has_converged(mu, oldmu, epsilon, iter, maxIterations):
          print("iter: " + str(iter))
          print("maxIterations: " + str(maxIterations))
          if oldmu != []:
              if iter < maxIterations:
                  aux = [np.linalg.norm(oldmu[i] - mu[i]) for i in range(len(mu))]
                  distancia = sum(aux)
                  if distancia < epsilon * epsilon:
                      print("Distance_T: " + str(distancia))
                      return True
                  else:
                      print("Distance_F: " + str(distancia))
                      return False
              else:
                  # Reached the max amount of iterations
                  return True
```

```
[13]: def plotKMEANS(dim, mu, clusters, data):
          import pylab as plt
          colors = ['b','g','r','c','m','y','k']
          if dim == 2 and len(mu) <= len(colors):
              from matplotlib.patches import Circle
              from matplotlib.collections import PatchCollection
              fig, ax = plt.subplots(figsize=(10,10))
              patches = []
              pcolors = []
```

```
            for i in range(len(clusters)):
                for key in clusters[i].keys():
                    d = clusters[i][key]
                    for j in d:
                        j = j - i * len(data[0])
                        C = Circle((data[i][j][0], data[i][j][1]), .05)
                        pcolors.append(colors[key])
                        patches.append(C)
            collection = PatchCollection(patches)
            collection.set_facecolor(pcolors)
            ax.add_collection(collection)
            x, y = zip(*mu)
            plt.plot(x, y, '*', c='y', markersize=20)
            plt.autoscale(enable=True, axis='both', tight=False)
            plt.show()
        elif dim == 3 and len(mu) <= len(colors):
            from mpl_toolkits.mplot3d import Axes3D
            fig = plt.figure()
            ax = fig.add_subplot(111, projection='3d')
            for i in range(len(clusters)):
                for key in clusters[i].keys():
                    d = clusters[i][key]
                    for j in d:
                        j = j - i * len(data[0])
                        ax.scatter(data[i][j][0], data[i][j][1], data[i][j][2], 'o',␣
→c=colors[key])
            x, y, z = zip(*mu)
            for i in range(len(mu)):
                ax.scatter(x[i], y[i], z[i], s=80, c='y', marker='D')
            plt.show()
        else:
            print("No representable dim or not enough colours")
```

10.2.3.1 MAIN

Parameters (that can be configured in the following cell): * numV: number of vectors (default: 10.000) * dim: dimension of the points (default: 2) * k: number of centers (default: 4) * numFrag: number of fragments (default: 16) * epsilon: convergence condition (default: 1e-10) * maxIterations: Maximum number of iterations (default: 20)

```
[14]: %matplotlib inline
import ipywidgets as widgets
from pycompss.api.api import compss_wait_on

w_numV = widgets.IntText(value=10000)          # Number of Vectors - with 1000 it is feasible␣
→to see the evolution across iterations
w_dim = widgets.IntText(value=2)               # Number of Dimensions
w_k = widgets.IntText(value=4)                 # Centers
w_numFrag = widgets.IntText(value=16)          # Fragments
w_epsilon = widgets.FloatText(value=1e-10)     # Convergence condition
w_maxIterations = widgets.IntText(value=20)    # Max number of iterations
w_seed = widgets.IntText(value=8)              # Random seed
```

```python
def kmeans(numV, dim, k, numFrag, epsilon, maxIterations, seed):
    size = int(numV / numFrag)
    cloudCenters = init_random(k, dim, seed) # centers to create data groups
    X = [genFragment(size, k, cloudCenters, dim, mode='gauss') for _ in range(numFrag)]
    mu = init_random(k, dim, seed - 1)        # First centers
    oldmu = []
    n = 0
    while not has_converged(mu, oldmu, epsilon, n, maxIterations):
        oldmu = mu
        clusters = [cluster_points_partial(X[f], mu, f * size) for f in range(numFrag)]
        partialResult = [partial_sum(X[f], clusters[f], f * size) for f in range(numFrag)]
        mu = mergeReduce(reduceCentersTask, partialResult, chunk=4)
        mu = compss_wait_on(mu)
        mu = [mu[c][1] / mu[c][0] for c in mu]
        while len(mu) < k:
            # Add new random center if one of the centers has no points.
            indP = np.random.randint(0, size)
            indF = np.random.randint(0, numFrag)
            mu.append(X[indF][indP])
        n += 1
    clusters = compss_wait_on(clusters)
    plotKMEANS(dim, mu, clusters, X)
    print("--------------------")
    print("Result:")
    print("Iterations: ", n)
    print("Centers: ", mu)
    print("--------------------")

widgets.interact_manual(kmeans, numV=w_numV, dim=w_dim, k=w_k, numFrag=w_numFrag, epsilon=w_
→epsilon, maxIterations=w_maxIterations, seed=w_seed)
```

```
interactive(children=(IntText(value=10000, description='numV'), IntText(value=2, description=
→'dim'), IntText(v...
```

```
[14]: <function __main__.kmeans(numV, dim, k, numFrag, epsilon, maxIterations, seed)>
```

```python
[15]: ipycompss.stop()
```

```
********************************************************
*************** STOPPING PyCOMPSs ******************
********************************************************
Checking if any issue happened.
Warning: some of the variables used with PyCOMPSs may
        have not been brought to the master.
********************************************************
```

10.2.4 Cholesky Decomposition/Factorization

Given a symmetric positive definite matrix A, the Cholesky decomposition is an upper triangular matrix U (with strictly positive diagonal entries) such that:

$A = U^T U$

```python
[1]: import pycompss.interactive as ipycompss
```

```
[2]:  # Start PyCOMPSs runtime with graph and tracing enabled
      import os
      if 'BINDER_SERVICE_HOST' in os.environ:
          ipycompss.start(graph=True, trace=True,
                          project_xml='../xml/project.xml',
                          resources_xml='../xml/resources.xml')
      else:
          ipycompss.start(graph=True, monitor=1000, trace=True)
```

```
********************************************************
**************** PyCOMPSs Interactive ******************
********************************************************
*          .-~~-.--.                ------        ---       *
*       :         )             |____ \       /   |     *
*     .~ ~ -.\        /.- ~~ .        __) |      /_/| |     *
*     >        `.   .'         <     |__  |         | |     *
*    (          .- -.          )   ____) |    _      | |     *
*     `- -.-~  `- -'  ~-.- -'   |_____/   |_|     |_|     *
*        (         :         )             _ _ .-:        *
*       ~--.    :    .--~        .-~  .-~ }      *
*            ~-.-^-.-~ \_        .~   .-~   .~        *
*                 \ \ '     \ '_ _ -~        *
*                  \`.\`.    //                    *
*           . - ~ ~-.__\`.\`-.//                    *
*         .-~     . - ~  }~ ~ ~-.~-.                  *
*       .' .-~     .-~       :/~-.~-./:               *
*      /_~_ _ . - ~                 ~-.~-._         *
*                               ~-.<       *
********************************************************
* - Starting COMPSs runtime...                         *
* - Log path : /home/user/.COMPSs/Interactive_20/       *
* - PyCOMPSs Runtime started... Have fun!              *
********************************************************
```

```
[3]:  from pycompss.api.task import task
      from scipy import linalg
      from scipy import random
      import numpy as np
      import ctypes
```

10.2.4.1 Task definitions

```
[4]:  @task(returns=list)
      def createBlock(BSIZE, MKLProc, diag):
          import os
          os.environ["MKL_NUM_THREADS"]=str(MKLProc)
          block = np.array(np.random.random((BSIZE, BSIZE)), dtype=np.double,copy=False)
          mb = np.matrix(block, dtype=np.double, copy=False)
          mb = mb + np.transpose(mb)
          if diag:
              mb = mb + 2*BSIZE*np.eye(BSIZE)
          return mb

      @task(returns=np.ndarray)
      def potrf(A, MKLProc):
```

<div align="right">(continues on next page)</div>

```python
    from scipy.linalg.lapack import dpotrf
    import os
    os.environ['MKL_NUM_THREADS']=str(MKLProc)
    A = dpotrf(A, lower=True)[0]
    return A

@task(returns=np.ndarray)
def solve_triangular(A, B, MKLProc):
    from scipy.linalg import solve_triangular
    from numpy import transpose
    import os
    os.environ['MKL_NUM_THREADS']=str(MKLProc)
    B = transpose(B)
    B = solve_triangular(A, B, lower=True)  # , trans='T'
    B = transpose(B)
    return B

@task(returns=np.ndarray)
def gemm(alpha, A, B, C, beta, MKLProc):
    from scipy.linalg.blas import dgemm
    from numpy import transpose
    import os
    os.environ['MKL_NUM_THREADS']=str(MKLProc)
    B = transpose(B)
    C = dgemm(alpha, A, B, c=C, beta=beta)
    return C
```

10.2.4.2 Auxiliar functions

```python
[5]: def genMatrix(MSIZE, BSIZE, MKLProc, A):
         for i in range(MSIZE):
             A.append([])
             for j in range(MSIZE):
                 A[i].append([])
         for i in range(MSIZE):
             mb = createBlock(BSIZE, MKLProc, True)
             A[i][i]=mb
             for j in range(i+1,MSIZE):
                 mb = createBlock(BSIZE, MKLProc, False)
                 A[i][j]=mb
                 A[j][i]=mb
```

```python
[6]: def cholesky_blocked(MSIZE, BSIZE, mkl_threads, A):
         import os
         for k in range(MSIZE):
             # Diagonal block factorization
             A[k][k] = potrf(A[k][k], mkl_threads)
             # Triangular systems
             for i in range(k+1, MSIZE):
                 A[i][k] = solve_triangular(A[k][k], A[i][k], mkl_threads)
                 A[k][i] = np.zeros((BSIZE,BSIZE))
             # update trailing matrix
             for i in range(k+1, MSIZE):
                 for j in range(i, MSIZE):
```

```
                A[j][i] = gemm(-1.0, A[j][k], A[i][k], A[j][i], 1.0, mkl_threads)
        return A
```

MAIN Code

Parameters (that can be configured in the following cell): * MSIZE: Matrix size (default: 8) * BSIZE: Block size (default: 1024) * mkl_threads: Number of MKL threads (default: 1)

```
[7]: import ipywidgets as widgets
     from pycompss.api.api import compss_barrier
     import time

     w_MSIZE = widgets.IntText(value=8)
     w_BSIZE = widgets.IntText(value=1024)
     w_mkl_threads = widgets.IntText(value=1)

     def cholesky(MSIZE, BSIZE, mkl_threads):
         # Generate de matrix
         startTime = time.time()

         # Generate supermatrix
         A = []
         res = []
         genMatrix(MSIZE, BSIZE, mkl_threads, A)
         compss_barrier()

         initTime = time.time() - startTime
         startDecompTime = time.time()
         res = cholesky_blocked(MSIZE, BSIZE, mkl_threads, A)
         compss_barrier()

         decompTime = time.time() - startDecompTime
         totalTime = decompTime + initTime

         print("---------- Elapsed Times ----------")
         print("initT:{}".format(initTime))
         print("decompT:{}".format(decompTime))
         print("totalTime:{}".format(totalTime))
         print("-----------------------------------")

     widgets.interact_manual(cholesky, MSIZE=w_MSIZE, BSIZE=w_BSIZE, mkl_threads=w_mkl_threads)
```

```
     interactive(children=(IntText(value=8, description='MSIZE'), IntText(value=1024, description=
     ↪'BSIZE'), IntText...
```

```
[7]: <function __main__.cholesky(MSIZE, BSIZE, mkl_threads)>
```

```
[8]: ipycompss.stop()
```

```
     *******************************************************
     *************** STOPPING PyCOMPSs ******************
     *******************************************************
     Checking if any issue happened.
     Warning: some of the variables used with PyCOMPSs may
             have not been brought to the master.
     *******************************************************
```

10.2.5 Wordcount Exercise

10.2.5.1 Sequential version

```
[1]: import os
```

```
[2]: def read_file(file_path):
         """ Read a file and return a list of words.
         :param file_path: file's path
         :return: list of words
         """
         data = []
         with open(file_path, 'r') as f:
             for line in f:
                 data += line.split()
         return data
```

```
[3]: def wordCount(data):
         """ Construct a frequency word dictorionary from a list of words.
         :param data: a list of words
         :return: a dictionary where key=word and value=#appearances
         """
         partialResult = {}
         for entry in data:
             if entry in partialResult:
                 partialResult[entry] += 1
             else:
                 partialResult[entry] = 1
         return partialResult
```

```
[4]: def merge_two_dicts(dic1, dic2):
         """ Update a dictionary with another dictionary.
         :param dic1: first dictionary
         :param dic2: second dictionary
         :return: dic1+=dic2
         """
         for k in dic2:
             if k in dic1:
                 dic1[k] += dic2[k]
             else:
                 dic1[k] = dic2[k]
         return dic1
```

```
[5]: # Get the dataset path
     pathDataset = os.getcwd() + '/dataset'

     # Read file's content execute a wordcount on each of them
     partialResult = []
     for fileName in os.listdir(pathDataset):
         file_path = os.path.join(pathDataset, fileName)
         data = read_file(file_path)
         partialResult.append(wordCount(data))

     # Accumulate the partial results to get the final result.
     result = {}
     for partial in partialResult:
```

```
    result = merge_two_dicts(result, partial)
```

```
[6]: print("Result:")
     from pprint import pprint
     pprint(result)
     print("Words: {}".format(sum(result.values())))
```

```
Result:
{'Adipisci': 227,
 'Aliquam': 233,
 'Amet': 207,
 'Consectetur': 201,
 'Dolor': 198,
 'Dolore': 236,
 'Dolorem': 232,
 'Eius': 251,
 'Est': 197,
 'Etincidunt': 232,
 'Ipsum': 228,
 'Labore': 229,
 'Magnam': 195,
 'Modi': 201,
 'Neque': 205,
 'Non': 226,
 'Numquam': 253,
 'Porro': 205,
 'Quaerat': 217,
 'Quiquia': 212,
 'Quisquam': 214,
 'Sed': 225,
 'Sit': 220,
 'Tempora': 189,
 'Ut': 217,
 'Velit': 218,
 'Voluptatem': 235,
 'adipisci': 1078,
 'aliquam': 1107,
 'amet': 1044,
 'consectetur': 1073,
 'dolor': 1120,
 'dolore': 1065,
 'dolorem': 1107,
 'eius': 1048,
 'est': 1101,
 'etincidunt': 1114,
 'ipsum': 1061,
 'labore': 1070,
 'magnam': 1096,
 'modi': 1127,
 'neque': 1093,
 'non': 1099,
 'numquam': 1094,
 'porro': 1101,
 'quaerat': 1086,
 'quiquia': 1079,
 'quisquam': 1144,
```

```
 'sed': 1109,
 'sit': 1130,
 'tempora': 1064,
 'ut': 1070,
 'velit': 1105,
 'voluptatem': 1121}
Words: 35409
```

10.2.6 Wordcount Solution

10.2.6.1 Complete version

```
[1]: import os
```

```
[2]: import pycompss.interactive as ipycompss
```

```
[3]: from pycompss.api.task import task
```

```
[4]: from pycompss.api.parameter import *
```

```
[5]: if 'BINDER_SERVICE_HOST' in os.environ:
         ipycompss.start(graph=True, trace=True, debug=False,
                         project_xml='../xml/project.xml',
                         resources_xml='../xml/resources.xml')
     else:
         ipycompss.start(graph=True, monitor=1000, trace=True, debug=False)
```

```
********************************************************
*************** PyCOMPSs Interactive ******************
********************************************************
*          .-~~-.--.                  _____          ___        *
*         :         )                |____  \       /   |        *
*   .~ ~ -.\       /.- ~~ .            __) |      /_/| |        *
*   >       `.   .'       <          |__  |         | |        *
*   (         .- -.         )        ____) |    _    | |        *
*    `- -.-~  `- -'  ~-.- -'         |_____/   |_|    |_|        *
*      (        :        )            _ _ .-:        *
*       ~--.    :    .--~          .-~  .-~  }        *
*           ~-.-^-.-~ \_        .~  .-~   .~        *
*                    \ \ '      \ '_ _ -~        *
*                     \`.\`.    //        *
*            . - ~ ~-.__\`.\`-.//        *
*        .-~    . - ~  }~ ~ ~-.~-.        *
*      .' .-~      .-~       :/~-.~-./:        *
*     /_~_ _ . - ~                 ~-.~-._        *
*                                     ~-.<        *
********************************************************
* - Starting COMPSs runtime...                        *
* - Log path : /home/user/.COMPSs/Interactive_21/
* - PyCOMPSs Runtime started... Have fun!             *
********************************************************
```

```
[6]: @task(returns=list)
     def read_file(file_path):
         """ Read a file and return a list of words.
         :param file_path: file's path
         :return: list of words
         """
         data = []
         with open(file_path, 'r') as f:
             for line in f:
                 data += line.split()
         return data
```

```
[7]: @task(returns=dict)
     def wordCount(data):
         """ Construct a frequency word dictionary from a list of words.
         :param data: a list of words
         :return: a dictionary where key=word and value=#appearances
         """
         partialResult = {}
         for entry in data:
             if entry in partialResult:
                 partialResult[entry] += 1
             else:
                 partialResult[entry] = 1
         return partialResult
```

```
[8]: @task(returns=dict, priority=True)
     def merge_two_dicts(dic1, dic2):
         """ Update a dictionary with another dictionary.
         :param dic1: first dictionary
         :param dic2: second dictionary
         :return: dic1+=dic2
         """
         for k in dic2:
             if k in dic1:
                 dic1[k] += dic2[k]
             else:
                 dic1[k] = dic2[k]
         return dic1
```

```
[9]: from pycompss.api.api import compss_wait_on

     # Get the dataset path
     pathDataset = os.getcwd() + '/dataset'

     # Read file's content execute a wordcount on each of them
     partialResult = []
     for fileName in os.listdir(pathDataset):
         file_path = os.path.join(pathDataset, fileName)
         data = read_file(file_path)
         partialResult.append(wordCount(data))

     # Accumulate the partial results to get the final result.
     result = {}
     for partial in partialResult:
         result = merge_two_dicts(result, partial)
```

(continues on next page)

```python
# Wait for result
result = compss_wait_on(result)
```

```
Found task: read_file
Found task: wordCount
Found task: merge_two_dicts
```

```python
[10]: print("Result:")
      from pprint import pprint
      pprint(result)
      print("Words: {}".format(sum(result.values())))
```

```
Result:
{'Adipisci': 227,
 'Aliquam': 233,
 'Amet': 207,
 'Consectetur': 201,
 'Dolor': 198,
 'Dolore': 236,
 'Dolorem': 232,
 'Eius': 251,
 'Est': 197,
 'Etincidunt': 232,
 'Ipsum': 228,
 'Labore': 229,
 'Magnam': 195,
 'Modi': 201,
 'Neque': 205,
 'Non': 226,
 'Numquam': 253,
 'Porro': 205,
 'Quaerat': 217,
 'Quiquia': 212,
 'Quisquam': 214,
 'Sed': 225,
 'Sit': 220,
 'Tempora': 189,
 'Ut': 217,
 'Velit': 218,
 'Voluptatem': 235,
 'adipisci': 1078,
 'aliquam': 1107,
 'amet': 1044,
 'consectetur': 1073,
 'dolor': 1120,
 'dolore': 1065,
 'dolorem': 1107,
 'eius': 1048,
 'est': 1101,
 'etincidunt': 1114,
 'ipsum': 1061,
 'labore': 1070,
 'magnam': 1096,
 'modi': 1127,
 'neque': 1093,
 'non': 1099,
```

```
 'numquam': 1094,
 'porro': 1101,
 'quaerat': 1086,
 'quiquia': 1079,
 'quisquam': 1144,
 'sed': 1109,
 'sit': 1130,
 'tempora': 1064,
 'ut': 1070,
 'velit': 1105,
 'voluptatem': 1121}
Words: 35409
```

```
[11]: ipycompss.stop()
```

```
********************************************************
*************** STOPPING PyCOMPSs ******************
********************************************************
Checking if any issue happened.
Warning: some of the variables used with PyCOMPSs may
         have not been brought to the master.
********************************************************
```

10.2.7 Wordcount Solution (With reduce)

10.2.7.1 Complete version

```
[1]: import os
```

```
[2]: import pycompss.interactive as ipycompss
```

```
[3]: from pycompss.api.task import task
```

```
[4]: from pycompss.api.parameter import *
```

```
[5]: if 'BINDER_SERVICE_HOST' in os.environ:
         ipycompss.start(graph=True, trace=True, debug=False,
                         project_xml='../xml/project.xml',
                         resources_xml='../xml/resources.xml')
     else:
         ipycompss.start(graph=True, monitor=1000, trace=True, debug=False)
```

```
********************************************************
*************** PyCOMPSs Interactive ******************
********************************************************
*          .-~~-.--.            _____        ___       *
*         :         )          |____  \      /   |      *
*   .~ ~ -.\       /.- ~~ .       __) |     /_/| |      *
*   >       `.   .'       <      |__  |        | |      *
*   (         .- -.        )    ____) |   _    | |      *
*   `- -.-~  `- -'  ~-.- -'    |_____/  |_|   |_|      *
*     (        :        )           _ _ .-:            *
*      ~--.    :    .--~         .-~  .-~  }           *
*          ~-.-^-.-~ \_       .~  .-~   .~            *
```

```
*                    \ \ '      \ '_ _ _~            *
*                     \`.\`.     //                  *
*          . - ~ ~-.__\`.\`-.//                      *
*       .-~    . - ~  }~ ~ ~-.~-.                    *
*     .' .-~        .-~        :/~-.~-./:            *
*    /_~_ _ . - ~                    ~-.~-._         *
*                                       ~-.<         *
*****************************************************
* - Starting COMPSs runtime...                      *
* - Log path : /home/user/.COMPSs/Interactive_22/   *
* - PyCOMPSs Runtime started... Have fun!            *
*****************************************************
```

[6]:
```python
@task(returns=list)
def read_file(file_path):
    """ Read a file and return a list of words.
    :param file_path: file's path
    :return: list of words
    """
    data = []
    with open(file_path, 'r') as f:
        for line in f:
            data += line.split()
    return data
```

[7]:
```python
@task(returns=dict)
def wordCount(data):
    """ Construct a frequency word dictionary from a list of words.
    :param data: a list of words
    :return: a dictionary where key=word and value=#appearances
    """
    partialResult = {}
    for entry in data:
        if entry in partialResult:
            partialResult[entry] += 1
        else:
            partialResult[entry] = 1
    return partialResult
```

[8]:
```python
@task(returns=dict, priority=True)
def merge_dicts(*dictionaries):
    import queue
    q = queue.Queue()
    for i in dictionaries:
        q.put(i)
    while not q.empty():
        x = q.get()
        if not q.empty():
            y = q.get()
            for k in y:
                if k in x:
                    x[k] += y[k]
                else:
                    x[k] = y[k]
            q.put(x)
```

```
        return(x)
```

```python
[9]: from pycompss.api.api import compss_wait_on

     # Get the dataset path
     pathDataset = os.getcwd() + '/dataset'

     # Construct a list with the file's paths from the dataset
     partialResult = []
     for fileName in os.listdir(pathDataset):
         p = os.path.join(pathDataset, fileName)
         data=read_file(p)
         partialResult.append(wordCount(data))

     # Accumulate the partial results to get the final result.
     result=merge_dicts(*partialResult)

     # Wait for result
     result = compss_wait_on(result)
```

```
Found task: read_file
Found task: wordCount
Found task: merge_dicts
```

```python
[10]: print("Result:")
      from pprint import pprint
      pprint(result)
      print("Words: {}".format(sum(result.values())))
```

```
Result:
{'Adipisci': 227,
 'Aliquam': 233,
 'Amet': 207,
 'Consectetur': 201,
 'Dolor': 198,
 'Dolore': 236,
 'Dolorem': 232,
 'Eius': 251,
 'Est': 197,
 'Etincidunt': 232,
 'Ipsum': 228,
 'Labore': 229,
 'Magnam': 195,
 'Modi': 201,
 'Neque': 205,
 'Non': 226,
 'Numquam': 253,
 'Porro': 205,
 'Quaerat': 217,
 'Quiquia': 212,
 'Quisquam': 214,
 'Sed': 225,
 'Sit': 220,
 'Tempora': 189,
 'Ut': 217,
 'Velit': 218,
```

```
 'Voluptatem': 235,
 'adipisci': 1078,
 'aliquam': 1107,
 'amet': 1044,
 'consectetur': 1073,
 'dolor': 1120,
 'dolore': 1065,
 'dolorem': 1107,
 'eius': 1048,
 'est': 1101,
 'etincidunt': 1114,
 'ipsum': 1061,
 'labore': 1070,
 'magnam': 1096,
 'modi': 1127,
 'neque': 1093,
 'non': 1099,
 'numquam': 1094,
 'porro': 1101,
 'quaerat': 1086,
 'quiquia': 1079,
 'quisquam': 1144,
 'sed': 1109,
 'sit': 1130,
 'tempora': 1064,
 'ut': 1070,
 'velit': 1105,
 'voluptatem': 1121}
Words: 35409
```

```
[11]: ipycompss.stop()
```

```
*********************************************************
*************** STOPPING PyCOMPSs ******************
*********************************************************
Checking if any issue happened.
Warning: some of the variables used with PyCOMPSs may
        have not been brought to the master.
*********************************************************
```

10.2.8 Integral PI (iterative)

10.2.8.1 1.1 Initialize PyCOMPSs

```
[1]: import pycompss.interactive as ipycompss
```

```
[2]: import os
     if 'BINDER_SERVICE_HOST' in os.environ:
         ipycompss.start(graph=True,                    # trace=True
                         project_xml='../xml/project.xml',
                         resources_xml='../xml/resources.xml')
     else:
         ipycompss.start(graph=True, monitor=1000, debug=True)  # trace=True
```

```
********************************************************
*************** PyCOMPSs Interactive ******************
********************************************************
*          .-~~-.--.                  _____        _____        *
*       :            )           |____  \       / __  \       *
*     .~ ~ -.\        /.- ~~ .         __) |        | |  | |       *
*     >        `.      .'        <      |__  |        | |  | |       *
*    (          .- -.              )   ___) |    _   | |__| |       *
*     `- -.-~  `- -'  ~-.- -'    |_____/   |_| _____/      *
*      (         :             )          _ _ .-:          *
*      ~--.      :       .--~        .-~  .-~  }        *
*            ~-.-^.-~ \_      .~  .-~    .~          *
*                 \ \ '    \ '_ _ -~          *
*                  \`.\`.     //              *
*            . - ~ ~-.__\`.\`-.//              *
*         .-~    . - ~  }~ ~ ~-.~-.              *
*       .' .-~       .-~          :/~-.~-./:          *
*      /_~_ _ . - ~                ~-.~-._          *
*                                   ~-.<          *
********************************************************
* - Starting COMPSs runtime...                            *
* - Log path : /home/user/.COMPSs/Interactive_21/
* - PyCOMPSs Runtime started... Have fun!            *
********************************************************
```

10.2.8.2 1.2 Required imports

```
[3]: from pycompss.api.api import compss_wait_on
     from pycompss.api.task import task
     from pycompss.api.parameter import *
```

10.2.8.3 2 Tasks Declaration

```
[4]: import numpy as np
```

```
[5]: @task(returns=float)
     def calculate_area(i, num_steps, number_of_batches, step_size):
         partial_area_sum = 0
         for i in range(i, num_steps, number_of_batches):
             x = (i+0.5) * step_size
             partial_area_sum += 4 / (1 + x**2)
         return partial_area_sum
```
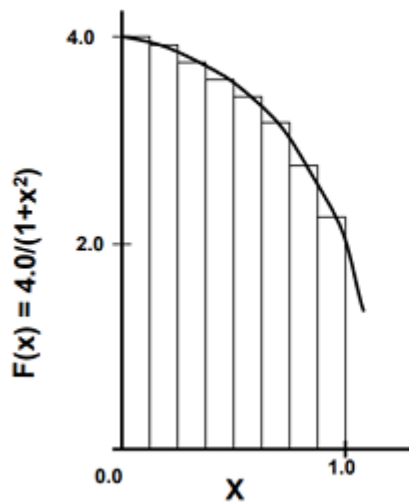
Mathematically, we know that:

$$\int_0^1 \frac{4.0}{(1+x^2)}\ dx = \pi$$

We can approximate the integral as a sum of rectangles:

$$\sum_{i=0}^{N} F(x_i)\Delta x \approx \pi$$

Where each rectangle has width Δx and height $F(x_i)$ at the middle of interval i.

```
[6]: @task(returns=float)
     def sum_areas(partial_area, total_area):
         total_area += partial_area
         return total_area
```

Run the algorithm

```
[7]: num_steps = 100000
     number_of_batches = 10
```

```
[8]: step_size = 1 / num_steps
```

```
[9]: total_area = 0
     for i in range(number_of_batches):
         partial_area = calculate_area(i, num_steps, number_of_batches, step_size)
         total_area = sum_areas(partial_area, total_area)
```

```
Task definition detected.
Found task: calculate_area
Task definition detected.
Found task: sum_areas
```

Wait for all tasks to finish and gather the result

```
[10]: total_area = compss_wait_on(total_area)
```

Calculate PI

```
[11]: pi = step_size * total_area
```

```
[12]: print('PI:', pi, 'Error:', abs(np.pi-pi))
```

```
PI: 3.141592653598127 Error: 8.333778112046275e-12
```

```
[13]: ipycompss.stop()
```

```
********************************************************
*************** STOPPING PyCOMPSs ******************
```

```
********************************************************
Checking if any issue happened.
Warning: some of the variables used with PyCOMPSs may
         have not been brought to the master.
********************************************************
```

10.2.9 Integral PI (with @reduction)

10.2.9.1 1.1 Initialize PyCOMPSs

```
[1]: import pycompss.interactive as ipycompss
```

```
[2]: import os
     if 'BINDER_SERVICE_HOST' in os.environ:
         ipycompss.start(graph=True,                     # trace=True
                         project_xml='../xml/project.xml',
                          resources_xml='../xml/resources.xml')
     else:
         ipycompss.start(graph=True, monitor=1000, debug=True)  # trace=True
```

```
********************************************************
**************** PyCOMPSs Interactive ******************
********************************************************
*          .-~~-.--.                _____        _____        *
*         :         )              |____ \      /  __  \      *
*    .~ ~ -.\       /.- ~~ .         __) |      | |  | |      *
*    >       `.   .'       <        |__  |      | |  | |      *
*   (         .- -.         )      ___) |   _  | |__| |      *
*    `- -.-~  `- -'  ~-.- -'      |_____/  |_| _____/      *
*      (        :        )           _ _ .-:                  *
*       ~--.    :    .--~         .-~  .-~  }                 *
*           ~-.-^-.-~ \_      .~  .-~   .~                   *
*                    \ \ '     \ '_ _ -~                     *
*                     \`.\`.    //                          *
*            . - ~ ~-.__\`.\`-.//                            *
*        .-~    . - ~  }~ ~ ~-.~-.                           *
*      .' .-~      .-~       :/~-.~-./:                      *
*     /_~_ _ . - ~                 ~-.~-._                   *
*                                      ~-.<                  *
********************************************************
* - Starting COMPSs runtime...                         *
* - Log path : /home/user/.COMPSs/Interactive_22/      *
* - PyCOMPSs Runtime started... Have fun!              *
********************************************************
```

10.2.9.2 1.2 Required imports

```
[3]: from pycompss.api.api import compss_wait_on
     from pycompss.api.task import task
     from pycompss.api.reduction import reduction
     from pycompss.api.parameter import *
```

10.2.9.3 2 Tasks Declaration

```
[4]: import numpy as np
```

```
[5]: @task(returns=float)
     def calculate_area(i, num_steps, number_of_batches, step_size):
         partial_area_sum = 0
         for i in range(i, num_steps, number_of_batches):
             x = (i+0.5) * step_size
             partial_area_sum += 4 / (1 + x**2)
         return partial_area_sum
```

Mathematically, we know that:

$$\int_0^1 \frac{4.0}{(1+x^2)} \, dx = \pi$$

We can approximate the integral as a sum of rectangles:

$$\sum_{i=0}^{N} F(x_i)\Delta x \approx \pi$$

Where each rectangle has width Δx and height $F(x_i)$ at the middle of interval i.
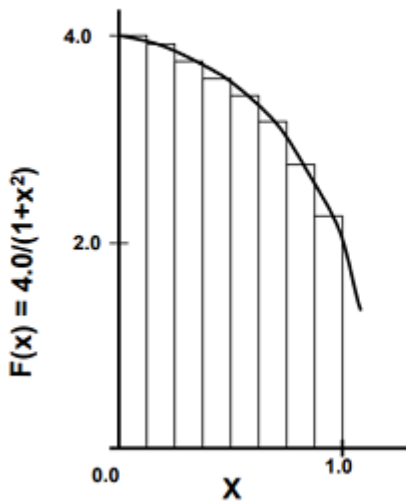

```
[6]: @reduction(chunk_size="2")
     @task(returns=float, batches_partial_areas=COLLECTION_IN)
     def sum_reduction(batches_partial_areas):
         total_area = 0
         for partial_area in batches_partial_areas:
             total_area += partial_area
         return total_area
```

Run the algorithm

```
[7]: num_steps = 100000
     number_of_batches = 10
```

```
[8]: step_size = 1 / num_steps
```

```
[9]: batches_partial_areas = []
     for i in range(number_of_batches):
         partial_area = calculate_area(i, num_steps, number_of_batches, step_size)
         batches_partial_areas.append(partial_area)


     total_area = sum_reduction(batches_partial_areas)
```

```
Task definition detected.
Found task: calculate_area
Task definition detected.
Found task: sum_reduction
```

Wait for all tasks to finish and gather the result

```
[10]: total_area = compss_wait_on(total_area)
```

Calculate PI

```
[11]: pi = step_size * total_area
```

```
[12]: print('PI:', pi, 'Error:', abs(np.pi-pi))
```

```
PI: 3.141592653598127 Error: 8.333778112046275e-12
```

```
[13]: ipycompss.stop()
```

```
********************************************************
*************** STOPPING PyCOMPSs ******************
********************************************************
Checking if any issue happened.
Warning: some of the variables used with PyCOMPSs may
         have not been brought to the master.
********************************************************
```

10.3 Demos

Here you will find the demonstration notebooks used in the tutorials.

10.3.1 Accelerating parallel code with PyCOMPSs and Numba

10.3.1.1 Demo Supercomputing 2019

What is mandelbrot?

The mandelbrot set is a fractal, which is plotted on the complex plane. It shows how intrincate can be formed from a simple equation.

It is generated using the algorithm:

$$Z_{n+1} = z_n^2 + A \tag{1}$$

$$\tag{2}$$

Where Z and A are complex numbers, and n represents the number of iterations.

First, import time to measure the elapsed execution times and create an ordered dictionary to keep all measures -> we are going to measure and plot the performance with different conditions!

```
[1]: import time
     from collections import OrderedDict
     times = OrderedDict()
```

And then, all required imports

```
[2]: from numpy import NaN, arange, abs, array
```

Mandelbrot set implementation:

```
[3]: def mandelbrot(a, max_iter):
         z = 0
         for n in range(1, max_iter):
             z = z**2 + a
             if abs(z) > 2:
                 return n
         return NaN
```

```
[4]: def mandelbrot_set(y, X, max_iter):
         Z = [0 for _ in range(len(X))]
         for ix, x in enumerate(X):
             Z[ix] = mandelbrot(x + 1j * y, max_iter)
         return Z
```

Main function to generate the mandelbrot set. It splits the space in vertical chunks, and calculates the mandelbrot set of each one, generating the result Z.

```
[5]: def run_mandelbrot(X, Y, max_iter):
         st = time.time()
         Z = [[] for _ in range(len(Y))]
         for iy, y in enumerate(Y):
             Z[iy] = mandelbrot_set(y, X, max_iter)
         elapsed = time.time() - st
         print("Elapsed time (s): {}".format(elapsed))
         return Z, elapsed
```

The following function plots the fractal inline (the coerced parameter ** is used to set NaN in coerced elements within Z).

```
[6]: %matplotlib inline
     def plot_fractal(Z, coerced):
         if coerced:
             Z = [[NaN if c == -2**63 else c for c in row] for row in Z]
         import matplotlib.pyplot as plt
         Z = array(Z)
         plt.imshow(Z, cmap='plasma')
         plt.show()
```

Define a benchmarking function:

```
[7]: def generate_fractal(coerced=False):
         X = arange(-2, .5, .01)
         Y = arange(-1.0,  1.0, .01)
         max_iterations = 2000
         Z, elapsed = run_mandelbrot(X, Y, max_iterations)
         plot_fractal(Z, coerced)
         return elapsed
```

Run the previous code **sequentially**:

```
[8]: times['Sequential'] = generate_fractal()
```

```
Elapsed time (s): 28.323102712631226
```
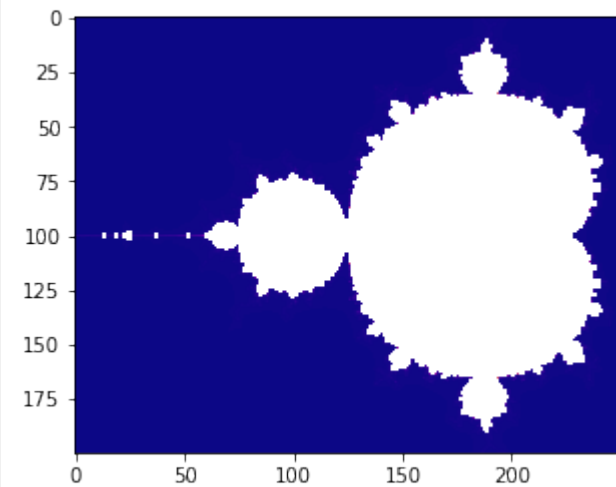

10.3.1.2 Paralellization with PyCOMPSs

After analysing the code, each mandelbrot set can be considered as a task, requiring only to decorate the `mandelbrot_set` function. It is interesting to observe that all sets are independent among them, so they can be computed completely independently, enabling to exploit multiple resources concurrently.

In order to run this code with we need first to start the **COMPSs** runtime:

```
[9]: import os
     import pycompss.interactive as ipycompss
     if 'BINDER_SERVICE_HOST' in os.environ:
         ipycompss.start(project_xml='../xml/project.xml',
                         resources_xml='../xml/resources.xml')
     else:
         ipycompss.start(graph=False, trace=True, monitor=1000)
```

```
********************************************************
**************** PyCOMPSs Interactive ******************
********************************************************
*          .-~~-.--.                  _____        ___      *
*         :         )                |_____ \      / _|     *
*   .~ ~ -.\       /.- ~~ .            __) |      /_/| |     *
*   >       `.   .'       <           |__  |         | |     *
*   (         .- -.         )        ____) |    _    | |     *
*    `- -.-~  `- -'  ~-.- -'         |_____/   |_|    |_|    *
*      (        :        )               _ _ .-:              *
*       ~--.    :    .--~              .-~  .-~  }             *
*           ~-.-^-.-~ \_          .~  .-~    .~               *
*                    \ \ '     \ '_ _ -~               *
*                     \`.\`.    //                            *
*            . - ~ ~-.__\`.\`-.//                             *
*        .-~   . - ~  }~ ~ ~-.~-.                             *
*      .' .-~      .-~       :/~-.~-./:                        *
*     /_~_ _ . - ~                 ~-.~-._                    *
*                                     ~-.<                    *
********************************************************
```

```
* - Starting COMPSs runtime...                          *
* - Log path : /home/user/.COMPSs/Interactive_16/
* - PyCOMPSs Runtime started... Have fun!               *
*********************************************************
```

It is necessary to decorate the `mandelbrot_set` function with the `@task` decorator.

Note that the `mandelbrot_set` function returns a list of elements.

```
[10]: from pycompss.api.task import task
```

```
[11]: @task(returns=list)
      def mandelbrot_set(y, X, max_iter):
          Z = [0 for _ in range(len(X))]
          for ix, x in enumerate(X):
              Z[ix] = mandelbrot(x + 1j * y, max_iter)
          return Z
```

And finally, include the synchronization of `Z` with `compss_wait_on`.

```
[12]: from pycompss.api.api import compss_wait_on
```

```
[13]: def run_mandelbrot(X, Y, max_iter):
          st = time.time()
          Z = [[] for _ in range(len(Y))]
          for iy, y in enumerate(Y):
              Z[iy] = mandelbrot_set(y, X, max_iter)
          Z = compss_wait_on(Z)
          elapsed = time.time() - st
          print("Elapsed time (s): {}".format(elapsed))
          return Z, elapsed
```

Run the benchmark with **PyCOMPSs**:

```
[14]: times['PyCOMPSs'] = generate_fractal()
```

```
Found task: mandelbrot_set
Elapsed time (s): 15.903970718383789
```
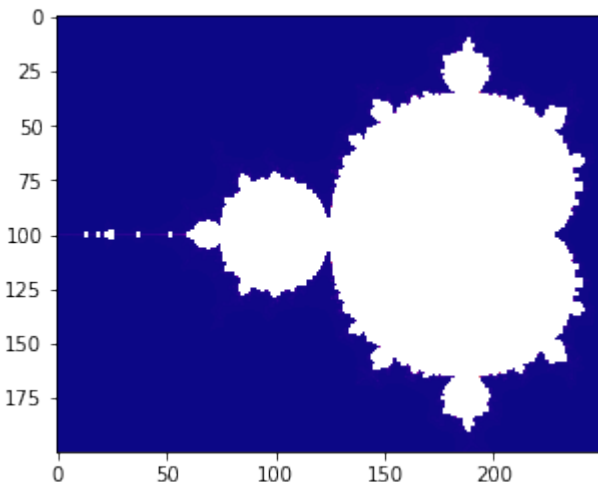
10.3.1.3 Accelerating the tasks with Numba

To this end, it is necessary to either use: 1. the Numba's `@jit` decorator under the PyCOMPSs `@task` decorator 2. or define the `numba=True` within the `@task` decorator.

First, we decorate the inner function (`mandelbrot`) with `@jit` since it is also a target function to be optimized with Numba.

```
[15]: from numba import jit

      @jit
      def mandelbrot(a, max_iter):
          z = 0
          for n in range(1, max_iter):
              z = z**2 + a
              if abs(z) > 2:
                  return n
          return NaN # NaN is coerced by Numba
```

Option 1 - Add the `@jit` decorator explicitly under `@task` decorator

@task(returns=list) @jit def mandelbrot_set(y, X, max_iter): Z = [0 for _ in range(len(X))] for ix, x in enumerate(X): Z[ix] = mandelbrot(x + 1j * y, max_iter) return Z

Option 2 - Add the `numba=True` flag within `@task` decorator

```
[16]: @task(returns=list, numba=True)
      def mandelbrot_set(y, X, max_iter):
          Z = [0 for _ in range(len(X))]
          for ix, x in enumerate(X):
              Z[ix] = mandelbrot(x + 1j * y, max_iter)
          return Z
```

Run the benchmark with **Numba**:

```
[17]: times['PyCOMPSs + Numba'] = generate_fractal(coerced=True)
```

```
Found task: mandelbrot_set
Elapsed time (s): 10.109647274017334
```
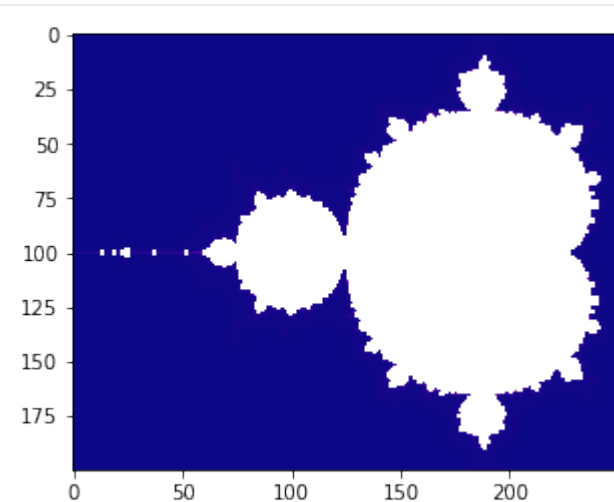

Plot the times:

```
[18]: import matplotlib.pyplot as plt
      plt.bar(*zip(*times.items()))
      plt.show()
```
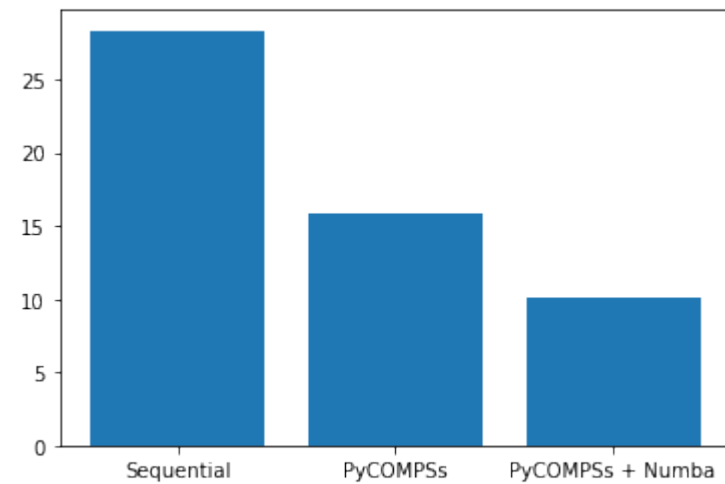

Stop **COMPSs** runtime

```
[19]: ipycompss.stop()
```

```
*********************************************************
*************** STOPPING PyCOMPSs ******************
*********************************************************
Checking if any issue happened.
Warning: some of the variables used with PyCOMPSs may
        have not been brought to the master.
*********************************************************
```

Hint: These notebooks can be used within **MyBinder**, with the **PyCOMPSs CLI**, within **Docker**, within **Virtual Machine** (recommended for Windows) provided by BSC, or locally.

Prerequisites

- Using *MyBinder*:
    - Open [launch binder]

    > **Caution:**   Sometimes it may take a while to deploy the COMPSs infrastructure.

- Using **PyCOMPSs CLI**:
    - pycompss-cli (see *Requirements and Installation*)
- Using **Docker**:
    - Docker
    - Git
- Using **Virtual Machine**:
    - VirtualBox
- For **local** execution:
    - Python 2 or 3
    - Install COMPSs requirements described in *Dependencies*.
    - Install COMPSs (See *Building from sources*)
    - Jupyter (with the desired ipykernel)
    - ipywidgets (only for some hands-on notebooks)
    - numpy (only for some notebooks)
    - dislib (only for some notebooks)
    - numba (only for some notebooks)

– Git

Instructions

- Using **MyBinder**:

  Just explore the folders and run the examples (they have the same structure as this documentation).

- Using `pycompss-cli`:

  Check the `pycompss-cli` usage instructions (see *Usage*)

  Get the notebooks:

  ```
  $ git clone https://github.com/bsc-wdc/notebooks.git
  ```

- Using **Docker**:

  Run in your machine:

  ```
  $ git clone https://github.com/bsc-wdc/notebooks.git
  $ docker pull compss/compss:3.1
  $ # Update the path to the notebooks path in the next command before running␣
  ↪it
  $ docker run --name mycompss -p 8888:8888 -p 8080:8080 -v /PATH/TO/notebooks:/
  ↪home/notebooks -itd compss/compss:3.1
  $ docker exec -it mycompss /bin/bash
  ```

  Now that docker is running and you are connected:

  ```
  $ cd /home/notebooks
  $ /etc/init.d/compss-monitor start
  $ jupyter-notebook --no-browser --allow-root --ip=172.17.0.2 --NotebookApp.
  ↪token=
  ```

  From local web browser:

  ```
  Open COMPSs monitor: http://localhost:8080/compss-monitor/index.zul
  Open Jupyter notebook interface: http://localhost:8888/
  ```

- Using **Virtual Machine**:

  – Download the OVA from: https://www.bsc.es/research-and-development/software-and-apps/software-list/comp-superscalar/downloads (*Look for Virtual Appliances section*)

  – Import the OVA from VirtualBox

  – Start the Virtual Machine

   * User: **compss**
   * Password: **compss2019**

  – Open a console and run:

  ```
  $ git clone https://github.com/bsc-wdc/notebooks.git
  $ cd notebooks
  $ /etc/init.d/compss-monitor start
  $ jupyter-notebook
  ```

  – Open the web browser:

  ```
  * Open COMPSs monitor: http://localhost:8080/compss-monitor/index.zul
  * Open Jupyter notebook interface: http://localhost:8888/
  ```

- Using local installation

  – Get the notebooks and start jupyter

  ```
  $ git clone https://github.com/bsc-wdc/notebooks.git
  $ cd notebooks
  $ /etc/init.d/compss-monitor start
  $ jupyter-notebook
  ```

  – Then

```
* Open COMPSs monitor: http://localhost:8080/compss-monitor/index.zul
* Open Jupyter notebook interface: http://localhost:8888/
* Look for the application.ipynb of interest.
```

Important: It is necessary to **RESTART the python kernel from Jupyter** after the execution of any notebook.

Troubleshooting
- ISSUE 1: Cannot connect using docker pull.
  REASON: *The docker service is not running*:

```
$ # Error messsage:
$ Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the␣
→docker daemon running?
$ # SOLUTION: Restart the docker service:
$ sudo service docker start
```

- ISSUE 2: The notebooks folder is empty or contains other data using docker.
  REASON: *The notebooks path in the docker run command is wrong.*

```
$ # Remove the docker instance and reinstantiate with the appropriate␣
→notebooks path
$ exit
$ docker stop mycompss
$ docker rm mycompss
$ # Pay attention and UPDATE: /PATH/TO in the next command
$ docker run --name mycompss -p 8888:8888 -p 8080:8080 -v /PATH/TO/notebooks:/
→home/notebooks -itd compss/compss:3.1
$ # Continue as normal
```

- ISSUE 3: COMPSs does not start in Jupyter.
  REASON: *The python kernel has not been restarted between COMPSs start, or some processes from previous failed execution may exist.*

```
$ # SOLUTION: Restart the python kernel from Jupyter and check that there are␣
→no COMPSs' python/java processes running.
```

- ISSUE 4: Numba is not working with the VM or Docker.
  REASON: *Numba is not installed in the VM or docker*

```
$ # SOLUTION: Install Numba in the VM/Docker
$ #           Open a console in the VM/Docker and follow the next steps.
$ # For Python 2:
$ sudo python2 -m pip install numba
$ # For Python 3:
$ sudo python3 -m pip install numba
```

- ISSUE 5: Matplotlib is not working with the VM or Docker.
  REASON: *Matplotlib is not installed in the VM or docker*

```
$ # SOLUTION: Install Matplotlib in the VM/Docker
$ #           Open a console in the VM/Docker and follow the next steps.
$ # For Python 2:
$ sudo python2 -m pip install matplotlib
$ # For Python 3:
$ sudo python3 -m pip install matplotlib
```

Contact support-compss@bsc.es

Chapter 11

Troubleshooting

This section provides answers for the most common issues of the execution of COMPSs applications and its known limitations.

For specific issues not covered inthis section, please do not hesitate to contact us at: support-compss@bsc.es .

11.1 How to debug

When an error/exception happens during the execution of an application, the first thing that users must do is to check the application output:

- Using `runcompss` the output is shown in the console.
- Using `enqueue_compss` the output is in the `compss-<JOB_ID>.out` and `compss-<JOB_ID>.err`

If the error happens within a task, it will not appear in these files. Users must check the log folder in order to find what has failed. The log folder is by default in:

- Using `runcompss`: `$HOME/.COMPSs/<APP_NAME>_XX` (where XX is a number between 00 and 99, and increases on each run).
- Using `enqueue_compss`: `$HOME/.COMPSs/<JOB_ID>`

This log folder contains the `jobs` folder, where all output/errors of the tasks are stored. In particular, each task produces a `JOB<TASK_NUMBER>_NEW.out` and `JOB<TASK_NUMBER>_NEW.err` files when a task fails.

Tip: If the user enables the **debug mode** by including the `-d` flag into `runcompss` or `enqueue_compss` command, more information will be stored in the log folder of each run easing the error detection. In particular, all output and error output of all tasks will appear within the `jobs` folder.

In addition, some more log files will appear:

- `runtime.log`
- `pycompss.log` (only if using the Python binding).
- `pycompss.err` (only if using the Python binding and an error in the binding happens.)
- `resources.log`
- `workers` folder. This folder will contain four files per worker node:
    - `worker_<MACHINE_NAME>.out`
    - `worker_<MACHINE_NAME>.err`
    - `binding_worker_<MACHINE_NAME>.out`
    - `binding_worker_<MACHINE_NAME>.err`

As a suggestion, users should check the last lines of the `runtime.log`. If the file-transfers or the tasks are failing an error message will appear in this file. If the file-transfers are successfully and the jobs are submitted, users should check the `jobs` folder and look at the error messages produced inside each job. Users should notice that if there are **RESUBMITTED** files something inside the job is failing.

If the `workers` folder is empty, means that the execution failed and the COMPSs runtime was not able to retrieve the workers logs. In this case, users must connect to the workers and look directly into the worker logs. Alternatively, if the user is running with a shared disk (e.g. in a supercomputer), the user can define a shared folder in the `--worker_working_directory=/shared/folder` where a `tmp_XXXXXX` folder will be created on the application execution and all worker logs will be stored.

Tip: When debug is enabled, the workers also produce log files which are transferred to the master when the application finishes. These log files are always removed from the workers (even if there is a failure to avoid abandoning files). Consequently, it is possible **to disable the removal of the log files produced by the workers**, so that users can still check them in the worker nodes if something fails and these logs are not transferred to the master node. To this end, include the following flag into `runcompss` or `enqueue_compss`:

```
--keep_workingdir
```

Please, note that the workers will store the log files into the folder defined by the `--worker_working_directory`, that can be a shared or local folder.

Tip: If segmentation fault occurs, the core dump file can be generated by setting the following flag into `runcompss` or `enqueue_compss`:

```
--gen_coredump
```

The following subsections show debugging examples depending on the choosen flavour (Java, Python or C/C++).

11.1.1 Java examples

11.1.1.1 Exception in the main code

TODO

Missing subsection

11.1.1.2 Exception in a task

TODO

Missing subsection

11.1.2 Python examples

11.1.2.1 Exception in the main code

Consider the following code where an intended error in the main code has been introduced to show how it can be debugged.

```python
from pycompss.api.task import task

@task(returns=1)
```

<div align="right">(continues on next page)</div>

```python
def increment(value):
    return value + 1


def main():
    initial_value = 1
    result = increment(initial_value)

    result = result + 1   # Try to use result without synchronizing it: Error

    print("Result: " + str(result))

if __name__=='__main__':
    main()
```

When executed, it produces the following output:

```
$ runcompss error_in_main.py

[  INFO] Inferred PYTHON language
[  INFO] Using default location for project file: /opt/COMPSs//Runtime/configuration/xml/
→projects/default_project.xml
[  INFO] Using default location for resources file: /opt/COMPSs//Runtime/configuration/xml/
→resources/default_resources.xml
[  INFO] Using default execution type: compss

----------------- Executing error_in_main.py --------------------------

WARNING: COMPSs Properties file is null. Setting default values
[(377)    API]  -  Starting COMPSs Runtime v3.1
[ ERROR ]: An exception occurred: unsupported operand type(s) for +: 'Future' and 'int'
Traceback (most recent call last):
  File "/opt/COMPSs//Bindings/python/2/pycompss/runtime/launch.py", line 204, in compss_main
    execfile(APP_PATH, globals())  # MAIN EXECUTION
  File "error_in_main.py", line 16, in <module>
    main()
  File "error_in_main.py", line 11, in main
    result = result + 1  # Try to use result without synchronizing it: Error
TypeError: unsupported operand type(s) for +: 'Future' and 'int'
[ERRMGR]  -  WARNING: Task 1(Action: 1) with name error_in_main.increment has been cancelled.
[ERRMGR]  -  WARNING: Task canceled: [[Task id: 1], [Status: CANCELED], [Core id: 0],␣
→[Priority: false], [NumNodes: 1], [MustReplicate: false], [MustDistribute: false], [error_
→in_main.increment(INT_T)]]
[(3609)    API]  -  Execution Finished

Error running application
```

It can be identified the complete trackeback pointing where the error is, and the reason. In this example, the reason is `TypeError: unsupported operand type(s) for +: 'Future' and 'int'` since we are trying to use an object that has not been synchronized.

Tip: Any exception raised from the main code will appear in the same way, showing the traceback helping to idenftiy the line which produced the exception and its reason.

11.1.2.2 Exception in a task

Consider the following code where an intended error in a task code has been introduced to show how it can be debugged.

```python
from pycompss.api.task import task
from pycompss.api.api import compss_wait_on

@task(returns=1)
def increment(value):
    return value + 1  # value is an string, can not add an int: Error

def main():
    initial_value = "1"  # the initial value is a string instead of an integer
    result = increment(initial_value)
    result = compss_wait_on(result)
    print("Result: " + str(result))

if __name__=='__main__':
    main()
```

When executed, it produces the following output:

```
$ runcompss error_in_task.py

[  INFO] Inferred PYTHON language
[  INFO] Using default location for project file: /opt/COMPSs//Runtime/configuration/xml/
↪projects/default_project.xml
[  INFO] Using default location for resources file: /opt/COMPSs//Runtime/configuration/xml/
↪resources/default_resources.xml
[  INFO] Using default execution type: compss

----------------- Executing error_in_task.py --------------------------

WARNING: COMPSs Properties file is null. Setting default values
[(570)    API]  -  Starting COMPSs Runtime v3.1
[ERRMGR]   -  WARNING: Job 1 for running task 1 on worker localhost has failed; resubmitting␣
↪task to the same worker.
[ERRMGR]   -  WARNING: Task 1 execution on worker localhost has failed; rescheduling task␣
↪execution. (changing worker)
[ERRMGR]   -  WARNING: Job 2 for running task 1 on worker localhost has failed; resubmitting␣
↪task to the same worker.
[ERRMGR]   -  WARNING: Task 1 has already been rescheduled; notifying task failure.
[ERRMGR]   -  WARNING: Task 'error_in_task.increment' TOTALLY FAILED.
                        Possible causes:
                                -Exception thrown by task 'error_in_task.increment'.
                                -Expected output files not generated by task 'error_in_task.
↪increment'.

                                -Could not provide nor retrieve needed data between master and␣
↪worker.

                        Check files '/home/user/.COMPSs/error_in_task.py_01/jobs/job[1|2]' to␣
↪find out the error.

[ERRMGR]   -  ERROR:   Task failed: [[Task id: 1], [Status: FAILED], [Core id: 0], [Priority:␣
↪false], [NumNodes: 1], [MustReplicate: false], [MustDistribute: false], [error_in_task.
↪increment(STRING_T)]]
[ERRMGR]   -  Shutting down COMPSs...
```

```
[(4711)    API] - Execution Finished
Shutting down the running process

Error running application
```

The output describes that there has been an issue with the task number 1. Since the default behaviour of the runtime is to resubmit the failed task, task 2 also fails.

In this case, the runtime suggests to check the log files of the tasks: `/home/user/.COMPSs/error_in_task.py_-01/jobs/job[1|2]`

Looking into the logs folder, it can be seen that the `jobs` folder contains the logs of the failed tasks:

```
$HOME/.COMPSs
   └── error_in_task.py_01
           ├── jobs
           │   ├── job1_NEW.err
           │   ├── job1_NEW.out
           │   ├── job1_RESUBMITTED.err
           │   ├── job1_RESUBMITTED.out
           │   ├── job2_NEW.err
           │   ├── job2_NEW.out
           │   ├── job2_RESUBMITTED.err
           │   └── job2_RESUBMITTED.out
           ├── resources.log
           ├── runtime.log
           ├── tmpFiles
           └── workers
```

And the `job1_NEW.err` contains the complete traceback of the exception that has been raised (`TypeError: cannot concatenate 'str' and 'int' objects` as consequence of using a string for the task input which tries to add 1):

```
 [EXECUTOR] executeTask - Error in task execution
 es.bsc.compss.types.execution.exceptions.JobExecutionException: Job 1 exit with value 1
     at es.bsc.compss.invokers.external.piped.PipedInvoker.invokeMethod(PipedInvoker.java:78)
     at es.bsc.compss.invokers.Invoker.invoke(Invoker.java:352)
     at es.bsc.compss.invokers.Invoker.processTask(Invoker.java:287)
     at es.bsc.compss.executor.Executor.executeTask(Executor.java:486)
     at es.bsc.compss.executor.Executor.executeTaskWrapper(Executor.java:322)
     at es.bsc.compss.executor.Executor.execute(Executor.java:229)
     at es.bsc.compss.executor.Executor.processRequests(Executor.java:198)
     at es.bsc.compss.executor.Executor.run(Executor.java:153)
     at es.bsc.compss.executor.utils.ExecutionPlatform$2.run(ExecutionPlatform.java:178)
     at java.lang.Thread.run(Thread.java:748)
 Traceback (most recent call last):
 File "/opt/COMPSs/Bindings/python/2/pycompss/worker/commons/worker.py", line 265, in task_
→execution
     **compss_kwargs)
 File "/opt/COMPSs/Bindings/python/2/pycompss/api/task.py", line 267, in task_decorator
     return self.worker_call(*args, **kwargs)
 File "/opt/COMPSs/Bindings/python/2/pycompss/api/task.py", line 1523, in worker_call
     **user_kwargs)
 File "/home/user/temp/Bugs/documentation/error_in_task.py", line 6, in increment
     return value + 1
TypeError: cannot concatenate 'str' and 'int' objects
```

Tip: Any exception raised from the task code will appear in the same way, showing the traceback helping to identify the line which produced the exception and its reason.

11.1.3 C/C++ examples

11.1.3.1 Exception in the main code

TODO

Missing subsection

11.1.3.2 Exception in a task

TODO

Missing subsection

11.2 Common Issues

11.2.1 Tasks are not executed

If the tasks remain in **Blocked** state probably there are no existing resources matching the specific task constraints. This error can be potentially caused by two facts: the resources are not correctly loaded into the runtime, or the task constraints do not match with any resource.

In the first case, users should take a look at the `resouces.log` and check that all the resources defined in the `project.xml` file are available to the runtime. In the second case users should re-define the task constraints taking into account the resources capabilities defined into the `resources.xml` and `project.xml` files.

11.2.2 Jobs fail

If all the application's tasks fail because all the submitted jobs fail, it is probably due to the fact that there is a resource miss-configuration. In most of the cases, the resource that the application is trying to access has no passwordless access through the configured user. This can be checked by:

- Open the `project.xml`. (The default file is stored under /opt/COMPSs/ Runtime/configuration/xml/ projects/project.xml)
- For each resource annotate its name and the value inside the `User` tag. Remember that if there is no `User` tag COMPSs will try to connect this resource with the same username than the one that launches the main application.
- For each annotated resourceName - user please try `ssh user@resourceName`. If the connection asks for a password then there is an error in the configuration of the ssh access in the resource.

The problem can be solved running the following commands:

```
compss@bsc:~$ scp ~/.ssh/id_rsa.pub user@resourceName:./myRSA.pub
compss@bsc:~$ ssh user@resourceName "cat myRSA.pub >> ~/.ssh/authorized_keys; rm ./myRSA.pub"
```

These commands are a quick solution, for further details please check the *Additional Configuration* Section.

11.2.3 Exceptions when starting the Worker processes

When the COMPSs master is not able to communicate with one of the COMPSs workers described in the *project.xml* and *resources.xml* files, different exceptions can be raised and logged on the *runtime.log* of the application. All of them are raised during the worker start up and contain the *[WorkerStarter]* prefix. Next we provide a list with the common exceptions:

InitNodeException Exception raised when the remote SSH process to start the worker has failed.
UnstartedNodeException Exception raised when the worker process has aborted.
Connection refused Exception raised when the master cannot communicate with the worker process (NIO).

All these exceptions encapsulate an error when starting the worker process. This means that **the worker machine is not properly configured** and thus, you need to check the environment of the failing worker. Further information about the specific error can be found on the worker log, available at the working directory path in the remote worker machine (the worker working directory specified in the *project.xml*} file).

Next, we list the most common errors and their solutions:

java command not found Invalid path to the java binary. Check the *JAVA_HOME* definition at the remote worker machine.
Cannot create WD Invalid working directory. Check the rw permissions of the worker's working directory.
No exception The worker process has started normally and there is no exception. In this case the issue is normally due to the firewall configuration preventing the communication between the COMPSs master and worker. Please check that the worker firewall has in and out permissions for TCP and UDP in the adaptor ports (the adaptor ports are specified in the `resources.xml` file. By default the port rank is 43000-44000.

11.2.4 Compilation error: @Method not found

When trying to compile Java applications users can get some of the following compilation errors:

```
error: package es.bsc.compss.types.annotations does not exist
import es.bsc.compss.types.annotations.Constraints;
                                      ^
error: package es.bsc.compss.types.annotations.task does not exist
import es.bsc.compss.types.annotations.task.Method;
                                      ^
error: package es.bsc.compss.types.annotations does not exist
import es.bsc.compss.types.annotations.Parameter;
                                      ^
error: package es.bsc.compss.types.annotations.Parameter does not exist
import es.bsc.compss.types.annotations.parameter.Direction;
                                                ^
error: package es.bsc.compss.types.annotations.Parameter does not exist
import es.bsc.compss.types.annotations.parameter.Type;
                                                ^
error: cannot find symbol
@Parameter(type = Type.FILE, direction = Direction.INOUT)
^
  symbol:   class Parameter
  location: interface APPLICATION_Itf

error: cannot find symbol
@Constraints(computingUnits = "2")
^
  symbol:   class Constraints
  location: interface APPLICATION_Itf

error: cannot find symbol
@Method(declaringClass = "application.ApplicationImpl")
```

```
^
  symbol:   class Method
  location: interface APPLICATION_Itf
```

All these errors are raised because the `compss-engine.jar` is not listed in the CLASSPATH. The default COMPSs installation automatically inserts this package into the CLASSPATH but it may have been overwritten or deleted. Please check that your environment variable CLASSPATH containts the `compss-engine.jar` location by running the following command:

```
$ echo $CLASSPATH | grep compss-engine
```

If the result of the previous command is empty it means that you are missing the `compss-engine.jar` package in your classpath.

The easiest solution is to manually export the CLASSPATH variable into the user session:

```
$ export CLASSPATH=$CLASSPATH:/opt/COMPSs/Runtime/compss-engine.jar
```

However, you will need to remember to export this variable every time you log out and back in again. Consequently, we recommend to add this export to the `.bashrc` file:

```
$ echo "# COMPSs variables for Java compilation" >> ~/.bashrc
$ echo "export CLASSPATH=$CLASSPATH:/opt/COMPSs/Runtime/compss-engine.jar" >> ~/.bashrc
```

> **Warning:** The `compss-engine.jar` is installed inside the COMPSs installation directory. If you have performed a custom installation, the path of the package may be different.

11.2.5 Jobs failed on method reflection

When executing an application the main code gets stuck executing a task. Taking a look at the `runtime.log` users can check that the job associated to the task has failed (and all its resubmissions too). Then, opening the `jobX_NEW.out` or the `jobX_NEW.err` files users find the following error:

```
[ERROR|es.bsc.compss.Worker|Executor] Can not get method by reflection
es.bsc.compss.nio.worker.executors.Executor$JobExecutionException: Can not get method by
→reflection
        at es.bsc.compss.nio.worker.executors.JavaExecutor.executeTask(JavaExecutor.java:142)
        at es.bsc.compss.nio.worker.executors.Executor.execute(Executor.java:42)
        at es.bsc.compss.nio.worker.JobLauncher.executeTask(JobLauncher.java:46)
        at es.bsc.compss.nio.worker.JobLauncher.processRequests(JobLauncher.java:34)
        at es.bsc.compss.util.RequestDispatcher.run(RequestDispatcher.java:46)
        at java.lang.Thread.run(Thread.java:745)
Caused by: java.lang.NoSuchMethodException: simple.Simple.increment(java.lang.String)
        at java.lang.Class.getMethod(Class.java:1678)
        at es.bsc.compss.nio.worker.executors.JavaExecutor.executeTask(JavaExecutor.java:140)
        ... 5 more
```

This error is due to the fact that COMPSs cannot find one of the tasks declared in the Java Interface. Commonly this is triggered by one of the following errors:

- The *declaringClass* of the tasks in the Java Interface has not been correctly defined.
- The parameters of the tasks in the Java Interface do not match the task call.
- The tasks have not been defined as *public*.

11.2.6 Jobs failed on reflect target invocation null pointer

When executing an application the main code gets stuck executing a task. Taking a look at the `runtime.log` users can check that the job associated to the task has failed (and all its resubmissions too). Then, opening the `jobX_NEW.out` or the `jobX_NEW.err` files users find the following error:

```
[ERROR|es.bsc.compss.Worker|Executor]
java.lang.reflect.InvocationTargetException
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:
→43)
        at java.lang.reflect.Method.invoke(Method.java:606)
        at es.bsc.compss.nio.worker.executors.JavaExecutor.executeTask(JavaExecutor.java:154)
        at es.bsc.compss.nio.worker.executors.Executor.execute(Executor.java:42)
        at es.bsc.compss.nio.worker.JobLauncher.executeTask(JobLauncher.java:46)
        at es.bsc.compss.nio.worker.JobLauncher.processRequests(JobLauncher.java:34)
        at es.bsc.compss.util.RequestDispatcher.run(RequestDispatcher.java:46)
        at java.lang.Thread.run(Thread.java:745)
Caused by: java.lang.NullPointerException
        at simple.Ll.printY(Ll.java:25)
        at simple.Simple.task(Simple.java:72)
        ... 10 more
```

This cause of this error is that the Java object accessed by the task has not been correctly transferred and one or more of its fields is null. The transfer failure is normally caused because the transferred object is not serializable.

Users should check that all the object parameters in the task are either implementing the serializable interface or following the *java beans* model (by implementing an empty constructor and getters and setters for each attribute).

11.2.7 Tracing merge failed: too many open files

When too many nodes and threads are instrumented, the tracing merge can fail due to an OS limitation, namely: the maximum open files. This problem usually happens when using advanced mode due to the larger number of threads instrumented. To overcome this issue users have two choices. **First option**, use *Extrae* parallel MPI merger. This merger is automatically used if COMPSs was installed with MPI support. In Ubuntu you can install the following packets to get MPI support:

```
$ sudo apt-get install libcr-dev mpich2 mpich2-doc
```

Please note that extrae is never compiled with MPI support when building it locally (with buildlocal command).

To check if COMPSs was deployed with MPI support, you can check the installation log and look for the following *Extrae* configuration output:

```
Package configuration for Extrae VERSION based on extrae/trunk rev. 3966:
-----------------------
Installation prefix: /gpfs/apps/MN3/COMPSs/Trunk/Dependencies/extrae
Cross compilation: no
CC: gcc
CXX: g++
Binary type: 64 bits


MPI instrumentation: yes
    MPI home: /apps/OPENMPI/1.8.1-mellanox
    MPI launcher: /apps/OPENMPI/1.8.1-mellanox/bin/mpirun
```

On the other hand, if you already installed COMPSs, you can check *Extrae* configuration executing the script `/opt/COMPSs/Dependencies/extrae/etc/configured.sh`. Users should check that flags `--with-mpi=/usr` and

--enable-parallel-merge are present and that MPI path is correct and exists. Sample output:

```
EXTRAE_HOME is not set. Guessing from the script invoked that Extrae was installed in /opt/
↪COMPSs/Dependencies/extrae
The directory exists .. OK
Loaded specs for Extrae from /opt/COMPSs/Dependencies/extrae/etc/extrae-vars.sh

Extrae SVN branch extrae/trunk at revision 3966

Extrae was configured with:
$ ./configure --enable-gettimeofday-clock --without-mpi --without-unwind --without-dyninst --
↪without-binutils --with-mpi=/usr --enable-parallel-merge --with-papi=/usr --with-java-jdk=/
↪usr/lib/jvm/java-7-openjdk-amd64/ --disable-openmp --disable-nanos --disable-smpss --
↪prefix=/opt/COMPSs/Dependencies/extrae --with-mpi=/usr --enable-parallel-merge --libdir=/
↪opt/COMPSs/Dependencies/extrae/lib

CC was gcc
CFLAGS was -g -O2 -fno-optimize-sibling-calls -Wall -W
CXX was g++
CXXFLAGS was -g -O2 -fno-optimize-sibling-calls -Wall -W

MPI_HOME points to /usr and the directory exists .. OK
LIBXML2_HOME points to /usr and the directory exists .. OK
PAPI_HOME points to /usr and the directory exists .. OK
DYNINST support seems to be disabled
UNWINDing support seems to be disabled (or not needed)
Translating addresses into source code references seems to be disabled (or not needed)

Please, report bugs to tools@bsc.es
```

Important: Disclaimer: the parallel merge with MPI will not bypass the system's maximum number of open files, just distribute the files among the resources. If all resources belong to the same machine, the merge will fail anyways.

The **second option** is to increase the OS maximum number of open files. For instance, in Ubuntu add `` ulimit -n 40000 `` just before the start-stop-daemon line in the do_start section.

11.2.8 Performance issues

11.2.8.1 Different work directories

Having different work directories (for master and workers) may lead to performance issues. In particular, if the work directories belong to different mount points and with different performance, where the copy of files may be required. For example, using folders that are shared across nodes in a supercomputer but with different performance (e.g. scratch and projects in MareNostrum 4) for the master and worker workspaces.

11.3 Memory Profiling

COMPSs also provides a mechanism to show the memory usage over time when running Python applications. This is particularly useful when memory issues happen (e.g. memory exhausted – causing the application crash), or performance analysis (e.g. problem size scalability).

To this end, the `runcompss` and `enqueue_compss` commands provide the `--python_memory_profile` flag, which provides a set of files (one per node used in the application execution) where the memory used during the execution is recorded at the end of the application. They are generated in the same folder where the execution has been launched.

Important: The `memory-profiler` and `psutil` packages are mandatory in order to use the `--python_memory_-profile` flag.

It can be easily installed with pip:

```
$ python -m pip install psutil memory-profiler --user
```

Tip: If you want to store from the memory profiler in a different folder, export the `COMPSS_WORKER_PROFILE_PATH` with the destination path:

```
$ export COMPSS_WORKER_PROFILE_PATH=/path/to/destination
```

When `--python_memory_profile` is included, a file with name `mprofile_<DATE_TIME>.dat` is generated for the master memory profiling, while for the workers they are named `<WORKER_NODE_NAME>.dat`. These files can be displayed with the `mprof` tool:
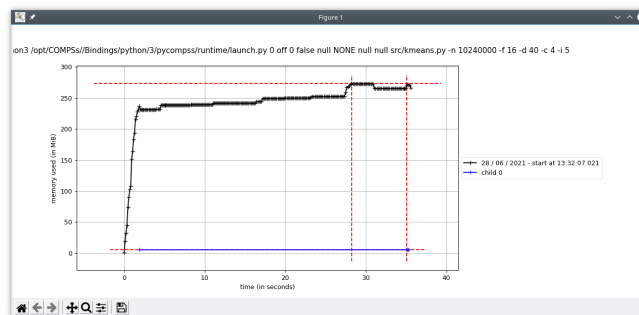
```
$ mprof plot <FILE>.dat
```


Figure 54: mprof plot example

11.3.1 Advanced profiling

For a more fine grained memory profiling and analysing the **workers** memory usage, PyCOMPSs provides the `@profile` decorator. This decorator is able to display the memory usage per line of the code. It can be imported from the PyCOMPSs functions module:

```
from pycompss.functions.profile import profile
```

This decorator can be placed over any function:

Over the `@task` decorator (or over the decorator stack of a task) This will display the memory usage in the master (through standard output).

Under the @task decorator: This will display the memory used by the actual task in the worker. The memory usage will be shown through standard output, so it is mandatory to enable debug (`--log_level=debug`) and check the job output file from `.COMPSs/<app_folder>/jobs/`.

Over a non task function: Will display the memory usage of the function in the master (through standard output).

By default, the `@profile` decorator reports the memory usage line by line:

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
     7     53.3 MiB     53.3 MiB           1   @task(returns=1)
     8                                          @profile()
     9                                          def increment(value):
    10     61.0 MiB      7.7 MiB           1       a = [1] * (10 ** 6)
    11     83.7 MiB     22.7 MiB           1       b = [2] * (value * 10 ** 6)
    12    312.6 MiB    228.9 MiB           1       c = [3] * (value * 10 ** 7)
    13    289.9 MiB    -22.7 MiB           1       del b
    14    289.9 MiB      0.0 MiB           1       return value + 1
Job name: job10_NEW
Task start time: 1653572135.1119144
Elapsed time: 0.10722756385803223
Initial memory: 8150122496
Final memory: 7759843328
```

But this information can be reduce to show only the peak memory usage of each task by setting `full_report=False` in the `@profile` decorator (`@profile(full_report=False)`). More specifically, the profiling information reported will be a one-liner per task showing:

1. The task start time
2. The task job name
3. The file that contains the task
4. The task name
5. The task elapsed time
6. The amount of memory used before executing the task
7. The amount of memory used after executing the task
8. The peak memory usage

```
1653572135.1119144 job10_NEW /path/to/increment.py increment 0.10722756385803223 8150122496␣
→7759843328 312.6 MiB
```

Tip: It is possible to redirect the profiling output to a single file by exporting the `COMPSS_PROFILING_FILE` environment variable with the path to the destination file.

Please, remind that this variable needs to be available in the worker if the `@profile` decorator is used to report the memory usage of the tasks. Consequently, consider the usage of the `--env_script` flag in the `runcompss` command defining a script that exports the `COMPSS_PROFILING_FILE` in order to make it available in the workers in local executions.

11.4 Known Limitations

The current COMPSs version has the following limitations:

11.4.1 Global

Exceptions The current COMPSs version is not able to propagate exceptions raised from a task to the master. However, the runtime catches any exception and sets the task as failed.

Use of file paths The persistent workers implementation has a unique *Working Directory* per worker. That means that tasks should not use hardcoded file names to avoid file collisions and tasks misbehaviours. We recommend to use files declared as task parameters, or to manually create a sandbox inside each task execution and/or to generate temporary random file names.

11.4.2 With Java Applications

Java tasks Java tasks **must** be declared as **public**. Despite the fact that tasks can be defined in the main class or in other ones, we recommend to define the tasks in a separated class from the main method to force its public declaration.

Java objects Objects used by tasks must follow the *java beans* model (implementing an empty constructor and getters and setters for each attribute) or implement the *serializable* interface. This is due to the fact that objects will be transferred to remote machines to execute the tasks.

Java object aliasing If a task has an object parameter and returns an object, the returned value must be a new object (or a cloned one) to prevent any aliasing with the task parameters.

```java
// @Method(declaringClass = "...")
// DummyObject incorrectTask (
//     @Parameter(type = Type.OBJECT, direction = Direction.IN) DummyObject a,
//     @Parameter(type = Type.OBJECT, direction = Direction.IN) DummyObject b
// );
public DummyObject incorrectTask (DummyObject a, DummyObject b) {
    if (a.getValue() > b.getValue()) {
        return a;
    }
    return b;
}

// @Method(declaringClass = "...")
// DummyObject correctTask (
//     @Parameter(type = Type.OBJECT, direction = Direction.IN) DummyObject a,
//     @Parameter(type = Type.OBJECT, direction = Direction.IN) DummyObject b
// );
public DummyObject correctTask (DummyObject a, DummyObject b) {
    if (a.getValue() > b.getValue()) {
        return a.clone();
    }
    return b.clone();
}

public static void main() {
    DummyObject a1 = new DummyObject();
    DummyObject b1 = new DummyObject();
    DummyObject c1 = new DummyObject();
    c1 = incorrectTask(a1, b1);
    System.out.println("Initial value: " + c1.getValue());
    a1.modify();
```

(continues on next page)

```java
        b1.modify();
        System.out.println("Aliased value: " + c1.getValue());


        DummyObject a2 = new DummyObject();
        DummyObject b2 = new DummyObject();
        DummyObject c2 = new DummyObject();
        c2 = incorrectTask(a2, b2);
        System.out.println("Initial value: " + c2.getValue());
        a2.modify();
        b2.modify();
        System.out.println("Non-aliased value: " + c2.getValue());
}
```

11.4.3 With Python Applications

Python constraints in the cloud When using python applications with constraints in the cloud the minimum number of VMs must be set to 0 because the initial VM creation does not respect the tasks constraints. Notice that if no constraints are defined the initial VMs are still usable.

Intermediate files Some applications may generate intermediate files that are only used among tasks and are never needed inside the master's code. However, COMPSs will transfer back these files to the master node at the end of the execution. Currently, the only way to avoid transferring these intermediate files is to manually erase them at the end of the master's code. Users must take into account that this only applies for files declared as task parameters and **not** for files created and/or erased inside a task.

User defined classes in Python User defined classes in Python **must not be** declared **in the same file that contains the main method** (if `__name__=='__main__'`) to avoid serialization problems of the objects.

Python object hierarchy dependency detection Dependencies are detected only on the objects that are task parameters or outputs. Consider the following code:

```python
# a.py
class A:
  def __init__(self, b):
    self.b  = b

# main.py
from a import A
from pycompss.api.task import task
from pycompss.api.parameter import *
from pycompss.api.api import compss_wait_on

@task(obj = IN, returns = int)
def get_b(obj):
  return obj.b

@task(obj = INOUT)
def inc(obj):
  obj += [1]

def main():
  my_a = A([5])
  inc(my_a.b)
  obj = get_b(my_a)
  obj = compss_wait_on(obj)
  print obj
```

```
if __name__ == '__main__':
    main()
```

Note that there should exist a dependency between `A` and `A.b`. However, PyCOMPSs is not capable to detect dependencies of that kind. These dependencies must be handled (and avoided) manually.

Python modules with global states Some modules (for example `logging`) have internal variables apart from functions. These modules are not guaranteed to work in PyCOMPSs due to the fact that master and worker code are executed in different interpreters. For instance, if a `logging` configuration is set on some worker, it will not be visible from the master interpreter instance.

Python global variables This issue is very similar to the previous one. PyCOMPSs does not guarantee that applications that create or modify global variables while worker code is executed will work. In particular, this issue (and the previous one) is due to Python's Global Interpreter Lock (GIL).

Python application directory as a module If the Python application root folder is a python module (i.e: it contains an `__init__.py` file) then `runcompss` must be called from the parent folder. For example, if the Python application is in a folder with an `__init__.py` file named `my_folder` then PyCOMPSs will resolve all functions, classes and variables as `my_folder.object_name` instead of `object_name`. For example, consider the following file tree:

```
my_apps/
└── kmeans/
        ├── __init__.py
        └── kmeans.py
```

Then the correct command to call this app is `runcompss kmeans/kmeans.py` from the `my_apps` directory.

Python early program exit All intentional, premature exit operations must be done with `sys.exit`. PyCOMPSs needs to perform some cleanup tasks before exiting and, if an early exit is performed with `sys.exit`, the event will be captured, allowing PyCOMPSs to perform these tasks. If the exit operation is done in a different way then there is no guarantee that the application will end properly.

Python with numpy and MKL Tasks that invoke numpy and MKL may experience issues if tasks use a different number of MKL threads. This is due to the fact that MKL reuses threads along different calls and it does not change the number of threads from one call to another.

11.4.4 With Services

Services types The current COMPSs version only supports SOAP based services that implement the WS interoperability standard. REST services are not supported.