COMPSs Documentation

Release 2.6

Workflows and Distributed Computing Group (WDC)

Nov 11, 2020

Table of Contents


COMPSs
14
23
12
18
11
16
19

CHAPTER 1

What is COMPSs?

COMP Superscalar (COMPSs) is a programming model which aims to ease the development of applications for distributed infrastructures, such as Clusters, Grids and Clouds. COMP Superscalar also features a runtime system that exploits the inherent parallelism of applications at execution time.

For the sake of programming productivity, the COMPSs model has four key 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.
- **Infrastructure unaware:** 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.
- **Standard programming languages:** COMPSs is based on the popular programming language Java, but also offers language bindings for Python and C/C++ applications. This facilitates the learning of 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.

CHAPTER 2

Installation and Administration

2.1 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 (>= 2.7 | >=3.6), python-dev | python3-dev, python-setuptools|python3-setuptools, libpython2.7
COMPSs C/C++ Binding	libtool, automake, build-essential, libboost-all-dev, libxml2-dev
COMPSs Autoparallel	libgmp3-dev, flex, bison, libbison-dev, texinfo, libffi-dev, astor, sympy, enum34, islpy
COMPSs Tracing	libxml2 (>= 2.5), libxml2-dev (>= 2.5), gfortran, papi

2.1.1 Build Dependencies

To build COMPSs from sources you will also need `wget`, `openjdk-8-jdk` and `maven`.

2.1.2 Optional Dependencies

For the Python binding it is also recommended to have `dill` and `guppy` installed. The `dill` package increases the variety of serializable objects by Python (for example: lambda functions), and the `guppy` package is needed to use the `@local` decorator. Both packages can be found in pyPI and can be installed via `pip`.

2.2 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 --single-branch --branch=2.6 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
$ compss> ./submodules_patch.sh
```

Finally you just need to run the installation script. You have to options: For installing COMPSs for all the users run the following command. (root access is required)

```
$ compssk> cd builders/
$ builders> INSTALL_DIR=/opt/COMPSs/
$ builders> sudo -E ./buildlocal [options] ${INSTALL_DIR}
```

For installing COMPSs for the current user run the following command.

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

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

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

2.2.1 Post installation

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

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.

2.3 Pip

2.3.1 Pre-requisites

In order to be able to install COMPSs and PyCOMPSs with Pip the following requirements must be met:

1. Have all the dependencies (excluding the COMPSs packages) mentioned in the *Dependencies* Section satisfied and Python `pip`. As an example for some distributions:

   Fedora 25 dependencies installation command:

   ```
   $ 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
   ```

   Ubuntu 16.04 dependencies installation command:

   ```
   $ sudo apt-get install -y openjdk-8-jdk graphviz xdg-utils libtool automake build-
   ↪essential python2.7 libpython2.7 libboost-serialization-dev libboost-iostreams-
   ↪dev  libxml2 libxml2-dev csh gfortran python-pip libpapi-dev
   ```

Ubuntu 18.04 dependencies installation command:

```
$ sudo apt-get install -y openjdk-8-jdk graphviz xdg-utils libtool automake build-
↪essential python2.7 libpython2.7 python3 python3-dev libboost-serialization-dev␣
↪libboost-iostreams-dev  libxml2 libxml2-dev csh gfortran libgmp3-dev flex bison␣
↪texinfo python3-pip libpapi-dev
```

OpenSuse 42.2 dependencies installation command:

```
$ 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
```

Debian 8 dependencies installation command:

```
$ 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
```

CentOS 7 dependencies installation command:

```
$ 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
```

2. 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/
```

2.3.2 Installation

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

1. Install systemwide:

```
$ sudo -E 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 /etc/profile.d/compss.sh
```

However, this command should be executed in every different terminal during the current user session.

2. 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
```

3. Within a Python virtual environment:

```
$ 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.

2.3.3 Configuration (using pip)

The steps mentioned in Section *Configure SSH passwordless* must be done in order to have a functional COMPSs and PyCOMPSs installation.

2.3.4 Post installation (using pip)

As mentioned in *Configure SSH passwordless* Section, it is recommended to restart the user session or virtual environment once the installation process has finished.

2.4 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.

2.4.1 SC 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)

2.4.2 SC 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 <targetDir> [<supercomputer.cfg>]

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

The installation script will create a COMPSs folder inside the given `<targetDir>` so the final COMPSs installation will be placed under the `<targetDir>/COMPSs` folder.

Attention: If the `<targetDir>/COMPSs` 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 recomend to source the `<targetDir>/COMPSs/compssenv` into the users *.bashrc*.

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

2.4.3 SC Configuration

For queue system executions, COMPSs has a pre-build structure (see Figure 1) to execute applications in SuperComputers. For this purpose, users must use the *enqueue_compss* script provided in the COMPSs installation. This script has several parameters (see *enqueue_compss -h*) that allow users to customize their executions for any SuperComputer.

To make this structure works, the administrators must define a configuration file for the queue system (under `<targetDir>/COMPSs/scripts/queues/queue_system/QUEUE.cfg`) and a configuration file for the specific SuperComputer parameters (under `<targetDir> /COMPSs/scripts/queues/supercomputers/SC_NAME.cfg`). The COMPSs installation already provides queue configurations for *LSF* and *SLURM* and several examples for SuperComputer configurations.

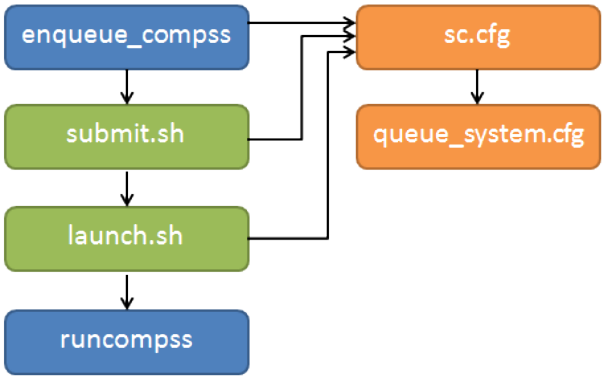


Figure 1: Structure of COMPSs queue scripts. In Blue user scripts, in Green queue scripts and in Orange system dependant scripts

To create a new configuration we recommend to use one of the configurations provided by COMPSs (such as the configuration for the *MareNostrum IV* SuperComputer) or to contact us at support-compss@bsc.es.

2.4.4 SC 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: 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.
```

(continues on next page)

(continued from previous page)

```
                                        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
                                        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
  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
```

(continues on next page)

(continued from previous page)

```
    --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
    --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
    --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: scratch | gpfs |
↪<path>
                                            Default: scratch

    --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
    --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
```

(continues on next page)

(continued from previous page)

```
                                          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

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

  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␣
↪true
                                          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

  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/2.6.pr/Runtime/
↪configuration/xml/projects/default_project.xml
    --resources=<path>                    Path to the resources XML file
                                          Default: /apps/COMPSs/2.6.pr/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
```

(continues on next page)

(continued from previous page)

```
                                        Default: false
    --log_level=<level>, --debug, -d    Set the debug level: off | info | debug
                                        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
    --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: null
    --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.fullGraphScheduler.FullGraphScheduler
                                                            | es.bsc.compss.
↪scheduler.fifoScheduler.FIFOScheduler
                                                            | es.bsc.compss.
↪scheduler.resourceEmptyScheduler.ResourceEmptyScheduler
                                        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/bsc19/bsc19234
    --pythonpath=<path>                 Additional folders or paths to add to the␣
↪PYTHONPATH
```

(continues on next page)

(continued from previous page)

```
                                          Default: /home/bsc19/bsc19234
    --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:
    --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.
```

(continues on next page)

(continued from previous page)

```
                                          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

* 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://compss.bsc.es/projects/bar. 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.

2.5 Additional Configuration

2.5.1 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 dsa
```

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

```
$ # For every other available machine (MACHINE):
$ scp ~/.ssh/id_dsa.pub MACHINE:./myDSA.pub
$ ssh MACHINE "cat ./myDSA.pub >> ~/.ssh/authorized_keys; rm ./myDSA.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 2, users will have to execute the following commands to grant free ssh access between any pair of machines:

```
me@localhost:~$ ssh-keygen -t id_dsa
# Granting access localhost -> m1.bsc.es
me@localhost:~$ scp ~/.ssh/id_dsa.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_dsa.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_dsa
user_m1@m1.bsc.es:~> exit
# Granting access m1.bsc.es -> localhost
me@localhost:~$ scp user_m1@m1.bsc.es:~/.ssh/id_dsa.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_dsa
user_m2@m2.bsc.es:~> exit
# Granting access m2.bsc.es -> localhost
me@localhost:~$ scp user_m2@m1.bsc.es:~/.ssh/id_dsa.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
```

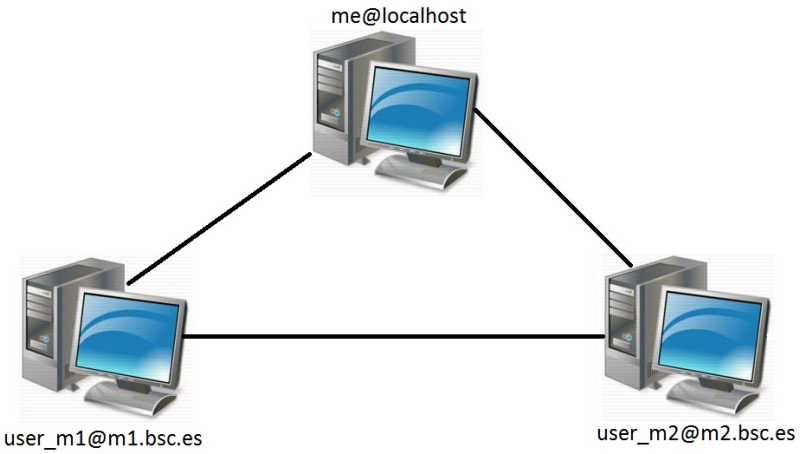


Figure 2: Cluster example

2.5.2 Configure the COMPSs Cloud Connectors

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

OCCI (Open Cloud Computing Interface) connector

In order to execute a COMPSs application using cloud resources, the rOCCI (Ruby OCCI) connector 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

2.6 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.

2.6.1 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">
```

(continues on next page)

(continued from previous page)

```
                <SubmissionSystem>
                    <Interactive/>
                </SubmissionSystem>
                <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>
```

2.6.2 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 than 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-->
```

(continues on next page)

(continued from previous page)

```
    <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>
```

2.6.3 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.

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>
```

(continues on next page)

(continued from previous page)

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

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 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>
```

(continues on next page)

(continued from previous page)

```
    </ComputeNode>

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

```
<?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>
```

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.

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 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>
```

(continues on next page)

(continued from previous page)

```
                    </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>
                <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>
```

(continues on next page)

(continued from previous page)

```
                </Image>
            </Images>
            <InstanceTypes>
                <InstanceType Name="Instance1"/>
                <InstanceType Name="Instance2"/>
            </InstanceTypes>
        </CloudProvider>

        <CloudProvider Name="PROVIDER_NAME2">
            ...
        </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 4 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 *JClouds extensions in the <project>.xml file* 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.

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 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>
        <Port>HmmerObjectsPort</Port>
```

(continues on next page)

(continued from previous page)

```
    </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 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>
```

CHAPTER 3

Application development

3.1 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.

3.1.1 Programming Model

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.
- **Java annotated interface:** It 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 app (Code 1 **Simple.java**) 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)

```
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 (Code 2 **SimpleImpl.java**) that will be executed remotely by COMPSs.

Code 2: Simple Implementation (SimpleImpl.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();
    }
  }
}
```

Java annotated interface

The 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 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.

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

- **Method-level Metadata:** for each selected method, the following metadata has to be defined:
  - **@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 successor 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.
    - **computingNodes** String defining the number of computing nodes reserved for the MPI execution (only a single node is reserved by default).
    - **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).
  - **@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.
  - **@SchedulerHints:** It specifies the class that implements the method.
    - **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).

- **Parameter-level Metadata (@Parameter):** for each parameter and method, the user must define:
  - **Direction:** *Direction.IN, Direction.INOUT, Direction.OUT or Direction.CONCURRENT*
  - **Type:** COMPSs supports the following types for task parameters:
    - **Basic 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:** *Type.STRING*. It can only have **IN** direction, since Java Strings are immutable.
    - **File:** *Type.FILE*. It can have any direction (IN, OUT, INOUT or CONCURRENT). 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 a FILE, COMPSs will treat it as such.
    - **Object:** *Type.Object*. It can have any direction (IN, OUT, INOUT or CONCURRENT).
  - **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 StdIOStream 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.

- **Service-level Metadata:** for each selected service, the following metadata has to be defined:
  - **@Service:** Mandatory. It specifies the service properties.
    * **namespace** Mandatory. Service namespace
    * **name** Mandatory. Service name.
    * **port** Mandatory. Service port.
    * **operation** Operation type.
    * **priority** "true" if the service takes priority, "false" otherwise. This parameter is used by the COMPSs scheduler (it is a String not a Java boolean).

The Java annotated interface of the Simple app example (Code 3 SimpleItf.java) 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)

```
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
  );

}
```

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 4 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 4: Alternative sorting method definition example

```
@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 5) only one core is required to run the method of both sorting algorithms.

Code 5: Alternative sorting method definition with constraint example

```
@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 6 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 6: Alternative sorting method definition with specific constraints example

```
@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
);
```

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 7).

Code 7: 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 if 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 8 shows a usage example of this API call.

Code 8: 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 9 contains an example of its usage.

Code 9: 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);
    }
}
```

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

An Integrated Development Environment for Eclipse is also available to simplify the development, compilation, deployment and execution COMPSs applications. For further information about the *COMPSs IDE* please refer to the *COMPSs IDE User Guide* available at http://compss.bsc.es .

3.1.3 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 *Application execution* 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.

The next sections describe the Python and C/C++ language bindings offered by COMPSs.

3.2 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.

3.2.1 Programming Model

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 10), let us assume that the application calls a function *func*, which receives a file path (string parameter) and an integer parameter. The code of *func* updates the file.

Code 10: Python application example

```
def func(file_path, value):
    # update the file 'file_path'

if __name__=='__main__':
    my_file = '/tmp/sample_file.txt'
    func(my_file, 1)
```

Hint: the main code is defined within *if __name__==__main__'*. A better alternative would be to define the main code within a function and invoke it from the *if __name__=='__main__'*.

In order to select *func* 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 11).

Code 11: Python task import

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

The metadata corresponding to a parameter is specified as an argument of the 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) and *files* are supported.
- Direction: it can be read-only (*IN* - default), read-write (*INOUT*), write-only (*OUT*) or in some cases concurrent (*CONCURRENT*).

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* and *OUT* parameters. Thus, when defining the parameter metadata in the *@task* decorator, the user has the following options:

- *IN*: the parameter is read-only. The type will be inferred.
- *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 acces. The type will be inferred.

- *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.
- *COLLECTION_IN*: the parameter is read-only collection.
- *COLLECTION_INOUT*: the parameter is read-write collection.

Consequently, 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 12)

Code 12: Python task parameters import

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

Continuing with the example, in Code 13 the decorator specifies that *func* has a parameter called *f*, of type *FILE* 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.

Code 13: Python task example

```
from pycompss.api.task import task      # Import @task decorator
from pycompss.api.parameter import *    # Import parameter metadata for the @task␣
↪decorator

@task(f=FILE_INOUT)
def func(f, i):
    fd = open(f, 'r+')
    ...
```

The user can also define that the access to a parameter is concurrent with *CONCURRENT* or to a file *FILE_CONCURRENT* (Code 14). 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. However, note that COMPSs does not manage the interaction with the objects or files used/modified concurrently. Taking care of the access/modification of the concurrent objects is responsibility of the developer.

Code 14: Python task example with CONCURRENT

```
from pycompss.api.task import task      # Import @task decorator
from pycompss.api.parameter import *    # Import parameter metadata for the @task␣
↪decorator

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

Moreover, it is possible to specify that a parameter is a collection of elements (e.g. list) and its direction (COLLECTION_IN or COLLECTION_INOUT) (Code 15). In this case, 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 15: Python task example with COLLECTION

```
from pycompss.api.task import task     # Import @task decorator
from pycompss.api.parameter import *   # Import parameter metadata for the @task␣
↪decorator

@task(my_collection=COLLECTION_IN)
def func(my_collection):
    for element in my_collection:
        ...
```

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 16)

Code 16: Python task example with COLLECTION and depth

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

If the function or method returns a value, the programmer must use the *returns* argument within the *@task* decorator. In this argument, the programmer can specify the type of that value (Code 17).

Code 17: Python task returns example

```
@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 18 shows how to specify that two values (an integer and a list) are returned.

Code 18: Python task with multireturn example

```
@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 19). 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 19: Python task returns with integer example

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

(continues on next page)

(continued from previous page)

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

The use of **args* and ***kwargs* as function parameters is also supported (Code 20).

Code 20: Python task **args* and ***kwargs example*

```
@task(returns=int)
def argkwarg_func(*args, **kwargs):
    return sum(args) + len(kwargs)
```

And even with other parameters, such as usual parameters and *default defined arguments*. Code 21 shows an example of a task with two three parameters (whose one of them (*'s'*) has a default value), **args* and ***kwargs*.

Code 21: Python task with default parameters example

```
@task(returns=int)
def multiarguments_func(v, w, s = 2, *args, **kwargs):
    return (v * w) + sum(args) + len(kwargs) + s
```

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 22).

Code 22: Python instance method example

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

Caution: In order to avoid serialization issues, the classes must not be declared in the same file that contains the main method (`if __name__=='__main__'`).

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 23). 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 23: Python task priority example

```
@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 24). 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 24: Python task is_distributed and is_replicated examples

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

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

In case a task fails, the whole application behaviour can be defined using the *on_failure* argument (Code 25). It has four possible values: **'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 25: Python task on_failure example

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

Table 8 summarizes the 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 tipe that will be inferred. • IN: Read-only parameter, all types. • INOUT: Read-write parameter, all types except file (primitives, strings, objects). • OUT: Write-only parameter, all types except file (primitives, strings, objects). • CONCURRENT: Concurrent read-write parameter, all types except file (primitives, strings, objects). • FILE/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. • COLLECTION_IN: Read-only collection parameter (list). • COLLECTION_INOUT: Read-write collection parameter (list). • Dictionary: {Type:(empty=object)/FILE/COLLECTION, Direction:(empty=IN)/IN/INOUT/OUT/CONCURRENT}
returns	int (for integer and boolean), long, float, str, dict, list, tuple, user-defined classes
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'

Other task types

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

For instance, there is a set of decorators that can be placed over the *@task* decorator in order to define the task methods as a **binary invocation** (with the *Binary decorator*), as a **OmpSs invocation** (with the *OmpSs decorator*), as a **MPI invocation** (with the *MPI decorator*), as a **COMPSs application** (with the *COMPSs decorator*), or as a **task that requires multiple nodes** (with the *Multinode decorator*). These decorators must be placed over the *@task* decorator, and under the *@constraint* decorator if defined.

Consequently, the task body will be empty and the function parameters will be used as invocation parameters with some extra information that can be provided within the *@task* decorator.

The following subparagraphs describe their usage.

Binary decorator

The *@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 26 shows the most simple binary task definition without/with constraints (without parameters):

Code 26: Binary task example

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

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

@constraint(computingUnits="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.
```

Code 27 shows a more complex binary invocation, with files as parameters:

Code 27: Binary task example 2

```
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 folloowing, 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.

Thus, PyCOMPSs can also deal with prefixes for the given parameters. Code 28 performs a system call (ls) with specific prefixes:

Code 28: Binary task example 3

```
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 conjuntion 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.

OmpSs decorator

The *@ompss* decorator shall be used to define that a task is going to invoke a OmpSs executable (Code 29).

Code 29: OmpSs task example

```
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* decorator shall be used to define that a task is going to invoke a MPI executable (Code 30).

Code 30: MPI task example

```
from pycompss.api.mpi import mpi
```

(continues on next page)

(continued from previous page)

```
@mpi(binary="mpiApp.bin", runner="mpirun", computing_nodes=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.

COMPSs decorator

The *@compss* decorator shall be used to define that a task is going to be a COMPSs application (Code 31). It enables to have nested PyCOMPSs/COMPSs applications.

Code 31: 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* decorator shall be used to define that a task is going to use multiple nodes (e.g. using internal parallelism) (Code 32).

Code 32: 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.

Parameters summary

Next tables summarizes the parameters of these decorators.

- **@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.

- **@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**

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.
runner	(Mandatory) String defining the MPI runner command.
computing_nodes	Integer defining the number of computing nodes reserved for the MPI execution (only a single node is reserved by default).

- **@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**

Parameter	Description
computing_nodes	Integer defining the number of computing nodes reserved for the task execution (only a single node is reserved by default).

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)	Type: COLLECTION, Direction: INOUT
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

These parameter keys, as well as the shortcuts, can be imported from the PyCOMPSs library:

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

Constraints

As in Java COMPSs applications, it is possible to define constraints for each task. To this end, the decorator *@constraint* followed by the desired constraints needs to be placed over the @task decorator (Code 33).

Code 33: 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):
    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 34). A full description of the supported constraints can be found in Table 14.

For example:

Code 34: Constrained task with environment variable example

```
from pycompss.api.task import task
from pycompss.api.constraint import constraint
```

(continues on next page)

(continued from previous page)

```
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 35).

Code 35: 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.

Implements

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* decorator followed with the specific implementations constraints (with the *@constraint* decorator, see Section [subsubsec:constraints]) needs to be placed over 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 36).

Code 36: Multiple task implementations example

```
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):
```

(continues on next page)

(continued from previous page)

```
    # 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 or multinode 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.

Main Program

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 four API functions that can to 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_delete_file(file_name) Notifies the runtime to delete a file.

compss_wait_on_file(file_name) Synchronizes for the last version of the file *file_name*. Returns True if success (False otherwise).

compss_delete_object(object) Notifies the runtime to delete all the associated files to a given object.

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_wait_on(obj, to_write=True) Synchronizes for the last version of object *obj* and returns the synchronized object. It can have an optional boolean parameter *to_write*, which defaults to *True*, 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.

To illustrate the use of the aforementioned API functions, the following example (Code 37) 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 barrier performs a synchronization, and the execution of the main user code will not continue until the ten *func* tasks have finished.

Code 37: PyCOMPSs API usage

```
from pycompss.api.api import compss_open
from pycompss.api.api import compss_delete_file
from pycompss.api.api import compss_wait_on
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'
```

(continues on next page)

(continued from previous page)

```
    func(my_file2)
    fd = compss_delete_file(my_file2)
    ...

    my_obj = MyClass()
    my_obj.method()
    my_obj = compss_wait_on(my_obj)
    ...

    for i in range(10):
        func(str(i) + my_file)
    compss_barrier()
    ...
```

The corresponding task selection for the example above would be (Code 38):

Code 38: PyCOMPSs API usage tasks

```
@task(f=FILE_OUT)
def func(f):
    ...

class MyClass(object):
    ...

    @task()
    def method(self):
        ... # self is modified here
```

Table 11 summarizes the API functions to be used in the main program of a COMPSs Python application.

Table 11: COMPSs Python API functions

API Function	Description
compss_open(file_name, mode='r')	Synchronizes for the last version of a file and returns its file descriptor.
compss_delete_file(file_name)	Notifies the runtime to remove a file.
compss_wait_on_file(file_name)	Synchronizes for the last version of a file.
compss_delete_object(object)	Notifies the runtime to delete the associated file to this object.
compss_barrier(no_more_tasks=False)	Wait for all tasks submitted before the barrier.
compss_wait_on(obj, to_write=True)	Synchronizes for the last version of an object (or a list of objects) and returns it.

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 39). 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 39: @local decorator example

```
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
```

(continues on next page)

(continued from previous page)

```
@task(returns=list)
@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)
```

Important Notes

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 40).

Code 40: Task return value generation

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

...

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

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 41).

Code 41: Task return value subsequent usage

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

    ...

    another_task(o)

    ...

    o.yet_another_task()
```

In order to synchronize the object from the main program, the programmer has to synchronize (using the *compss_wait_on* function) in the same way as with any object updated by a task (Code 42).

Code 42: Task return value synchronization

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

    ...

    o = compss_wait_on(o)
```

Moreover, 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 43. 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 43: 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)
```

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 44).

Code 44: Using user-defined classes as task returns

```
# 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_func():
    ...
    myc = MyClass()
    ...
    return myc

if __name__=='__main__':
```

(continues on next page)

(continued from previous page)

```
    o = ret_func()

    # 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()
```

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

3.2.2 Application Execution

The next subsections describe how to execute applications with the COMPSs Python binding.

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/*)

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 \
                --library_path=/home/user/libdir \
                $TEST_DIR/test.py arg1 arg2
```

Or alternatively, use the `pycompss` module:

```
compss@bsc:~$ python -m pycompss \
                --pythonpath=$TEST_DIR \
                --library_path=/home/user/libdir \
                $TEST_DIR/test.py arg1 arg2
```

For full description about the options available for the runcompss command please check the *Application execution* Section.

3.2.3 Development 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.

Environment

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/*)

LD_LIBRARY_PATH The path where the *libbindings-commons.so* library is located (e.g. */opt/COMPSs/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/*).

API calls

In this case, the user is responsible of starting and stopping the COMPSs runtime. To this end, PyCOMPSs provides a module that provides two API calls: one for starting the COMPSs runtime, and another for stopping it.

This module can be imported from the *pycompss* library:

```
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:

```
# Previous user code

ipycompss.start()

# User code that can benefit from PyCOMPSs

ipycompss.stop()

# Subsequent code
```

Between the *start* and *stop* function calls, the user can write its own python code including PyCOMPSs imports, decorators and synchronization calls described in Section [subsec:Python_programming_model]. 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 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)
monitor	Integer	Monitor refresh rate (Default: None - Monitoring disabled)
project_xml	String	Path to the project XML file (Default: $COMPSS/Runtime/configuration/xml/p
resources_xml	String	Path to the resources XML file (Default: $COMPSs/Runtime/configuration/xm
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)

Parameter Name	Parameter Type	Description
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 created i
specific_log_dir	String	Use a specific directory to store COMPSs log files (the folder MUST exist and
extrae_cfg	String	Sets a custom extrae config file. Must be in a shared disk between all COMPSs
comm	String	Class that implements the adaptor for communications. Supported adaptors: “e
conn	String	Class that implements the runtime connector for the cloud. Supported connecto
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 adaptor (Defau
scheduler	String	Class that implements the Scheduler for COMPSs. Supported schedulers: “es.b
jvm_workers	String	Extra options for the COMPSs Workers JVMs. Each option separed by “,” and
cpu_affinity	String	Sets the CPU affinity for the workers. Supported options: “disabled”, “automat
gpu_affinity	String	Sets the GPU affinity for the workers. Supported options: “disabled”, “automat
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 execution (Defau
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 Optimizer (D
propatage_virtual_environment	Boolean	Propagate the master virtual environment to the workers (DefaultFalse)
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.

```
# Previous user code

ipycompss.start(graph=True, trace=True, monitor=2000)

# User code that can benefit from PyCOMPSs

ipycompss.stop(sync=True)

# Subsequent code
```

Application execution

The application can be executed as a common Jupyter notebook by steps or the whole application.

Attention: Once the COMPSs runtime has been stopped it is necessary to restart the python kernel in Jupyter before starting another COMPSs runtime. To this end, click on “Kernel” and “Restart” (or “Restart & Clear Output” or “Restart & Run All”, depending on the need).

3.2.4 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 functios 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 that task.

```
from pycompss.api.task import task      # Import @task decorator
from numba import jit

@task(returns=1)
@jit()
def numba_func(a, b):
    ...
```

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. This 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 *jit* to a function:

```
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*):

```
from pycompss.api.task import task

@task(numba='jit', numba_flags={'parallel':True})
```

(continues on next page)

(continued from previous page)

```
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
```

Details about numba and the specification of the signature, declaration and flags can be found in the Numba's webpage (http://numba.pydata.org/).

3.3 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.

3.3.1 Programming Model

Task Selection

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 45: 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

Code 46 shows an example of matrix multiplication written in C++.

Code 46: Matrix multiplication

```
#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;
```

(continues on next page)

(continued from previous page)

```
}
```

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.

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.

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.

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.

```
#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);
    void print();

private:
```

(continues on next page)

(continued from previous page)

```
    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>*.

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();
```

(continues on next page)

(continued from previous page)

```
}

#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";
       }
}
```

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);

};
```

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);

};
```

3.3.2 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.

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.

3.3.3 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.

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.

Therefore, let's see two examples, first, the application is going to be build for the native architecture, in our case *x86-64*, and then for a target architecture, for instance *armhf*. 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.

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;

```
$~/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

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
```

(continues on next page)

(continued from previous page)

```
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.

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
```

(continues on next page)

(continued from previous page)

```
...

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).
```

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.

3.3.4 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 \
            --lang=c \
            -g \
            /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 *Application execution*. .

3.3.5 Task Dependency Graph

Figure 3 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.

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.

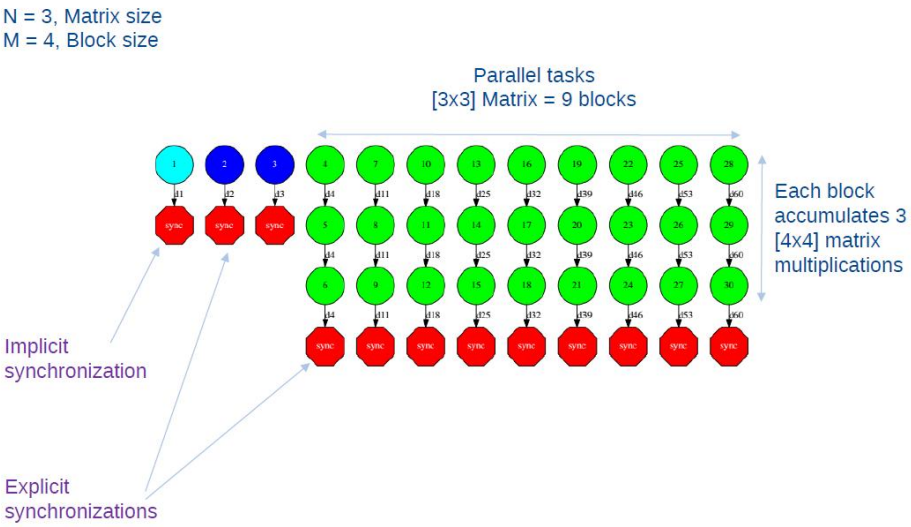


Figure 3: Matmul Execution Graph.

3.4 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	ComputingUnits	<string>	"1"	Required number of computing units
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

3.5 Known Limitations

The current COMPSs version () has the following limitations:

- **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.
- **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.

```
// @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();
```

(continues on next page)

(continued from previous page)

```
    }
    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();
    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());
}
```

- **Services types:** The current COMPSs version only supports SOAP based services that implement the WS interoperability standard. REST services are not supported.
- **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.
- **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 doesn't respect the tasks contraints. Notice that if no contraints 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:

```
# a.py
class A:
  def __init__(self, b):
    self.b  = b

# main.py
from a import A
from pycompss.api.task import task
```

(continues on next page)

(continued from previous page)

```
from pycompss.api.parameter import *

@task(obj = IN, returns = int)
def get_b(obj):
  return obj.b

@task(obj = INOUT)
def inc(obj):
  obj += [1]

def main():
  from pycompss.api.api import compss_wait_on
  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.


BSC
Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación

CHAPTER 4

Application execution

4.1 Executing COMPSs applications

4.1.1 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 *buildapp* command.

For further information about how to develop COMPSs applications please refer to *Application development*.

4.1.2 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: runcompss [options] application_name application_arguments

* Options:
General:
  --help, -h                                Print this help message

```

(continues on next page)

(continued from previous page)

```
  --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=<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 true
                                          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

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/2.6.pr/Runtime/
↪configuration/xml/projects/default_project.xml
  --resources=<path>                      Path to the resources XML file
                                          Default: /apps/COMPSs/2.6.pr/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 | debug
                                          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
  --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
```

(continues on next page)

(continued from previous page)

```
  --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: null
  --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.fullGraphScheduler.FullGraphScheduler
                                                                | es.bsc.compss.
↪scheduler.fifoScheduler.FIFOScheduler
                                                                | es.bsc.compss.
↪scheduler.resourceEmptyScheduler.ResourceEmptyScheduler
                                            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/bsc19/bsc19234
  --pythonpath=<path>                       Additional folders or paths to add to the␣
↪PYTHONPATH
                                            Default: /home/bsc19/bsc19234
  --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]
```

(continues on next page)

(continued from previous page)

```
  --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:
  --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
  --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

* 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:
```

(continues on next page)

(continued from previous page)

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

4.1.3 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*.

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.

Running C/C++ applications

To launch a COMPSs C/C++ application users have to compile the C/C++ application by means of the `buildapp` command. For further information please refer to *Application development*. 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

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

4.1.4 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 information about the *Eclipse IDE* please refer to *COMPSs IDE* Section.

For further details please check the *Configuration Files* Subsection inside the *Installation and Administration* Section.

4.2 Results and logs

4.2.1 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 4 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 4: 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 5 and Figure 6 show an example of the results obtained from the execution of the *Hello* Java application. While Figure 5 provides the output of the sequential execution of the application (without COMPSs), Figure 6 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.

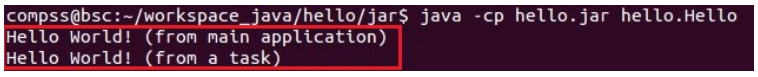


Figure 5: Sequential execution of the *Hello* java application

4.2.2 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.

```
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: IT Properties file is null. Setting default values
[(744)    API]  -  Deploying COMPSs Runtime
[(747)    API]  -  Starting COMPSs Runtime
[(747)    API]  -  Initializing components
[(1193)   API]  -  Ready to process tasks
Hello World! (from main application)
[(1203)   API]  -  Creating task from method sayHello in hello.HelloImpl
[(1203)   API]  -  There is 0 parameter
[(1235)   API]  -  No more tasks for app 1
[(3776)   API]  -  Getting Result Files 1
[(3777)   API]  -  Stop IT reached
[(3778)   API]  -  Stopping AP...
[(3779)   API]  -  Stopping TD...
[(3932)   API]  -  Stopping Comm...
[(3934)   API]  -  Runtime stopped
[(3934)   API]  -  Execution Finished

------------------------------------------------------------
compss@bsc:~/tutorial_apps/java/hello/jar$ more ~/.COMPSs/hello.Hello_01/jobs/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
```

Figure 6: COMPSs execution of the *Hello* java application

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 7 shows the logs generated by the execution of the Simple java application (without errors) in **off** mode.

```
.COMPSs/
└── [4.0K]  simple.Simple_01
    ├── [4.0K]  jobs
    ├── [   0]  resources.log
    ├── [   0]  runtime.log
    └── [4.0K]  tmpFiles
```

Figure 7: 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 8 shows the logs generated by the same execution than the previous case but with **info** mode.

```
.COMPSs/
└── [4.0K]  simple.Simple_02
    ├── [4.0K]  jobs
    ├── [ 612]  resources.log
    ├── [ 10K]  runtime.log
    └── [4.0K]  tmpFiles
```

Figure 8: 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 *COMPSs Monitor*.

Figure 9 and Figure 10 provide the content of these two files generated by the execution of the *Simple* java application.

```
compss@bsc:~/.COMPSs/simple.Simple_02$ cat runtime.log
[(732)(2015-08-20 16:34:30,731)       TaskScheduler]    @<init>           -  Initialization finished
[(738)(2015-08-20 16:34:30,737)       TaskScheduler]    @<init>           -  Initialization finished
[(742)(2015-08-20 16:34:30,741)          JobManager]    @<init>           -  Initialization finished
[(742)(2015-08-20 16:34:30,741)      TaskDispatcher]    @<init>           -  Initialization finished
[(748)(2015-08-20 16:34:30,747)       TaskAnalyser]    @<init>           -  Initialization finished
[(753)(2015-08-20 16:34:30,752)       TaskScheduler]    @esourcesCreated  -  Resource http://bscgrid05.bsc.es:20390/hmmerobj/hmmerobj?wsdl
 created
[(753)(2015-08-20 16:34:30,752)    DataInfoProvider]    @<init>           -  Initialization finished
[(787)(2015-08-20 16:34:30,786)       TaskAnalyser]    @processTask      -  New method task(increment), ID = 1
[(791)(2015-08-20 16:34:30,790)       TaskScheduler]    @scheduleTask     -  Blocked: Task(1, increment)
[(1479)(2015-08-20 16:34:31,478)       Communication]    @etWorkerIsReady  -  Notifying that worker is ready localhost
[(1892)(2015-08-20 16:34:31,891)       TaskScheduler]    @esourcesCreated  -  Resource localhost created
[(1893)(2015-08-20 16:34:31,892)       TaskScheduler]    @asksForResource  -  Available Resource: localhost. Task: 1, score: 0
[(1894)(2015-08-20 16:34:31,893)          JobManager]    @processJob       -  New Job 1 (Task: 1)
[(1894)(2015-08-20 16:34:31,893)          JobManager]    @processJob       -    * Method name: increment
[(1895)(2015-08-20 16:34:31,894)          JobManager]    @processJob       -    * Target host: localhost
[(1899)(2015-08-20 16:34:31,898)       Communication]    @submit           -  Submit NIOJob with ID 1
[(1944)(2015-08-20 16:34:31,943)          JobManager]    @completedJob     -  Received a notification for job 1 with state OK
[(1945)(2015-08-20 16:34:31,944)       TaskProcessor]    @notifyTaskEnd    -  Notification received for task 1 with end status FINISHED
[(1946)(2015-08-20 16:34:31,945)       TaskProcessor]    @waitForTask      -  End of waited task for data 1
[(1955)(2015-08-20 16:34:31,954)       TaskProcessor]    @noMoreTasks      -  All tasks finished
[(1962)(2015-08-20 16:34:31,961)       TaskProcessor]    @run              -  AccessProcessor shutdown
[(1965)(2015-08-20 16:34:31,964)       Communication]    @stop             -  Shutting down localhost:43001
```

Figure 9: runtime.log generated by the execution of the *Simple* java application

```
compss@bsc:~/.COMPSs/simple.Simple_02$ cat resources.log
TIMESTAMP = 1440081270727
INFO_MSG = [New resource available in the pool. Name = http://bscgrid05.bsc.es:20390/hmmerobj/hmmerobj?wsdl]
TIMESTAMP = 1440081270752
LOAD_INFO = [
        CORE_INFO = [
                COREID = 0
                NO_RESOURCE = 0
                TO_RESCHEDULE = 0
                ORDINARY = 0
                MIN = 100
                MEAN = 100
                MAX = 100
        ]
]

TIMESTAMP = 1440081271891
INFO_MSG = [New resource available in the pool. Name = localhost]
TIMESTAMP = 1440081271962
INFO_MSG = [Stopping all workers]
TIMESTAMP = 1440081271962
LOAD_INFO = [
        CORE_INFO = [
                COREID = 0
                NO_RESOURCE = 0
                TO_RESCHEDULE = 0
                ORDINARY = 0
                MIN = 56
                MEAN = 56
                MAX = 56
        ]
]
```

Figure 10: resources.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 11 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 *COMPSs Tools*.

```
.COMPSs/
└── [4.0K]  simple.Simple_03
    ├── [4.0K]  jobs
    │   ├── [   0]  job1_NEW.err
    │   └── [ 380]  job1_NEW.out
    ├── [ 612]  resources.log
    ├── [ 70K]  runtime.log
    └── [4.0K]  tmpFiles
```

Figure 11: Structure of the logs folder for the Simple java application in **debug** mode

4.3 COMPSs Tools

4.3.1 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; the graph file is written in the `base_log_folder/monitor/complete_graph.dot` at the end of the execution.

Figure 12 shows a dependency graph example of a *SparseLU* java application. The graph can be visualized by running the following command:

```
compss@bsc:~$ compss_gengraph ~/.COMPSs/sparseLU.arrays.SparseLU_01/monitor/complete_
↪graph.dot
```

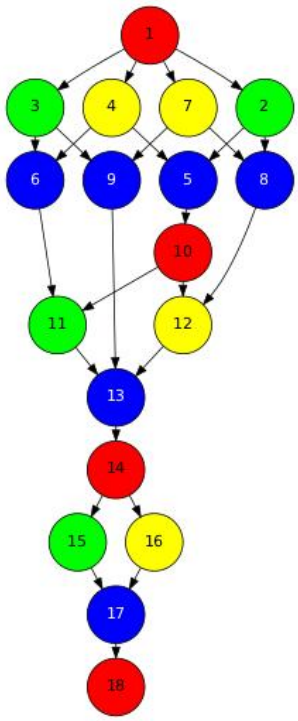


Figure 12: The dependency graph of the SparseLU application

4.3.2 COMPSs 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}
```

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).

Usage

In order to use the COMPSs Monitor users need to start the service as shown in Figure 13.

```
compss@bsc:~$ /etc/init.d/compss-monitor start
* Starting COMPSs Monitor
* Checking JAVA Installation...
Warning: JRE_HOME not defined
Info: JAVA_HOME found.
  Loading JRE_HOME from JAVA_HOME
   Success
* Checking IT_HOME...
WARNING: IT_HOME not defined. Trying default location /opt/COMPSs/
   Success
* Checking IT_MONITOR...
     IT_MONITOR=/home/compss/.COMPSs/
   Success
* Checking COMPSs Monitor Port...
Warning: COMPSs_MONITOR_PORT not defined.
  Loading from configuration file.
      COMPSs_MONITOR_PORT=8080
   Success
* Checking COMPSs Monitor Timeout...
Warning: COMPSs_MONITOR_TIMEOUT not defined.
  Loading from configuration file.
      COMPSs_MONITOR_TIMEOUT=20000
   Success
* Configuring COMPSs Monitor service...
   Success
Using CATALINA_BASE:   /opt/COMPSs/Tools/monitor/apache-tomcat
Using CATALINA_HOME:   /opt/COMPSs/Tools/monitor/apache-tomcat
Using CATALINA_TMPDIR: /opt/COMPSs/Tools/monitor/apache-tomcat/temp
Using JRE_HOME:        /usr/lib/jvm/java-8-openjdk-amd64//jre
Using CLASSPATH:       /opt/COMPSs/Tools/monitor/apache-tomcat/bin/bootstrap.jar:/opt/COMPSs/Tools/monitor/apache-tomcat/bin/tomcat-juli.jar
Tomcat started.
```

Figure 13: COMPSs Monitor start command

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 14, the users can select any of their executed or running COMPSs applications and display it.

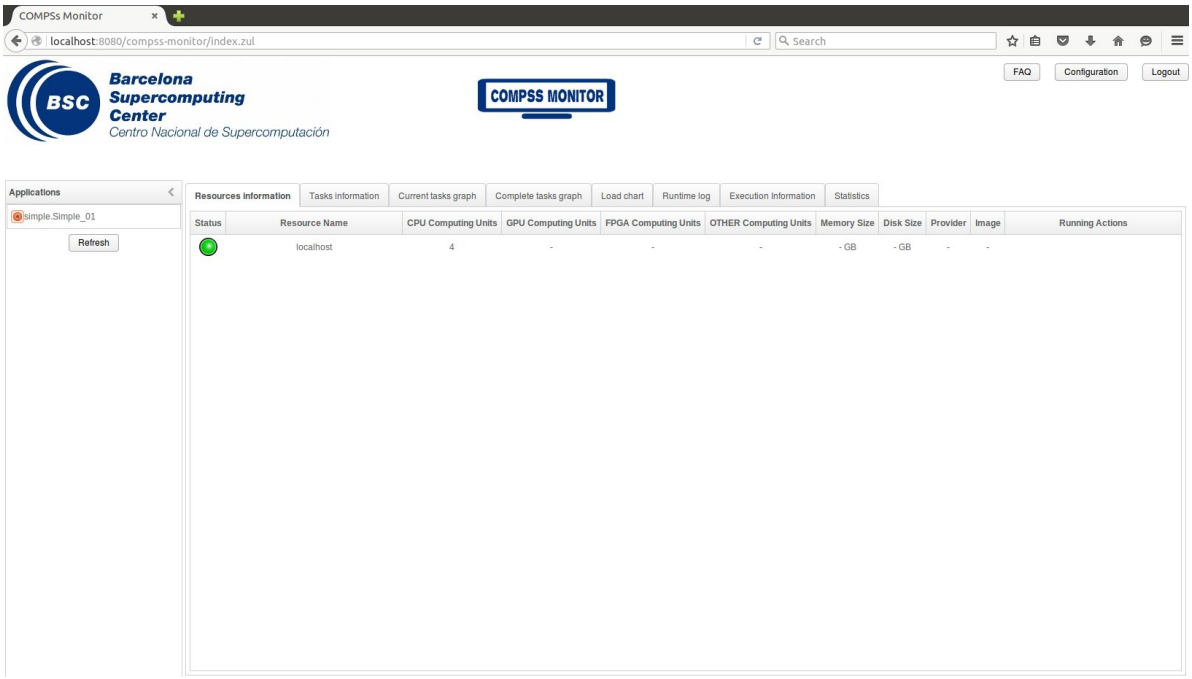


Figure 14: COMPSs monitoring interface

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 14 and Figure 15). 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
[(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
```

(continues on next page)

(continued from previous page)

```
[(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

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

```
compss@bsc:~$ cd .COMPSs/
compss@bsc:~/.COMPSs$ tree
.
└── simple.Simple_01
    ├── jobs
    │   ├── job1_NEW.err
    │   └── job1_NEW.out
    ├── monitor
    │   ├── complete_graph.dot
    │   ├── COMPSs_state.xml
    │   └── current_graph.dot
    ├── resources.log
    ├── runtime.log
    └── tmpFiles
```

Figure 15: Logs generated by the Simple java application with the monitoring flag enabled

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.

4.3.3 Application tracing

COMPSs Runtime can generate a post-execution trace of the execution of the application. This trace is useful for performance analysis and diagnosis.

A trace file may contain different events to determine the COMPSs master state, the task execution state or the file-transfers. The current release does not support file-transfers informations.

During the execution of the application, an XML file is created in the worker nodes to keep track of these events. At the end of the execution, all the XML files are merged to get a final trace file.

In this manual we only provide information about how to obtain a trace and about the available Paraver (the tool used to analyze the traces) configurations. For further information about the application instrumentation or the trace visualization and configurations please check the *Tracing* Section.

Trace Command

In order to obtain a post-execution trace file one of the following options `-t`, `--tracing`, `--tracing=true`, `--tracing=basic` must be added to the `runcompss` command. All this options activate the basic tracing mode; the advanced mode is activated with the option `--tracing=advanced`. For further information about advanced mode check the *Tracing* Section. Next, we provide an example of the command execution with the basic tracing option enabled for a java K-Means application.

```
compss@bsc:~$ runcompss -t kmeans.Kmeans
*** RUNNING JAVA APPLICATION KMEANS
[  INFO] Relative Classpath resolved: /path/to/jar/kmeans.jar

----------------- Executing kmeans.Kmeans --------------------------

Welcome to Extrae VERSION
Extrae: Parsing the configuration file (/opt/COMPSs/Runtime/configuration/xml/tracing/
↪extrae_basic.xml) begins
Extrae: Warning! <trace> tag has no <home> property defined.
Extrae: Generating intermediate files for Paraver traces.
Extrae: <cpu> tag at <counters> level will be ignored. This library does not support␣
↪CPU HW.
Extrae: Tracing buffer can hold 100000 events
Extrae: Circular buffer disabled.
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 (/opt/COMPSs/Runtime/configuration/xml/tracing/
↪extrae_basic.xml) has ended
Extrae: Intermediate traces will be stored in /user/folder
Extrae: Tracing mode is set to: Detail.
Extrae: Successfully initiated with 1 tasks and 1 threads

WARNING: COMPSs Properties file is null. Setting default values
[(751)    API]  -  Deploying COMPSs Runtime v<version>
[(753)    API]  -  Starting COMPSs Runtime v<version>
[(753)    API]  -  Initializing components
[(1142)   API]  -  Ready to process tasks
...
...
...
merger: Output trace format is: Paraver
merger: Extrae 3.3.0 (revision 3966 based on extrae/trunk)
```

(continues on next page)

(continued from previous page)

```
mpi2prv: Assigned nodes < Marginis >
mpi2prv: Assigned size per processor < <1 Mbyte >
mpi2prv: File set-0/TRACE@Marginis.0000001904000000000000.mpit is object 1.1.1 on␣
↪node Marginis assigned to processor 0
mpi2prv: File set-0/TRACE@Marginis.0000001904000000000001.mpit is object 1.1.2 on␣
↪node Marginis assigned to processor 0
mpi2prv: File set-0/TRACE@Marginis.0000001904000000000002.mpit is object 1.1.3 on␣
↪node Marginis assigned to processor 0
mpi2prv: File set-0/TRACE@Marginis.0000001980000001000000.mpit is object 1.2.1 on␣
↪node Marginis assigned to processor 0
mpi2prv: File set-0/TRACE@Marginis.0000001980000001000001.mpit is object 1.2.2 on␣
↪node Marginis assigned to processor 0
mpi2prv: File set-0/TRACE@Marginis.0000001980000001000002.mpit is object 1.2.3 on␣
↪node Marginis assigned to processor 0
mpi2prv: File set-0/TRACE@Marginis.0000001980000001000003.mpit is object 1.2.4 on␣
↪node Marginis assigned to processor 0
mpi2prv: File set-0/TRACE@Marginis.0000001980000001000004.mpit is object 1.2.5 on␣
↪node Marginis assigned to processor 0
mpi2prv: Time synchronization has been turned off
mpi2prv: A total of 9 symbols were imported from TRACE.sym file
mpi2prv: 0 function symbols imported
mpi2prv: 9 HWC counter descriptions imported
mpi2prv: Checking for target directory existance... 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 838848 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 991743 bytes
mpi2prv: Removing temporal files... done
mpi2prv: Elapsed time removing temporal files: 0 hours 0 minutes 0 seconds
mpi2prv: Congratulations! ./trace/kmeans.Kmeans_compss_trace_1460456106.prv has been␣
↪generated.
[   API]  -  Execution Finished

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

At the end of the execution the trace will be stored inside the `trace` folder under the application log directory.

```
compss@bsc:~$ cd .COMPSs/kmeans.Kmeans_01/trace/
compss@bsc:~$ ls -1
kmeans.Kmeans_compss_trace_1460456106.pcf
kmeans.Kmeans_compss_trace_1460456106.prv
kmeans.Kmeans_compss_trace_1460456106.row
```

Trace visualization

The traces generated by an application execution are ready to be visualized with *Paraver*. *Paraver* is a powerful tool developed by *BSC* that allows users to show many views of the trace data by means of different configuration files. Users can manually load, edit or create configuration files to obtain different trace data views.

If *Paraver* is installed, issue the following command to visualize a given tracefile:

```
compss@bsc:~$ wxparaver path/to/trace/trace_name.prv
```

For further information about *Paraver* please visit the following site:

http://www.bsc.es/computer-sciences/performance-tools/paraver

4.3.4 COMPSs IDE

COMPSs IDE is an Integrated Development Environment to develop, compile, deploy and execute COMPSs applications. It is available through the *Eclipse Market* as a plugin and provides an even easier way to work with COMPSs.

For further information please check the *COMPSs IDE User Guide* available at: http://compss.bsc.es .

4.4 Special Execution Platforms

This section provides information about how to run COMPSs Applications in specific platforms such as *Docker*, *Chameleon* or *MareNostrum*.

4.4.1 Docker

Introduction

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

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:

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'
```

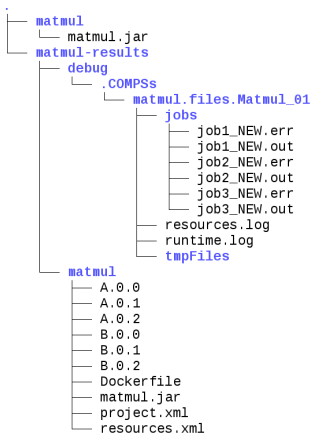


Figure 16: Result and log folders of a *Matmul* execution with COMPSs and Docker

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' \
```

(continues on next page)

(continued from previous page)

```
                    --c='/home/laura/apps/kmeans' \
                    --image-name='laura-67/my-kmeans' \
                    --classpath=/home/laura/apps/kmeans/kmeans.jar \
                    kmeans.KMeans
```

4.4.2 Chameleon

Introduction

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

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.

4.4.3 SuperComputers

To maintain the portability between different environments, COMPSs has a pre-build structure (see Figure [fig:queue_scripts_structure]) to execute applications in SuperComputers. For this purpose, users must use the `enqueue_compss` script provided in the COMPSs installation. This script has several parameters (see `enqueue_compss -h`) that allow users to customize their executions for any SuperComputer.

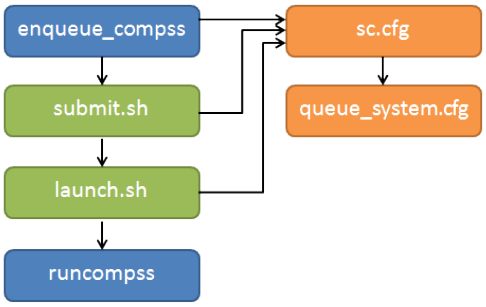


Figure 17: Structure of COMPSs queue scripts. In Blue user scripts, in Green queue scripts and in Orange system dependant scripts

To make this structure works, the administrators must define a configuration file for the queue system and a configuration file for the specific SuperComputer parameters. The COMPSs installation already provides queue configurations for *LSF* and *SLURM* and several examples for SuperComputer configurations. To create a new configuration we recommend to use one of the configurations provided by COMPSs (such as the configuration for the *MareNostrum IV* SuperComputer) or to contact us at support-compss@bsc.es .

For information about how to submit COMPSs applications at any Supercomputer please refer to *Supercomputers*.

4.5 Common Issues

This section provides answers for the most common issues of the execution of COMPSs applications. For specific issues not covered in this section, please do not hesitate to contact us at: support-compss@bsc.es .

4.5.1 How to debug

When the application does not behave as expected the first thing users must do is to run it in **debug** mode executing the `runcompss` command withthe `-d` flag to enable the debug log level.

In this case the application execution will produce the following files:

- `runtime.log`
- `resources.log`
- `jobs` folder

First, 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.

4.5.2 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.

4.5.3 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_dsa.pub user@resourceName:./mydsa.pub
compss@bsc:~$ ssh user@resourceName "cat mydsa.pub >> ~/.ssh/authorized_keys; rm ./
↪mydsa.pub"
```

These commands are a quick solution, for further details please check the *Additional Configuration* Section.

4.5.4 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.

4.5.5 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.

4.5.6 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*.

4.5.7 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)
```

(continues on next page)

(continued from previous page)

```
        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).

4.5.8 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
```

(continues on next page)

(continued from previous page)

```
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
```

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.

CHAPTER 5

Supercomputers

5.1 Common usage

5.1.1 Available COMPSs modules

COMPSs is configured as a Linux Module. As shown in next Figure, the 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/release(default)
COMPSs/trunk


$ module load COMPSs/release
load java/1.8.0u66 (PATH, MANPATH, JAVA_HOME, JAVA_ROOT, JAVA_BINDIR,
                    SDK_HOME, JDK_HOME, JRE_HOME)
load MKL/11.0.1 (LD_LIBRARY_PATH)
load PYTHON/2.7.3 (PATH, MANPATH, LD_LIBRARY_PATH, C_INCLUDE_PATH)
load COMPSs/release (PATH, MANPATH, COMPSS_HOME)
```

The following command can be run to check if the correct COMPSs version has been loaded:

```
$ enqueue_compss --version
COMPSs version <version>
```

5.1.2 Configuration

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 module needs to be loaded in all the nodes that will run COMPSs jobs. Consequently, the `module load` **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/1.8.0u66 (PATH, MANPATH, JAVA_HOME, JAVA_ROOT, JAVA_BINDIR,
                    SDK_HOME, JDK_HOME, JRE_HOME)
load MKL/11.0.1 (LD_LIBRARY_PATH)
load PYTHON/2.7.3 (PATH, MANPATH, LD_LIBRARY_PATH, C_INCLUDE_PATH)
load COMPSs/release (PATH, MANPATH, COMPSS_HOME)

USER@SC$ enqueue_compss --version
COMPSs version <version>
```

Important: Please remember that COMPSs runs in several nodes and your current enviroment is not exported to them. Thus, all the needed environment variables **must** be loaded through the *.bashrc* file.

Important: Please remember that PyCOMPSs uses Python 2.7 by default. In order to use Python 3, the Python 2.7 module **must** be unloaded after loading COMPSs module, and then load the Python 3 module.

5.1.3 COMPSs Job submission

COMPSs jobs can be easily submited by running the **enqueue_compss** command. This command allows to configure any **runcompss** 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: enqueue_compss [queue_system_options] [COMPSs_options]
          application_name [application_arguments]

* Options:
```

(continues on next page)

(continued from previous page)

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

(continues on next page)

(continued from previous page)

```
                                        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
    --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
    --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: scratch | gpfs |
↪<path>
```

(continues on next page)

(continued from previous page)

```
                                        Default: scratch

    --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
    --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

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

  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␣
↪true
                                        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
```

(continues on next page)

(continued from previous 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: /apps/COMPSs/2.6.pr/Runtime/
↪configuration/xml/projects/default_project.xml
    --resources=<path>                      Path to the resources XML file
                                            Default: /apps/COMPSs/2.6.pr/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 | debug
                                            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
    --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: null
    --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
```

(continues on next page)

(continued from previous page)

```
                                        Supported schedulers: es.bsc.compss.
↪scheduler.fullGraphScheduler.FullGraphScheduler
                                                            | es.bsc.compss.
↪scheduler.fifoScheduler.FIFOScheduler
                                                            | es.bsc.compss.
↪scheduler.resourceEmptyScheduler.ResourceEmptyScheduler
                                        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/bsc19/bsc19234
    --pythonpath=<path>                 Additional folders or paths to add to the␣
↪PYTHONPATH
                                        Default: /home/bsc19/bsc19234
    --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.
```

(continues on next page)

(continued from previous page)

```
                                          Default:
    --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
    --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

* 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.
```

5.2 MareNostrum 4

5.2.1 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 .

5.2.2 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=gpfs \
  --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=.
        --worker_working_dir=gpfs
        --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 144
#SBATCH --exclusive
#SBATCH -t00:15:00
...
```

In order to trac the jobs state users can run the following command:

```
$ squeue
JOBID   PARTITION  NAME    USER  TIME_LEFT  TIME_LIMIT   START_TIME  ST NODES  CPUS␣
↪NODELIST
```

(continues on next page)

(continued from previous page)

```
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 *Application execution* Section.

5.3 MinoTauro

5.3.1 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 .

5.3.2 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=gpfs \
  --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=gpfs
        --lang=python
        --classpath=.
        --library_path=.
```

(continues on next page)

(continued from previous page)

```
        --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 *Application execution* Section.

5.4 Nord 3

5.4.1 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 .

5.4.2 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 \
```

(continues on next page)

(continued from previous page)

```
  --cpus_per_node=16 \
  --master_working_dir=. \
  --worker_working_dir=gpfs \
  --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=gpfs
        --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
#
#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 *Application execution* Section.

5.5 Enabling COMPSs Monitor

5.5.1 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_dsa.pub ${SC_USER}@mn1.bsc.es:~/id_dsa_local.pub
compss@bsc:~$ ssh SC_USER@SC_ENDPOINT \
                  "cat ~/id_dsa_local.pub >> ~/.ssh/authorized_keys; \
                  rm ~/id_dsa_local.pub"
compss@bsc:~$ mkdir -p TARGET_LOCAL_FOLDER/.COMPSs
compss@bsc:~$ sshfs -o IdentityFile=$HOME/.ssh/id_dsa -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
```

5.5.2 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 *Application execution* Section.

Figure 18 illustrates how to login and Figure 19 shows the COMPSs Monitor main page for an application run inside a Supercomputer.

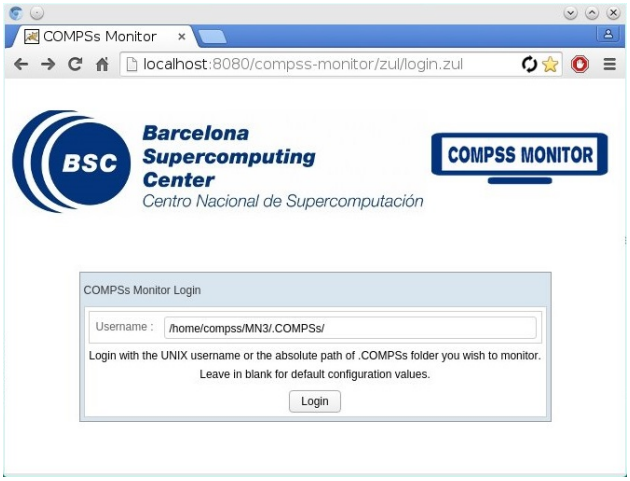


Figure 18: COMPSs Monitor login for Supercomputers

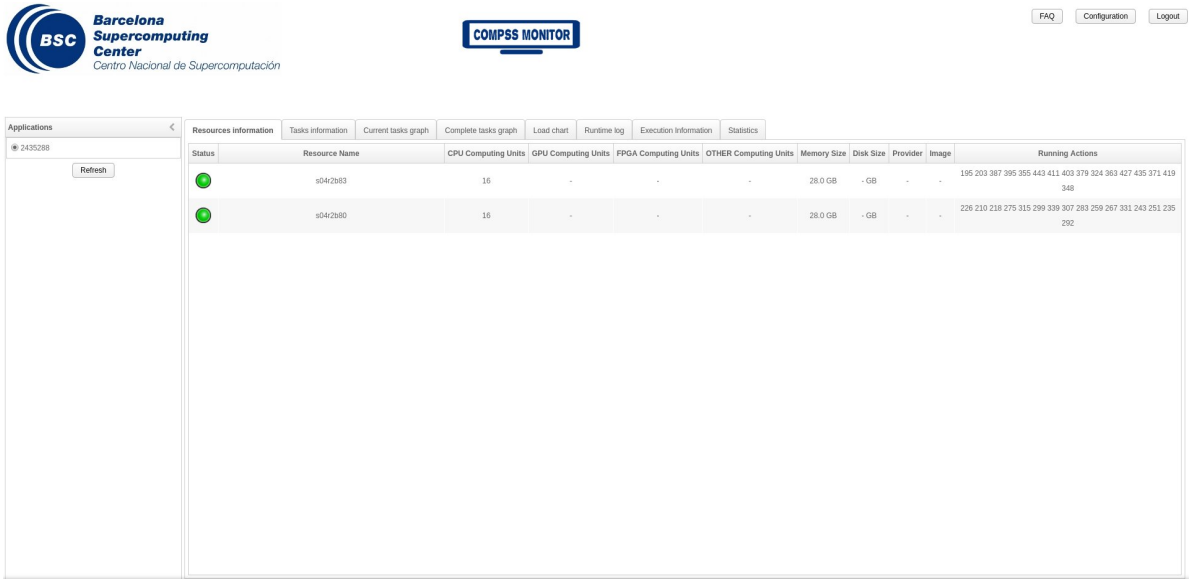


Figure 19: COMPSs Monitor main page for a test application at Supercomputers

CHAPTER 6

Tracing

6.1 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.

For each worker node and the master, Extrae keeps track of the events in an intermediate format file (with *.mpit* extension). At the end of the execution, all intermediate files are gathered and merged with Extrae's *mpi2prv* command in order to create the final tracefile, a Paraver format file (.prv). See the *Visualization* Sectio for further information about the Paraver tool.

When instrumentation is activated, Extrae outputs several messages corresponding to the tracing initialization, intermediate files' creation, and the merging process.

At present time, COMPSs tracing features two execution modes:

Basic Aimed at COMPSs applications developers

Advanced For COMPSs developers and users with access to its source code or custom installations

Next sections describe the information provided by each mode and how to use them.

6.1.1 Basic Mode

This mode is aimed at COMPSs' apps users and developers. It instruments computing threads and some management resources providing information about tasks' executions, data transfers, and hardware counters if PAPI is available

[1] For more information: https://www.bsc.es/computer-sciences/extrae
[2] For more information: https://www.bsc.es/computer-sciences/performance-tools/paraver

(see *PAPI: Hardware Counters* for more info).

Usage

In order to activate basic tracing one needs to provide one of the following arguments to the execution command:

- -t
- –tracing
- –tracing=basic
- –tracing=true

Examples given:

```
$ runcompss --tracing application_name application_args
```

Figure 20 was generated as follows:

```
$ runcompss \
     --lang=java \
     --tracing \
     --classpath=/path/to/jar/kmeans.jar \
     kmeans.KMeans
```

When tracing is activated, Extrae generates additional output to help the user ensure that instrumentation is turned on and working without issues. On basic mode this is the output users should see when tracing is working correctly:

```
*** RUNNING JAVA APPLICATION KMEANS
Resolved: /path/to/jar/kmeans.jar:

----------------- Executing kmeans.Kmeans ---------------------------

Welcome to Extrae VERSION
Extrae: Parsing the configuration file (/opt/COMPSs/Runtime/configuration/xml/tracing/
 ↪extrae_basic.xml) begins
Extrae: Tracing package is located on /opt/COMPSs/Dependencies/extrae/
Extrae: Generating intermediate files for Paraver traces.
Extrae: PAPI domain set to USER for HWC set 1
Extrae: HWC set 1 contains following counters < PAPI_TOT_INS (0x80000032) PAPI_TOT_
 ↪CYC (0x8000003b) PAPI_LD_INS (0x80000035) PAPI_SR_INS (0x80000036) > - changing␣
 ↪every 500000000 nanoseconds
Extrae: PAPI domain set to USER for HWC set 2
Extrae: HWC set 2 contains following counters < PAPI_TOT_INS (0x80000032) PAPI_TOT_
 ↪CYC (0x8000003b) PAPI_LD_INS (0x80000035) PAPI_SR_INS (0x80000036) PAPI_L2_DCM␣
 ↪(0x80000002) > - changing every 500000000 nanoseconds
WARNING: COMPSs Properties file is null. Setting default values
[(751)    API]  -  Deploying COMPSs Runtime v<version>
[(753)    API]  -  Starting COMPSs Runtime v<version>
[(753)    API]  -  Initializing components
[(1142)   API]  -  Ready to process tasks

...
...
...
merger: Output trace format is: Paraver
merger: Extrae VERSION
```

(continues on next page)

(continued from previous page)

```
mpi2prv: Assigned nodes < Marginis >
mpi2prv: Assigned size per processor < <1 Mbyte >
mpi2prv: File set-0/TRACE@Marginis.0000001904000000000000.mpit is object 1.1.1 on␣
↪node Marginis assigned to processor 0
mpi2prv: File set-0/TRACE@Marginis.0000001904000000000001.mpit is object 1.1.2 on␣
↪node Marginis assigned to processor 0
mpi2prv: File set-0/TRACE@Marginis.0000001904000000000002.mpit is object 1.1.3 on␣
↪node Marginis assigned to processor 0
mpi2prv: File set-0/TRACE@Marginis.0000001980000001000000.mpit is object 1.2.1 on␣
↪node Marginis assigned to processor 0
mpi2prv: File set-0/TRACE@Marginis.0000001980000001000001.mpit is object 1.2.2 on␣
↪node Marginis assigned to processor 0
mpi2prv: File set-0/TRACE@Marginis.0000001980000001000002.mpit is object 1.2.3 on␣
↪node Marginis assigned to processor 0
mpi2prv: File set-0/TRACE@Marginis.0000001980000001000003.mpit is object 1.2.4 on␣
↪node Marginis assigned to processor 0
mpi2prv: File set-0/TRACE@Marginis.0000001980000001000004.mpit is object 1.2.5 on␣
↪node Marginis assigned to processor 0
mpi2prv: Time synchronization has been turned off
mpi2prv: A total of 9 symbols were imported from TRACE.sym file
mpi2prv: 0 function symbols imported
mpi2prv: 9 HWC counter descriptions imported
mpi2prv: Checking for target directory existance... 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 838848 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 991743 bytes
mpi2prv: Removing temporal files... done
mpi2prv: Elapsed time removing temporal files: 0 hours 0 minutes 0 seconds
mpi2prv: Congratulations! ./trace/kmeans.Kmeans_compss_trace_1460456106.prv has been␣
↪generated.
[   API]  -  Execution Finished
Extrae: Tracing buffer can hold 100000 events
Extrae: Circular buffer disabled.
Extrae: Warning! <dynamic-memory> tag will be ignored. This library does support␣
↪instrumenting dynamic memory calls.
Extrae: Warning! <input-output> tag will be ignored. This library does support␣
↪instrumenting I/O calls.
Extrae: Dynamic memory instrumentation is disabled.
Extrae: Basic I/O memory instrumentation is disabled.
Extrae: Parsing the configuration file (/opt/COMPSs/Runtime/scripts/user/../../
↪configuration/xml/tracing/extrae_basic.xml) has ended
Extrae: Intermediate traces will be stored in /home/kurtz/compss/tests_local/app10
```

(continues on next page)

(continued from previous page)

```
Extrae: Tracing mode is set to: Detail.
Extrae: Successfully initiated with 1 tasks and 1 threads
```

It contains diverse information about the tracing, for example, Extrae version used (`VERSION` will be replaced by the actual number during executions), the XML configuration file used (`extrae_basic.xml`), the amount of threads instrumented (objects through 1.1.1 to 1.2.5), available hardware counters (`PAPI_TOT_INS (0x80000032) ... PAPI_L3_TCM (0x80000008)`) or the name of the generated tracefile (`./trace/kmeans.Kmeans_compss_trace_1460456106.prv`). When using NIO communications adaptor with debug activated, the log of each worker also contains the Extrae initialization information.

N.B. when using Python, COMPSs needs to perform an extra merging step in order to add the Python-produced events to the main tracefile. If Python events are not shown, check *runtime.log* file and search for the following expected output of this merging process to find possible errors:

```
[(9788)(2016-11-15 11:22:27,687)  Tracing]    @generateTrace -  Tracing: Generating␣
↪trace
[(9851)(2016-11-15 11:22:27,750)  Tracing]    @<init>        -  Trace's merger␣
↪initialization successful
[(9851)(2016-11-15 11:22:27,750)  Tracing]    @merge         -  Parsing master sync␣
↪events
[(9905)(2016-11-15 11:22:27,804)  Tracing]    @merge         -  Proceeding to merge␣
↪task traces into master
[(9944)(2016-11-15 11:22:27,843)  Tracing]    @merge         -  Merging finished,
[(9944)(2016-11-15 11:22:27,843)  Tracing]    @merge         -  Temporal task folder␣
↪removed.
```

Instrumented Threads

Basic traces instrument the following threads:

- Master node (3 threads)
  - COMPSs runtime
  - Task Dispatcher
  - Access Processor
- Worker node (1 + Computing Units)
  - Main thread
  - Number of threads available for computing

Information Available

The basic mode 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*)

Trace Example

Figure 20 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

Task Dispatcher - Thread 1.1.2 Shows information about the state and scheduling of the tasks to be executed.

Access Processor - Thread 1.1.3 All the events related to the tasks' parameters management, such as dependencies or transfers are shown in this thread.

Worker X Master - Thread 1.X.1 This thread is the master of each worker and handles the computing resources and transfers. Is 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 Computing Unit Y - Thread 1.X.Y Shows the actual tasks execution information and is repeated as many times as computing threads has the worker X

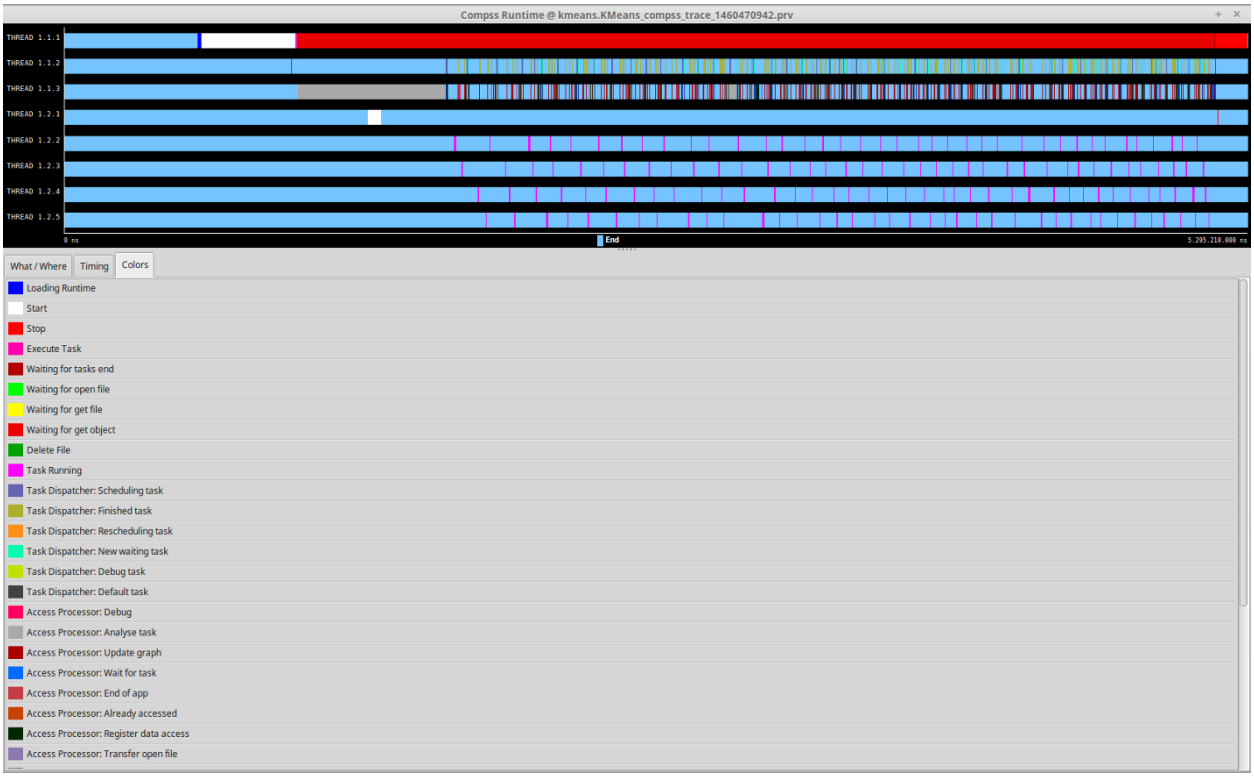


Figure 20: Basic mode tracefile for a k-means algorithm visualized with compss_runtime.cfg

6.1.2 Advanced Mode

This mode is for more advanced COMPSs' users and developers who want to customize further the information provided by the tracing or need rawer information like pthreads calls or Java garbage collection. With it, every single thread created during the execution is traced.

N.B.: The extra information provided by the advanced mode is only available on the workers when using NIO adaptor.

Usage

In order to activate the advanced tracing add the following option to the execution:

- –tracing=advanced

Examples given:

```
$ runcompss --tracing=advanced application_name application_args
```

Figure 21 was generated as follows:

```
$ runcompss \
     --lang=java \
     --tracing=advanced \
     --classpath=/path/to/jar/kmeans.jar \
     kmeans.KMeans
```

When advanced tracing is activated, the configuration file reported on the output is *extrae_advanced.xml*.

```
*** RUNNING JAVA APPLICATION KMEANS
...
...
...
Welcome to Extrae VERSION
Extrae: Parsing the configuration file (/opt/COMPSs/Runtime/scripts/user/../../
↪configuration/xml/tracing/extrae_advanced.xml) begins
```

This is the default file used for advanced tracing. However, advanced users can modify it in order to customize the information provided by Extrae. The configuration file is read first by the master on the *runcompss* script. When using NIO adaptor for communication, the configuration file is also read when each worker is started (on *persistent_worker.sh* or *persistent_worker_starter.sh* depending on the execution environment).

If the default file is modified, the changes always affect the master, and also the workers when using NIO. Modifying the scripts which turn on the master and the workers is possible to achieve different instrumentations for master/workers. However, not all Extrae available XML configurations work with COMPSs, some of them can make the runtime or workers crash so modify them at your discretion and risk. More information about instrumentation XML configurations on Extrae User Guide at: https://www.bsc.es/computer-sciences/performance-tools/trace-generation/extrae/extrae-user-guide.

Instrumented Threads

Advanced mode instruments all the pthreads created during the application execution. It contains all the threads shown on basic traces plus extra ones used to call command-line commands, I/O streams managers and all actions which create a new process. Due to the temporal nature of many of this threads, they may contain little information or appear just at specific parts of the execution pipeline.

Information Available

The advanced mode tracefiles contain the same information as the basic ones:

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*)

Trace Example

Figure Figure 21 shows the total completed instructions for a sample program executed with the advanced tracing mode. Note that the thread - resource correspondence described on the basic trace example is no longer static and thus cannot be inferred. Nonetheless, they can be found thanks to the named events shown in other configurations such as *compss_runtime.cfg*.

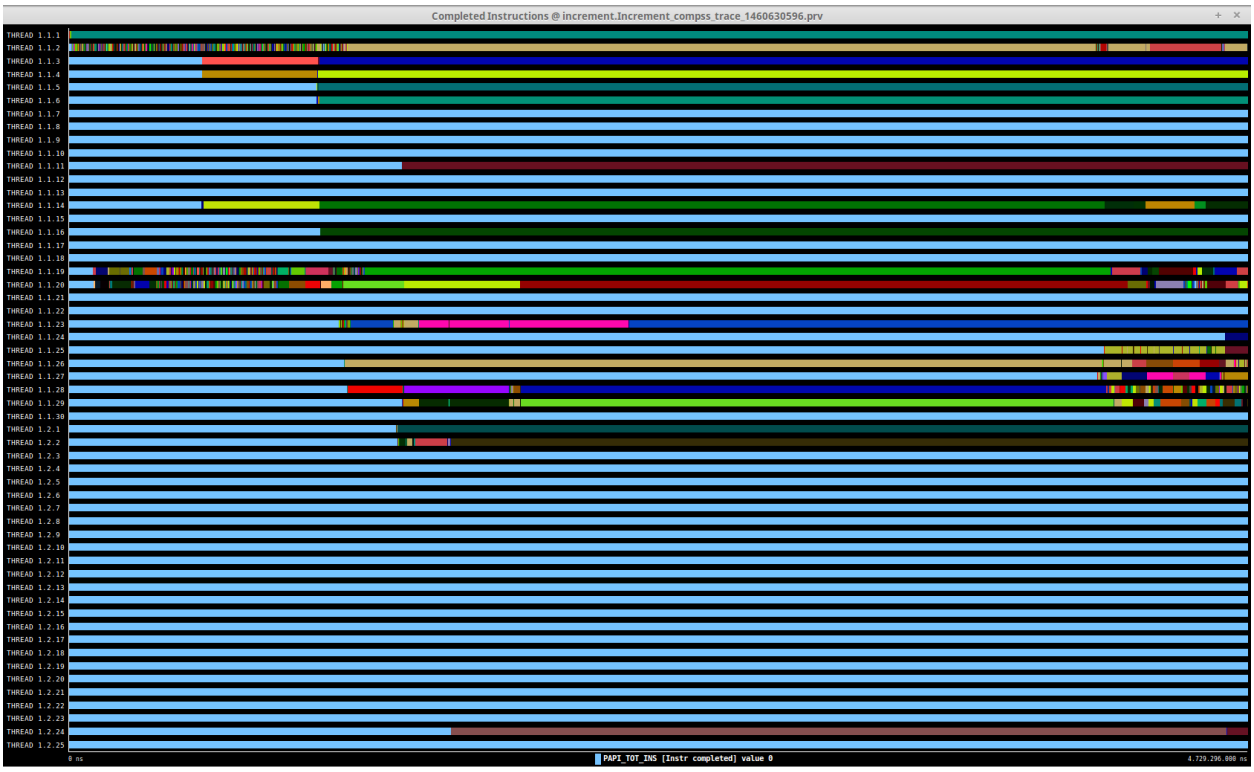


Figure 21: Advanced mode tracefile for a testing program showing the total completed instructions

For further information about Extrae, please visit the following site:

http://www.bsc.es/computer-science/extrae

6.1.3 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. To do so just pass the file as an execution

parameter as follows:

```
--extrae_config_file=/path/to/config/file.xml
```

The configuration file 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.

6.2 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

6.2.1 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
```

6.2.2 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 the *compss_tasks.cfg* 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 22 and Figure 23 show an example of this process.

6.2.3 View Adjustment

In a Paraver view, a red exclamation sign may appear in the bottom-left corner (see Figure 23 in the previous section). 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
  - Choose the option Fit Semantic Scale / Fit Both
- View Event Flags: marks with a green flag all the emitted the events.
  - Right click on the trace window

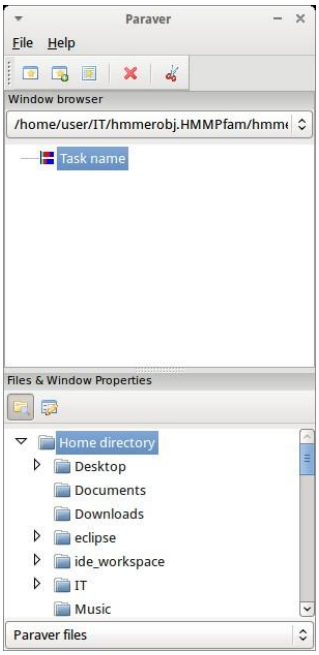


Figure 22: Paraver menu

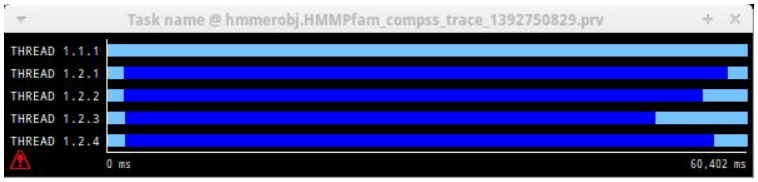


Figure 23: Trace file

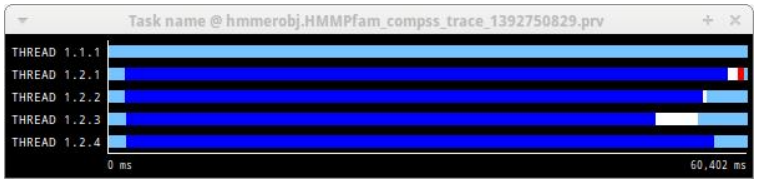


Figure 24: Paraver view adjustment: Fit window

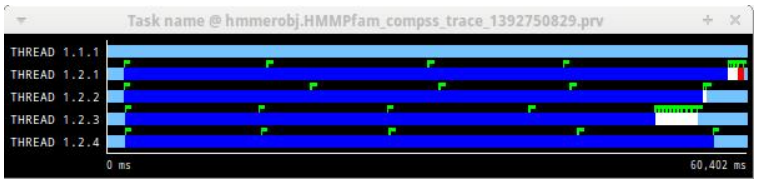


Figure 25: Paraver view adjustment: View Event Flags

  - Chose the option 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

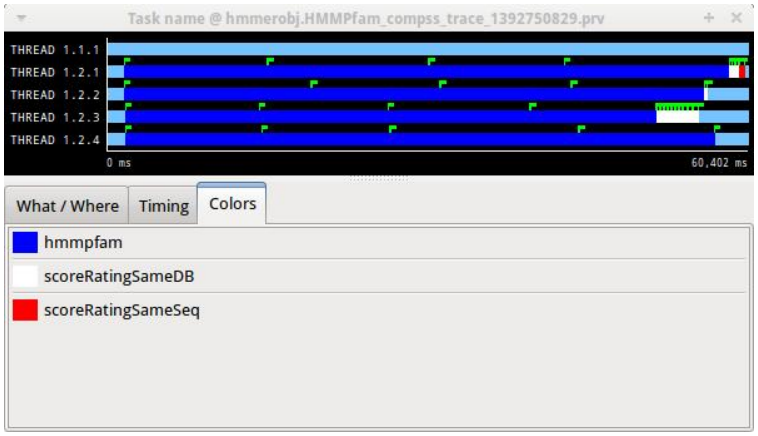


Figure 26: Paraver view adjustment: Show info panel

- 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: Zoom configuration

Figure 28: Paraver view adjustment: Zoom configuration

6.3 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 light blue 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.

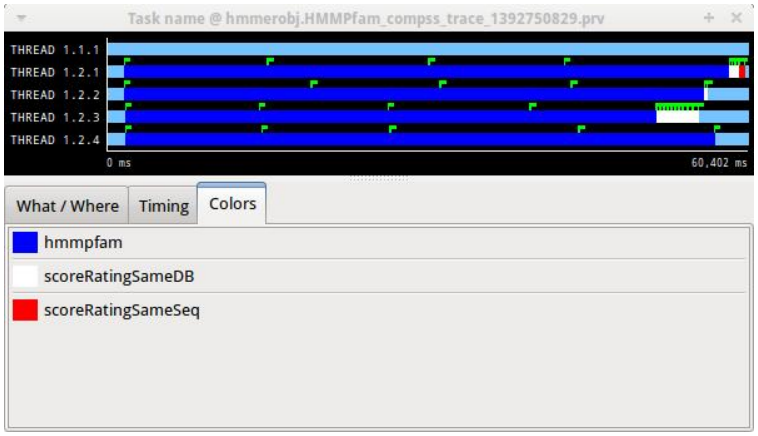


Figure 29: Trace interpretation

6.4 Analysis

This section gives some tips to analyze a COMPSs trace from two different points of view: graphically and numerically.

6.4.1 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 "hmmpfam" in dark blue appear to be well distributed among the four workers, each worker executes four "hmmpfam" tasks.

However, some workers finish earlier than the others, worker 1.2.3 finish the first and worker 1.2.1 the last. So there is an imbalance in the duration of "hmmpfam" tasks. The programmer should analyze then whether all the tasks process the same amount of input data and do the same thing in order to find out the reason for such imbalance.

Another thing to highlight is that tasks of type "scoreRatingSameDB" are not equally distributed among all the workers. Some workers execute more tasks of this type than the others. To understand better what happens here, one needs to take a look to the execution graph and also zoom in the last part of the trace.

There is only one task of type "scoreRatingSameSeq". This task appears in red in the trace (and in light-green in the graph). With the help of the graph we see that the "scoreRatingSameSeq" task has dependences on tasks of type "scoreRatingSameDB", in white (or yellow).

When the last task of type "hmmpfam" (in dark blue) ends, the previous dependencies are solved, and if we look at the graph, this means going across a path of three dependencies of type "scoreRatingSameDB" (in yellow). Moreover,

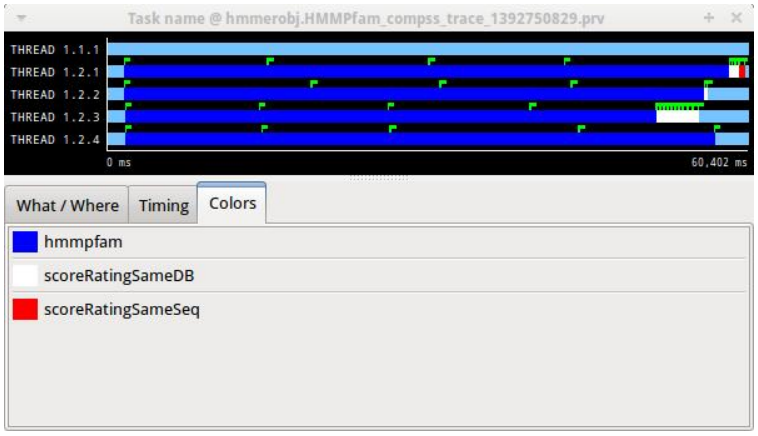


Figure 30: Basic trace view of a Hmmpfam execution.

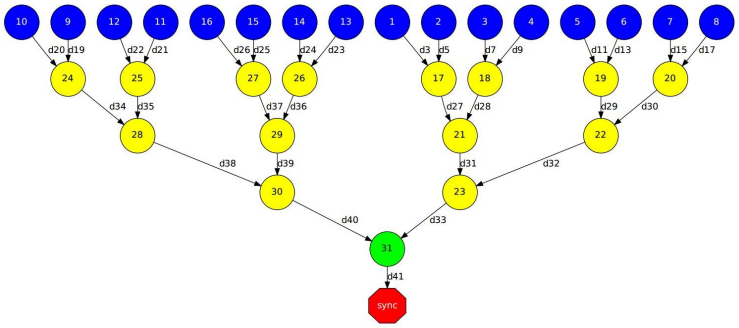


Figure 31: Data dependencies graph of a Hmmpfam execution.

Figure 32: Zoomed in view of a Hmmpfam execution.

because these are sequential dependencies (one depends on the previous) no more than a worker can be used at the same time to execute the tasks. This is the reason of why the last three task of type "scoreRatingSameDB" (in white) are executed in worker 1.2.1 sequentially.

6.4.2 Numerical Analysis

Here we show another trace from a different parallel execution of the Hmmer program.

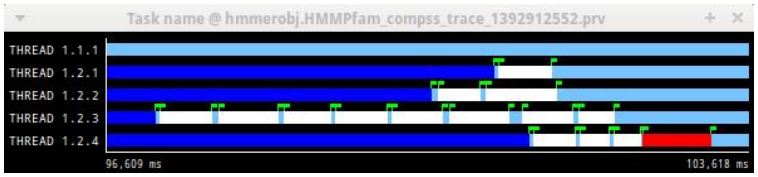
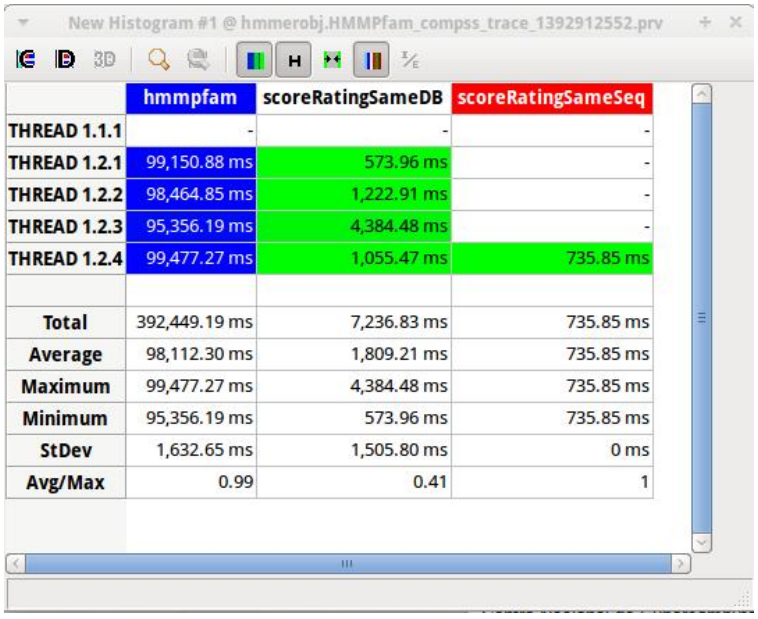


Figure 33: Original sample trace interval corresponding to the obtained Histogram.

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 34: 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. Task names appear in the same color than in the trace view. The color of a cell in a row corresponding to a worker ranges from light-green for lower values to dark-blue for higher ones. This conforms a color based histogram.



	hmmpfam	scoreRatingSameDB	scoreRatingSameSeq
THREAD 1.1.1	-	-	-
THREAD 1.2.1	99,150.88 ms	573.96 ms	-
THREAD 1.2.2	98,464.85 ms	1,222.91 ms	-
THREAD 1.2.3	95,356.19 ms	4,384.48 ms	-
THREAD 1.2.4	99,477.27 ms	1,055.47 ms	735.85 ms
Total	392,449.19 ms	7,236.83 ms	735.85 ms
Average	98,112.30 ms	1,809.21 ms	735.85 ms
Maximum	99,477.27 ms	4,384.48 ms	735.85 ms
Minimum	95,356.19 ms	573.96 ms	735.85 ms
StDev	1,632.65 ms	1,505.80 ms	0 ms
Avg/Max	0.99	0.41	1

Figure 35: Hmmpfam histogram corresponding to previous trace

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, it is possible to change the semantic of the statistics to see other factors rather than the time, for example, the number of bursts.

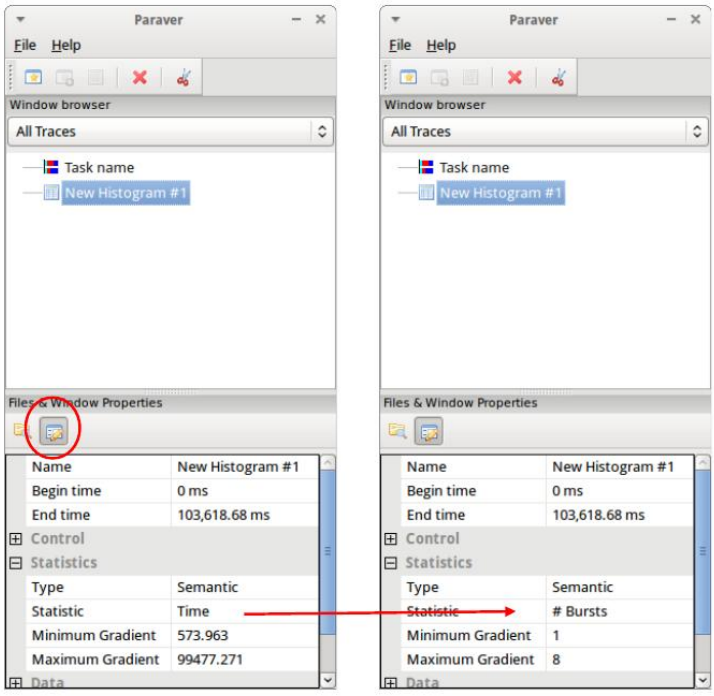


Figure 36: Paraver histogram options menu

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.

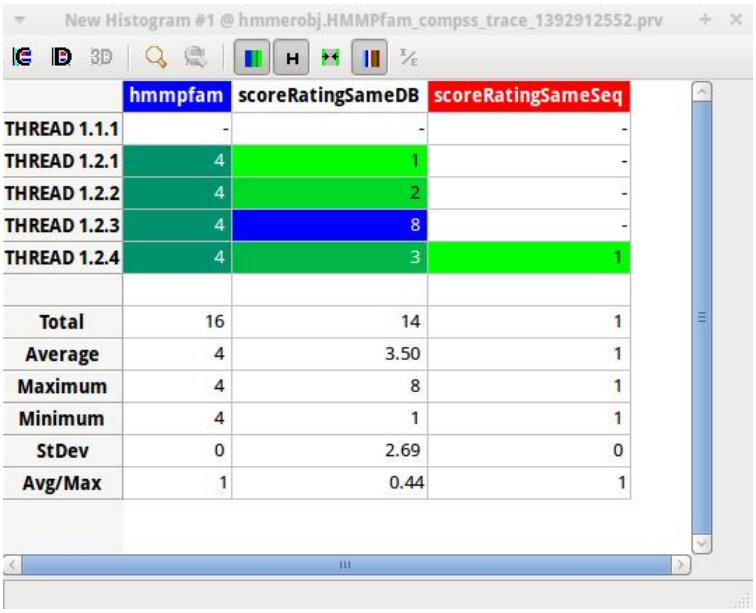
6.5 PAPI: Hardware Counters

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
```



	hmmpfam	scoreRatingSameDB	scoreRatingSameSeq
THREAD 1.1.1	-	-	-
THREAD 1.2.1	4	1	-
THREAD 1.2.2	4	2	-
THREAD 1.2.3	4	8	-
THREAD 1.2.4	4	3	1
Total	16	14	1
Average	4	3.50	1
Maximum	4	8	1
Minimum	4	1	1
StDev	0	2.69	0
Avg/Max	1	0.44	1

Figure 37: Hmmpfam histogram with the number of bursts

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 textitpapi-tools package; on OpenSuse textitpapi and textitpapi-dev. 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

To further customize the tracked counters, modify the XML to suit your needs. To find the available PAPI counters on a given computer issue the command *papi_avail -a*. For more information about Extrae's XML configuration refer to https://www.bsc.es/computer-sciences/performance-tools/trace-generation/extrae/extrae-user-guide.

6.6 Paraver: configurations

Table 16, Table 17 and Table 18 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.

Table 16: General paraver configurations for COMPSs Applications

Configuration File Name	Description
2dp_runtime_state.cfg	2D plot of runtime state
2dp_tasks.cfg	2D plot of tasks duration
3dh_duration_runtime.cfg	3D Histogram of runtime execution
3dh_duration_tasks.cfg	3D Histogram of tasks duration
compss_runtime.cfg	Shows COMPSs Runtime events (master and workers)
compss_tasks_and_runtime.cfg	Shows COMPSs Runtime events (master and workers) and tasks execution
compss_tasks.cfg	Shows tasks execution
compss_tasks_numbers.cfg	Shows tasks execution by task id
Interval_between_runtime.cfg	Interval between runtime events
thread_cpu.cfg	Shows the initial executing CPU.

Table 17: Available paraver configurations for Python events of COMPSs Applications

Configuration File Name	Description
3dh_events_inside_task.cfg	3D Histogram of python events
3dh_events_inside_tasks.cfg	Events showing python information such as user function execution time, modules imports, or serializations.

Table 18: Available paraver configurations for COMPSs Applications

Configuration File Name	Description
sr_bandwith.cfg	Send/Receive bandwith view for each node
send_bandwith.cfg	Send bandwith view for each node
receive_bandwith.cfg	Receive bandwith view for each node
process_bandwith.cfg	Send/Receive bandwith table for each node
compss_tasks_scheduling_transfers.cfg	Task's transfers requests for scheduling (gradient of tasks ID)
compss_tasksID_transfers.cfg	Task's transfers request for each task (task with its IDs are also shown)
compss_data_transfers.cfg	Shows data transfers for each task's parameter
communication_matrix.cfg	Table view of communications between each node

6.7 User Events in Python

Users can emit custom events inside their python **tasks**. Thanks to the fact that python isn't a compiled language, users can emit events inside their own tasks using the available extrae instrumentation object because it is already imported.

To emit an event first `import pyextrae` just use the call `pyextrae.event(type, id)` or `pyextrae.eventand counters (type, id)` if you also want to emit PAPI hardware counters. It is recommended to use a type number higher than 8000050 in order to avoid type's conflicts. This events will appear automatically on the generated trace. In order to visualize them, take, for example, `compss_runtime.cfg` and go to `Window Properties -> Filter -> Events -> Event Type` and change the value labeled *Types* for your custom events type. If you want to name the events, you will need to manually add them to the .pcf 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.q

More information and examples of common python usage can be found under the default directory `/opt/COMPSs/Dependencies/extrae/share/examples/PYTHON`.

CHAPTER 7

Sample Applications

7.1 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.

7.1.1 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.

```
// 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
```

(continues on next page)

(continued from previous page)

```
        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
```

(continues on next page)

(continued from previous page)

```
----------------- 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
[(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
```

7.1.2 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
```

(continues on next page)

(continued from previous page)

```
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");
        }
        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);
}
```

```
// 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
```

(continues on next page)

(continued from previous page)

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

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

7.1.3 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.

```
// 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 38.

```
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
```

7.1.4 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:

1. **matmul.objects.Matmul** Matrix stored by means of objects
2. **matmul.files.Matmul** Matrix stored in files

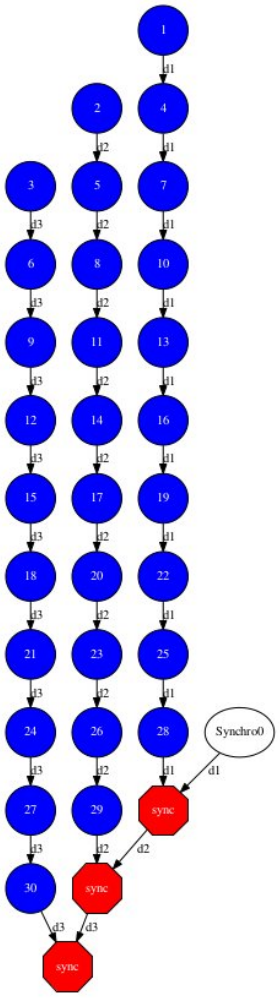


Figure 38: Java increment tasks graph

3. **matmul.arrays.Matmul** Matrix represented by an array

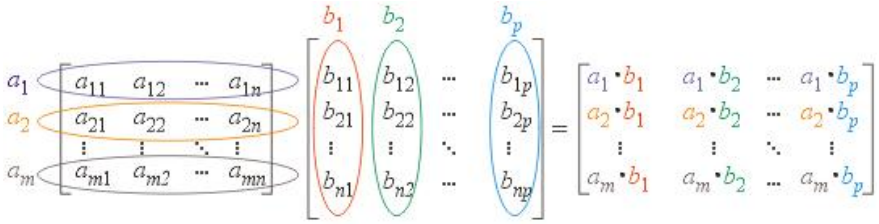


Figure 39: Matrix multiplication

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++) {
                        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
```

(continues on next page)

(continued from previous page)

```
[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

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

7.1.5 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.

The matrix is divided into N x N blocks on where 4 types of operations will be applied modifying the blocks: **lu0**,

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}.$$

Figure 40: Sparse LU decomposition

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
```

(continues on next page)

(continued from previous page)

```
[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

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

7.1.6 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.

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>
```

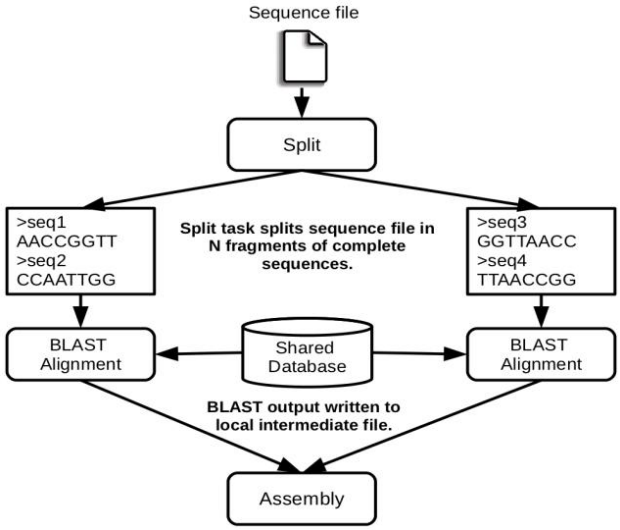


Figure 41: The COMPSs Blast workflow

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
```

7.2 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.

7.2.1 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:

```
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()

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 " + initialValue

    # Execute increment
    increment(fileName)

    # Write new value
    fis = compss_open(fileName, 'r+')
    finalValue = fis.read()
    fis.close()
    print "Final counter value is " + finalValue

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

The simple application can be executed by invoking the runcompss command with the `--lang=python` flag. The following lines provide an example of its execution.

```
compss@bsc:~$ cd ~/tutorial_apps/python/simple/
compss@bsc:~/tutorial_apps/python/simple$ runcompss --lang=python ~/tutorial_apps/
↪python/simple/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

----------------- Executing simple.py --------------------------

WARNING: COMPSs Properties file is null. Setting default values
[(639)    API]  -  Starting COMPSs Runtime v<version>
Initial counter value is 1
Final counter value is 2
[(6230)    API]  -  Execution Finished

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

7.2.2 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.

```
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()

def main_program():
    # Check and get parameters
    if len(sys.argv) != 5:
        exit(-1)
    N = int(sys.argv[1])
    counter1 = int(sys.argv[2])
    counter2 = int(sys.argv[3])
    counter3 = int(sys.argv[4])

    # Initialize counter files
    initializeCounters(counter1, counter2, counter3)
    print "Initial counter values:"
    printCounterValues()
```

(continues on next page)

(continued from previous page)

```
    # Execute increment
    for i in range(N):
        increment(FILENAME1)
        increment(FILENAME2)
        increment(FILENAME3)

    # Write final counters state (sync)
    print "Final counter values:"
    printCounterValues()

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

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 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 --lang=python -g ~/tutorial_
↪apps/python/increment/increment.py 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.py --------------------------

WARNING: COMPSs Properties file is null. Setting default values
[(670)    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
[(7390)    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 42.

```
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
```

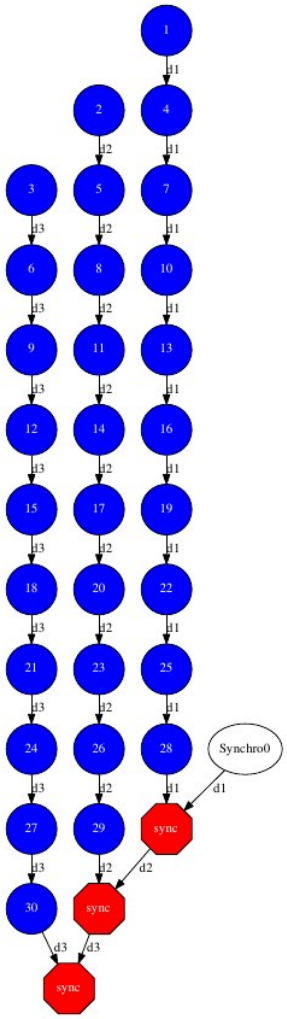


Figure 42: Python increment tasks graph

7.3 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.

7.3.1 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
    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
```

(continues on next page)

(continued from previous page)

```
    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;
}
```

```
//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 {
    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

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

7.3.2 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];
```

(continues on next page)

(continued from previous page)

```
    // 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
```

(continues on next page)

(continued from previous page)

```
[(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
[   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
```

(continues on next page)

(continued from previous page)

```
[   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
```

(continues on next page)

(continued from previous page)

```
[   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
```

(continues on next page)

(continued from previous page)

```
[   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
```

(continues on next page)

(continued from previous page)

```
[   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 43.

```
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
```

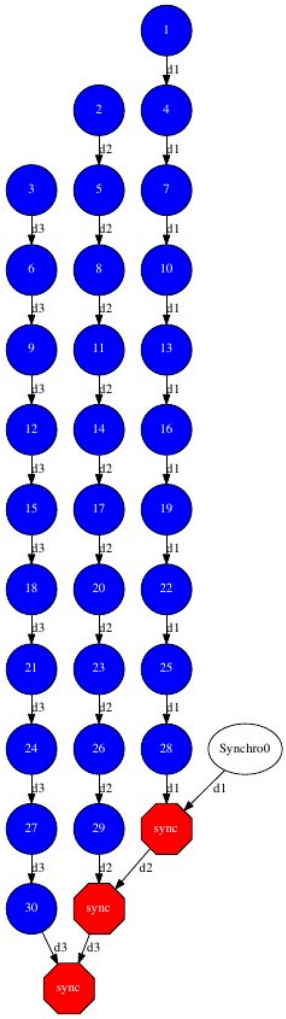


Figure 43: C increment tasks graph

CHAPTER 8

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.

8.1 Storage Integration

COMPSs relies on a Storage API to enable the interation with persistent storage frameworks (Figure 44), which is composed by two main modules: *Storage Object Interface* (SOI) and *Storage Runtime Interface* (SRI)

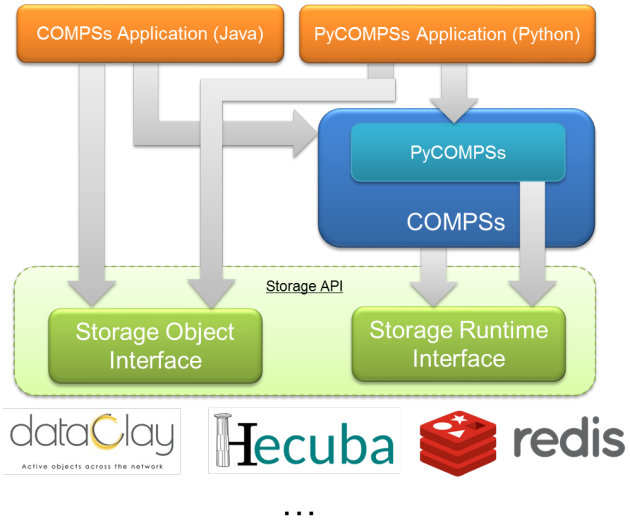


Figure 44: COMPSs with persistent storage architecture

Any COMPSs application aimed at using a persistent storage framework has to include calls to the SOI, and relies on COMPSs, which interacts with the persistent storage framework through the SRI. 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`.

Currently, there exists storage interfaces for dataClay, Hecuba and Redis. They are thoroughly described from the developer and user point of view in *COMPSs + dataClay*, *COMPSs + Hecuba* and *COMPSs + Redis* Sections.

The interface is open to any other storage framework by implementing the required functionalities described in *Implement your own Storage interface for COMPSs*.

8.2 COMPSs + dataClay

Warning: To Be Done

8.2.1 Dependencies

dataClay

Other dependencies

8.2.2 Enabling COMPSs applications with dataClay

Java

Python

C/C++

Warning: Unsupported language

8.2.3 Executing a COMPSs application with dataClay

Launching using an existing dataClay deployment

Launching on queue system based environments

8.3 COMPSs + Hecuba

Warning: To Be Done

8.3.1 Dependencies

Hecuba

Other dependencies

8.3.2 Enabling COMPSs applications with Hecuba

Java

Warning: Unsupported language

Python

C/C++

Warning: Unsupported language

8.3.3 Executing a COMPSs application with Hecuba

Launching using an existing Hecuba deployment

Launching on queue system based environments

8.4 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.

8.4.1 Dependencies

The required dependencies are:

- *Redis Server*
- *Redis Cluster script*
- *COMPSs-Redis Bundle*

[1] https://redis.io/topics/cluster-tutorial

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.

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.

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.

8.4.2 Enabling COMPSs applications with Redis

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:

```
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 19.

Table 19: 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.

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:

```
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:

```
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:

```
@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:

```
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:

```
@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.

C/C++

Warning: Unsupported language

8.4.3 Executing a COMPSs application with Redis

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.

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=${path_to_the_bundle_folder}` This must point to the root of the COMPSs-Redis bundle.
2. `--storage-props=${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.

[2] https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

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.

8.5 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.

8.5.1 Generic Storage Object Interface

SCO object definition 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 20: 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).

8.5.2 Generic Storage Runtime Interfaces

Java API shows the functions that must exist in the storage runtime interface, that enables the COMPSs runtime to interact with the storage framework.

Table 21: 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	Object	Retrieve the result of the	public static Object getResult(CallbackEvent event) throws StorageEx-

This functions enable the COMPSs runtime to keep the data consistency through the distributed execution.

In addition, *Python API* 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 22: 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

8.5.3 Storage Interface usage

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.

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_` flag, where the SOI for Python must exist.