COMP Superscalar

Installation and Administration Manual

Version:

December 11, 2018

This manual only provides information about how to install and configure COMPSs. Specifically, it details the installation process for Debian based distributions and for RedHat based distributions, and the steps to configure COMPSs properly.

If you are not wondering to install COMPSs please consider using our already prepared *Virtual Machine* available at our webpage: `http://compss.bsc.es` .

For further information about the application execution please refer to the *COMPSs User Manual: Application execution guide* available at `http://compss.bsc.es/releases/compss/latest/docs/COMPSs_User_Manual_App_Exec.pdf` .

For further information about the application development please refer to the *COMPSs User Manual: Application development guide* available at `http://compss.bsc.es/` .

For full COMPSs application examples (codes, execution commands, results, logs, etc.) please refer to the *COMPSs Sample Applications* available at `http://compss.bsc.es/releases/compss/latest/docs/COMPSs_User_Manual_App_Development.pdf` .

Contents

List of Figures

List of Tables

COMP Superscalar (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.

Packages description

Packages structure

Despite the fact that we recommend users to install the complete COMPSs Framework, we have built different packages to allow users customize as maximum as possible their installation. Figure 1 illustrates the COMPSs packaging structure and its internal dependencies.

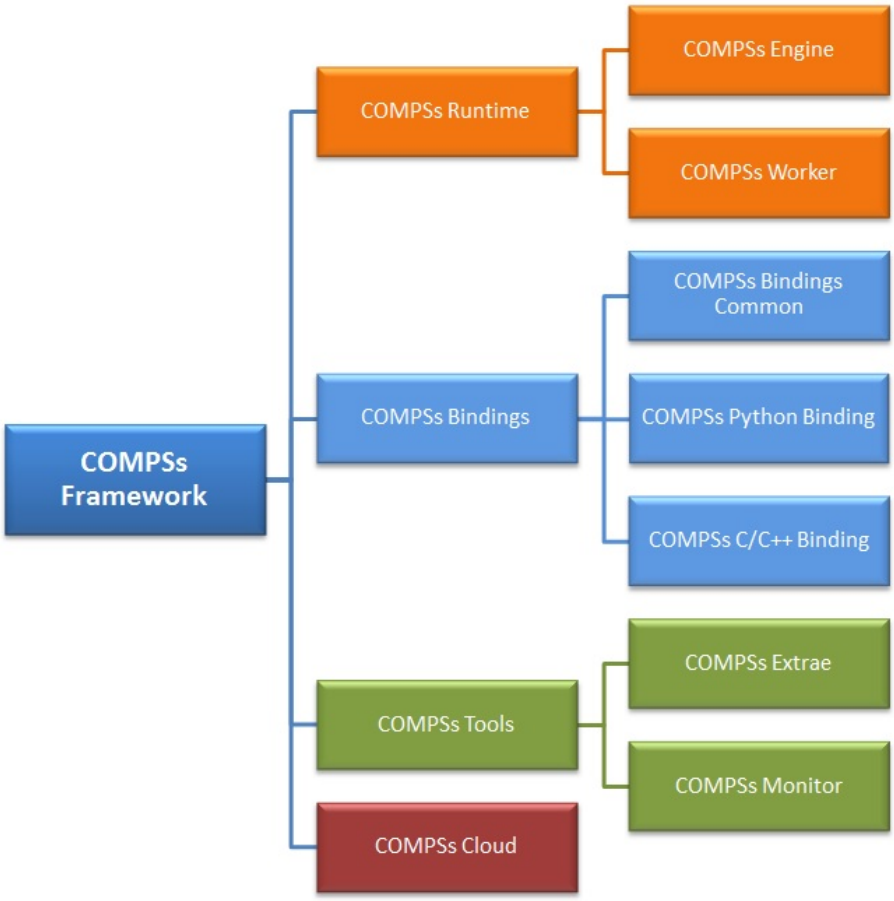

Figure 1: COMPSs packaging structure

Packages Dependencies

Next we provide a list of dependencies for each 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.).

COMPSs Framework	compss-runtime, compss-bindings, compss-tools, compss-cloud
COMPSs Runtime	compss-engine, compss-worker
COMPSs Engine	openjdk-8-jre, graphviz, xdg-utils
COMPSs Worker	openjdk-8-jre
COMPSs Bindings	compss-bindings-common, compss-c-binding, compss-python-binding
COMPSs Bindings Common	compss-engine, libtool, automake, build-essential
COMPSs Python Binding	compss-bindings-common, python (>= 2.7), libpython2.7
COMPSs C/C++ Binding	compss-binding-common, libboost-all-dev, libxml2-dev
COMPSs Tools	compss-extrae, compss-monitor
COMPSs Extrae	compss-engine, libxml2 (>= 2.5), libxml2-dev (>= 2.5), gfortran, papi
COMPSs Monitor	compss-engine
COMPSs Cloud	compss-engine

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`.

Debian-based distributions

Prerequisites

The commands described on the following sections require root privileges and Internet connection.

Once the installation process is finished, please log out and back in again to complete the installation.

Package Repository

To add the package repository you can easily download our predefined lists by executing the following command:

```
ubuntu x86_64 :
   wget http://compss.bsc.es/releases/repofiles/repo_deb_ubuntu_x86-64.list -O /etc/apt/sources.list.d/
     compss-framework_x86-64.list

ubuntu noarch :
   wget http://compss.bsc.es/releases/repofiles/repo_deb_ubuntu_noarch.list -O /etc/apt/sources.list.d/
     compss-framework_noarch.list

debian x86_64 :
   wget http://compss.bsc.es/releases/repofiles/repo_deb_debian_x86-64.list -O /etc/apt/sources.list.d/
     compss-framework_x86-64.list

debian noarch :
   wget http://compss.bsc.es/releases/repofiles/repo_deb_debian_noarch.list -O /etc/apt/sources.list.d/
     compss-framework_noarch.list
```

Next you need to add the repository key by executing:

```
wget -qO - http://compss.bsc.es/repo/debs/deb-gpg-bsc-grid.pub.key | apt-key add -
```

And finally, refresh the apt-get repositories:

```
apt-get update
```

Installation

This section describes how to install all the available COMPSs packages. If you are willing to have a full COMPSs installation just follow the COMPSs Framework instructions and skip directly to next section.

- **COMPSs Framework**
  Contains all the COMPSs functionalities including the Runtime, all the bindings, all the tools and the cloud connectors.
  To install this package please run:

```
apt-get install compss-framework
```

- **COMPSs Runtime**
  Contains the COMPSs runtime to support the native functionalities. Install this package if you only need to support Java applications.
  To install this package please run:

```
apt-get install compss-runtime
```

  This package is composed of two sub-packages:

  - **COMPSs Engine**
    Contains the COMPSs Engine, essential to run COMPSs applications as master.
    To install this package please run:

```
apt-get install compss-engine
```

  - **COMPSs Worker**
    Contains the minimum installation required to run a machine as a COMPSs worker.
    To install this package please run:

```
apt-get install compss-worker
```

- **COMPSs Bindings**
  Contains all the required bindings for C/C++ and Python applications.
  To install this package please run:

```
apt-get install compss-bindings
```

  This package is composed of three sub-packages:

  - **COMPSs Bindings Common**
    Contains the API required for the communication between any binding and the COMPSs Runtime. It is necessary for any binding installation.
    To install this package please run:

```
apt-get install compss-bindings-common
```

- **COMPSs C/C++ Binding**
  Contains the C/C++ Binding
  To install this package please run:

```
apt-get install compss-c-binding
```

- **COMPSs Python Binding**
  Contains the Python Binding
  To install this package please run:

```
apt-get install compss-python-binding
```

- **COMPSs Tools**
  Contains all the COMPSs Tools.
  To install this package please run:

```
apt-get install compss-tools
```

  This package is composed of three sub-packages:

  - **COMPSs Extrae**
    Contains the COMPSs Extrae tool needed to generate and process application traces.
    To install this package please run:

```
apt-get install compss-extrae
```

  - **COMPSs Monitor**
    Contains the COMPSs Monitor tool needed to monitor the application execution.
    To install this package please run:

```
apt-get install compss-monitor
```

- **COMPSs Cloud**
  Contains all the COMPSs Connectors needed to interact with the Cloud.
  To install this package please run:

```
apt-get install compss-cloud
```

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 Section 8 for a detailed description of the additional configuration.

Suse-based distributions (zypper)

Prerequisites

The commands described on the following sections require root privileges and Internet connection.

Once the installation process is finished, please log out and back in again to complete the installation.

Package Repository

To add the package repository you can easily download our predefined lists by executing the following command:

```
x86_64      :  zypper addrepo -f http://compss.bsc.es/repo/rpms/stable/suse/x86_64 compss
noarch      :  zypper addrepo -f http://compss.bsc.es/repo/rpms/stable/suse/noarch compss
```

And finally, refresh the repositories:

```
zypper refresh
```

Installation

This section describes how to install all the available COMPSs packages. If you are willing to have a full COMPSs installation just follow the COMPSs Framework instructions and skip directly to next section.

- **COMPSs Framework**
  Contains the all COMPSs functionalities including the Runtime, all the bindings, all the tools and the cloud connectors.
  To install this package please run:

  ```
  zypper install compss-framework
  ```

- **COMPSs Runtime**
  Contains the COMPSs runtime to support the native functionalities. Install this package if you only need to support Java applications.
  To install this package please run:

  ```
  zypper install compss-runtime
  ```

This package is composed of two sub-packages:

- **COMPSs Engine**
  Contains the COMPSs Engine, essential to run COMPSs applications as master.
  To install this package please run:

```
zypper install compss-engine
```

- **COMPSs Worker**
  Contains the minimum installation required to run a machine as a COMPSs worker.
  To install this package please run:

```
zypper install compss-worker
```

- **COMPSs Bindings**
  Contains all the required bindings for C/C++ and Python applications.
  To install this package please run:

```
zypper install compss-bindings
```

This package is composed of three sub-packages:

- **COMPSs Bindings Common**
  Contains the API required for the communication between any binding and the COMPSs Runtime. It is necessary for any binding installation.
  To install this package please run:

```
zypper install compss-bindings-common
```

- **COMPSs C/C++ Binding**
  Contains the C/C++ Binding
  To install this package please run:

```
zypper install compss-c-binding
```

  - **COMPSs Python Binding**
    Contains the Python Binding
    To install this package please run:

```
zypper install compss-python-binding
```

- **COMPSs Tools**
  Contains all the COMPSs Tools.
  To install this package please run:

```
zypper install compss-tools
```

  This package is composed of three sub-packages:

  - **COMPSs Extrae**
    Contains the COMPSs Extrae tool needed to generate and process application traces.
    To install this package please run:

```
zypper install compss-extrae
```

  - **COMPSs Monitor**
    Contains the COMPSs Monitor tool needed to monitor the application execution.
    To install this package please run:

```
zypper install compss-monitor
```

- **COMPSs Cloud**
  Contains all the COMPSs Connectors needed to interact with the Cloud.
  To install this package please run:

```
zypper install compss-cloud
```

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 Section 8 for a detailed description of the additional configuration.

RedHat-based distributions (yum)

Prerequisites

The commands described on the following sections require root privileges and Internet connection.

Once the installation process is finished, please log out and back in again to complete the installation.

Package Repository

To add the package repository you can easily download our predefined lists by executing the following command:

```
x86_64 :
   wget http://compss.bsc.es/releases/repofiles/repo_rpm_centos_x86-64.repo -O /etc/yum.repos.d/compss-
     framework_x86-64.repo
```

Installation

This section describes how to install all the available COMPSs packages. If you are willing to have a full COMPSs installation just follow the COMPSs Framework instructions and skip directly to next section.

- **COMPSs Framework**
  Contains the all COMPSs functionalities including the Runtime, all the bindings, all the tools and the cloud connectors.
  To install this package please run:

  ```
  yum install compss-framework
  ```

- **COMPSs Runtime**
  Contains the COMPSs runtime to support the native functionalities. Install this package if you only need to support Java applications.
  To install this package please run:

  ```
  yum install compss-runtime
  ```

  This package is composed of two sub-packages:

- **COMPSs Engine**
  Contains the COMPSs Engine, essential to run COMPSs applications as master.
  To install this package please run:

```
yum install compss-engine
```

- **COMPSs Worker**
  Contains the minimum installation required to run a machine as a COMPSs worker.
  To install this package please run:

```
yum install compss-worker
```

- **COMPSs Bindings**
  Contains all the required bindings for C/C++ and Python applications.
  To install this package please run:

```
yum install compss-bindings
```

  This package is composed of three sub-packages:

  - **COMPSs Bindings Common**
    Contains the API required for the communication between any binding and the COMPSs Runtime. It is necessary for any binding installation.
    To install this package please run:

```
yum install compss-bindings-common
```

  - **COMPSs C/C++ Binding**
    Contains the C/C++ Binding
    To install this package please run:

```
yum install compss-c-binding
```

  - **COMPSs Python Binding**
    Contains the Python Binding
    To install this package please run:

```
yum install compss-python-binding
```

- **COMPSs Tools**
  Contains all the COMPSs Tools.
  To install this package please run:

```
yum install compss-tools
```

  This package is composed of three sub-packages:

  - **COMPSs Extrae**
    Contains the COMPSs Extrae tool needed to generate and process application traces.
    To install this package please run:

```
yum install compss-extrae
```

  - **COMPSs Monitor**
    Contains the COMPSs Monitor tool needed to monitor the application execution.
    To install this package please run:

```
yum install compss-monitor
```

- **COMPSs Cloud**
  Contains all the COMPSs Connectors needed to interact with the Cloud.
  To install this package please run:

```
yum install compss-cloud
```

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 Section 8 for a detailed description of the additional configuration.

Pip

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 section 2.2 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, that 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
```

   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:

```
user@machine:~> echo $JAVA_HOME
/usr/lib64/jvm/java-openjdk/
```

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.

Configuration

The steps mentioned in Section 8.1 must be done in order to have a functional COMPSs and PyCOMPSs installation.

Post installation

As mentioned in Section 6.2, it is recommended to restart the user session or virtual environment once the installation process has finished.

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.

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)

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>

# 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. Please note that if the 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.

Configuration

For queue system executions, COMPSs has a pre-build structure (see Figure 2) 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.

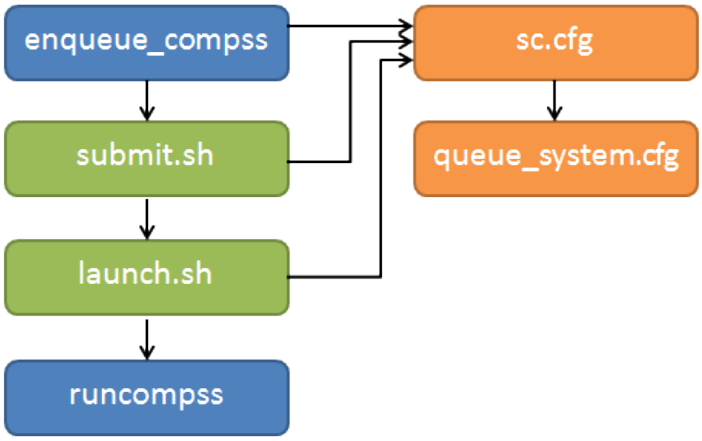

Figure 2: Structure of COMPSs queue scripts. In Blue user scripts, in Green queue scripts and in Orange system dependant scripts

To make this structure works, the administrators must define a configuration file for the queue system (under `<targetDir>/COMPSs/scripts/queues/cfgs/QUEUE/QUEUE.cfg`) and a configuration file for the specific SuperComputer parameters (under `<targetDir>/COMPSs/scripts/queues/cfgs/SC_NAME.cfg`). 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 III* SuperComputer) or to contact us at `support-compss@bsc.es` .

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


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


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

    --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

    --queue=<name>                          Queue name to submit the job. Depends on the queue
                                            system.
                                            For example (Nord3): bsc_cs | bsc_debug | debug
                                            | interactive
                                            Default: default

    --reservation=<name>                    Reservation to use when submitting the job.
                                            Default: disabled

    --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
```

```
Launch configuration:
  --cpus_per_node=<int>                    Available CPU computing units on each node
                                           Default: 16

  --gpus_per_node=<int>                    Available GPU computing units on each node
                                           Default: 0

  --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: 0

  --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_tasks is specified.
                                           Default: disabled

  --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: Empty

  --container_image=<path>                 Runs the application by means of a singularity
                                           container image
                                           Default: Empty

  --container_compss_path=<path>           Path where compss is installed in the Singularity
                                           container image
                                           Default: /opt/COMPSs


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 | 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_conf=<path>                 Path to the storage configuration file
                                        Default: None

  --project=<path>                      Path to the project XML file
                                        Default: default_project.xml

  --resources=<path>                    Path to the resources XML file
                                        Default: default_resources.xml

  --lang=<name>                         Language of the application (java/c/python)
                                        Default: Inferred if 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
                                        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
                                        Default: es.bsc.compss.connectors.DefaultSSHConnector

  --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.loadBalancingScheduler
                                        .LoadBalancingScheduler

  --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: User home

    --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 (the folder
                                            MUST exist and 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: None

    --master_port=<int>                     Port to run the COMPSs master communications.
                                            Only for NIO adaptor
                                            Default: [43000,44000]

    --jvm_master_opts="<string>"            Extra options for the COMPSs Master JVM. Each
                                            option separed by "," and without blank spaces
                                            (Notice the quotes)
                                            Default: Empty

    --jvm_workers_opts="<string>"           Extra options for the COMPSs Workers JVMs. Each
                                            option separed by "," and without blank spaces
                                            (Notice the quotes)
                                            Default: -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

    --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

    --pythonpath=<path>                     Additional folders or paths to add to the PYTHONPATH
                                            Default: User home

    --PyObject_serialize=<bool>             Only for Python Binding. Enable the object
                                            serialization to string when possible (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.

Additional Configuration

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 **in 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 3, 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
```

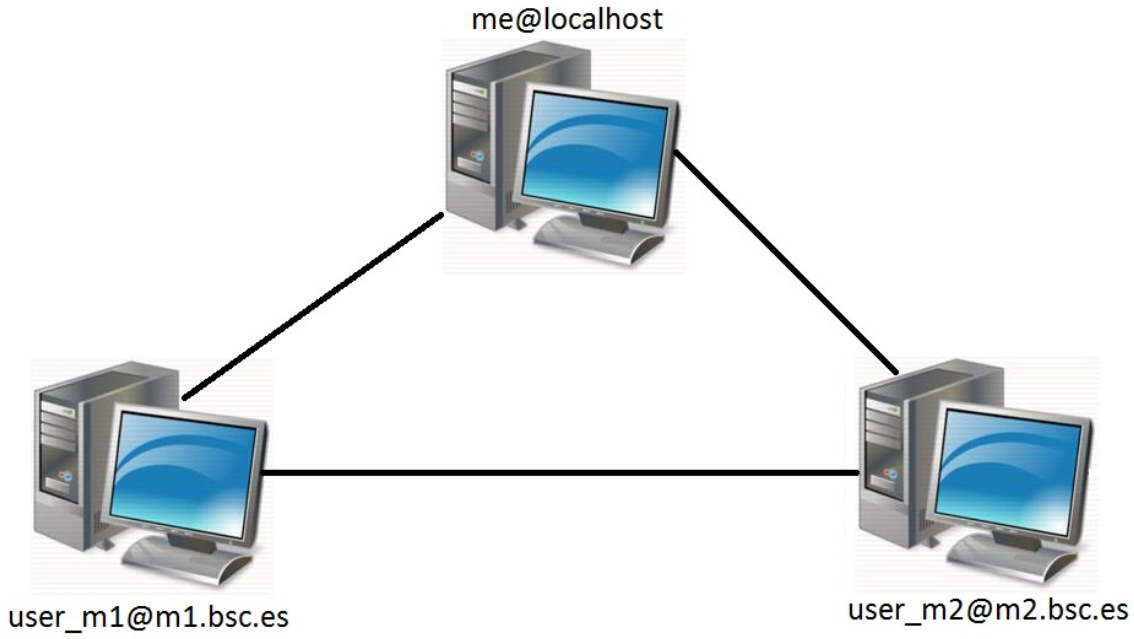

Figure 3: Cluster example

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`

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.

Resources file

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

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

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

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

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 (for more information check the *COMPSs Installation Manual* available at our website `http://compss.bsc.es`). 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\>

    <!--Description for a physical node-->
    <ComputeNode Name="localhost">
        <InstallDir>/opt/COMPSs/</InstallDir>
        <WorkingDir>/tmp/Worker/</WorkingDir>
        <Application>
            <AppDir>/home/user/apps/</AppDir>
            <LibraryPath>/usr/lib/</LibraryPath>
            <Classpath>/home/user/apps/jar/example.jar</Classpath>
            <Pythonpath>/home/user/apps/</Pythonpath>
        </Application>
        <LimitOfTasks>4</LimitOfTasks>
        <Adaptors>
            <Adaptor Name="es.bsc.compss.nio.master.NIOAdaptor">
                <SubmissionSystem>
```

```
                    <Interactive/>
                </SubmissionSystem>
                <Ports>
                    <MinPort>43001</MinPort>
                    <MaxPort>43002</MaxPort>
                </Ports>
                <User>user</User>
            </Adaptor>
        </Adaptors>
    </ComputeNode>
</Project>
```

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.

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>
    </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>
```

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 following the *Installation manual* available at `http://compss.bsc.es` .
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>
    </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>
                    </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>
```

```
                </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:

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

Table 1: Connector supported properties in the `project.xml` file.

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

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

Table 2: Properties supported by any SSH based connector in the `project.xml` file.

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,

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

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 3 contains the rOCCI specific properties that must be defined under the `Provider` tag in the `project.xml` file and Table 3 contains the specific properties that must be defined under the `Instance` tag.

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 3: rOCCI extensions in the `project.xml` 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

Table 4: Configuration of the `<provider>.xml` templates file.

Cloud connectors: JClouds

The JClouds connector is based on the JClouds API version *1.9.1*. Table 5 shows the extra available options under the *Properties* tag that are used by this connector.

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

Table 5: JClouds extensions in the `project.xml` file.

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 6.

Name	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.

Table 6: Mesos connector options in `project.xml` file.

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

COMPSs Removal

How to uninstall or remove COMPSs

COMPSs can be easily uninstalled via the *Linux Packaging Tools* by running the following commands:

```
Debian: apt-get remove compss-framework
RedHat (zypper): zypper remove compss-framework
RedHat (yum): yum remove compss-framework
```

Notice that some of the COMPSs packages are meta-packages and, thus, you will need to manually uninstall all the COMPSs packages or use the autoremove tools:

```
Debian: apt-get autoremove
RedHat (zypper): zypper remove --clean-deps compss-framework
RedHat (yum): yum autoremove
```

In *Debian* based distributions uninstalling COMPSs will not erase your configuration files. If you are willing to completely remove COMPSs please remember to use the *purge* option:

```
Debian: apt-get purge compss-framework
```

How to clean repositories

During the installation process you may have added the COMPSs repository. If you want to clean your respository list please erase the compss list by executing the following commands:

```
Debian:
  $ rm -f /etc/apt/sources.list.d/compss-framework_*.list
  $ apt-get update

RedHat (zypper):
  $ zypper removerepo compss
  $ zypper refresh

RedHat (yum):
  $ rm -f /etc/yum.repos.d/compss-framework_*.repo
```

Please find more details on the COMPSs framework at

http://compss.bsc.es