



COMP SUPERSCALAR

COMPSS at BSC

MareNostrum 3 Manual

VERSION: 1.4.RC07

July 11, 2016



**Barcelona
Supercomputing
Center**

Centro Nacional de Supercomputación

This manual only provides information about the COMPSs usage at MareNostrum. Specifically, it details the available COMPSs modules, how to load them and how to create and track COMPSs jobs.

If you want to install COMPSs on your local machine please refer to the *COMPSs Installation Manual* available at our webpage <http://compss.bsc.es>.

For further information about the application's execution please refer to the *COMPSs User Manual: Application execution guide* available at <http://compss.bsc.es> .

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

For full COMPSs example application (codes, execution commands, results, logs, etc.) please refer to the *COMPSs Sample Applications* available at <http://compss.bsc.es/>

Contents

| | | |
|----------|----------------------------------|----------|
| 1 | COMP Superscalar (COMPSs) | 1 |
| 2 | COMPSs Modules | 2 |
| 2.1 | Available modules | 2 |
| 2.2 | Configuration | 2 |
| 3 | COMPSs Jobs | 4 |
| 3.1 | Submitting COMPSs jobs | 4 |
| 3.2 | Tracking COMPSs jobs | 6 |
| 4 | Enabling COMPSs Monitor | 7 |
| 4.1 | Configuration | 7 |
| 4.2 | Execution | 7 |

1 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.

2 COMPSs Modules

2.1 Available modules

COMPSs is configured in MareNostrum (MN3) as a Linux Module. Type *module available COMPSs* to list the available COMPSs modules through Linux Module configuration and *module load COMPSs/ < version >* to load it.

```
$ module available COMPSs
----- /apps/modules/modulefiles/tools -----
COMPSs/0.0
COMPSs/0.1
COMPSs/0.2_Nested
COMPSs/1.1.2_gpfs
COMPSs/1.1.2_scratch
COMPSs/1.2
COMPSs/1.3

COMPSs/release(default)
COMPSs/trunk

$ module load COMPSs/release
load java/1.7.0u55 (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, IT_HOME)
```

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

```
$ runcompss --version
COMPSs version 1.3
```

2.2 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** recommend to change them. Before running any COMPSs job please check your environment and, if needed, comment out any line inside the *.bashrc* file loading 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@mn1.bsc.es
load java/1.7.0u55 (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/trunk (PATH, MANPATH, IT_HOME)

$ runcompss --version
COMPSs version 1.3
```

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

3 COMPSs Jobs

3.1 Submitting COMPSs jobs

COMPSs jobs can be easily submitted 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 --help
Usage: /apps/COMPSs/1.3/Runtime/scripts/user/enqueue_compss
       [queue_system_options] [COMPSs_options]
       application_name application_arguments

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

Queue system 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_system=<name>    Queue system to use: lsf | pbs | slurm
                           Default: lsf

  - -queue=<name>           Queue name to submit the job.
                           Depends on the queue system.
                           For example (MN3): bsc_cs | bsc_debug
                           | debug | interactive
                           Default: default

  - -job_dependency=<jobID> Postpone job execution until the job
                           dependency has ended.
                           Default: None

  - -tasks_per_node=<int>   Maximum number of simultaneous
                           tasks running on a node
                           Default: 16

  - -master_working_dir=<path> Working directory of the application
                           Default: .

  - -worker_working_dir=<name> Worker directory. Use: scratch | gpfs
                           Default: scratch

  - -tasks_in_master=<int>  Maximum number of tasks that the master
                           node can run as worker. Cannot exceed
                           tasks_per_node.
                           Default: 0

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

Runcompss delegated parameters:

Tools enablers:

- `-graph=<bool>`, - `-graph`, -`g` Generation of the `complete` graph (`true/false`)
When no value is provided it is `set` to `true`
Default: `false`
- `-tracing=<bool>`, - `-tracing`, -`t` Generation of traces (`true/false`)
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

Runtime configuration options:

- `-project=<path>` Path to the project XML file
Default: `/gpfs/apps/MN3/COMPSS/1.3/Runtime/configuration/xml/projects/project.xml`
- `-resources=<path>` Path to the resources XML file
Default: `/gpfs/apps/MN3/COMPSS/1.3/Runtime/configuration/xml/resources/resources.xml`
- `-lang=<name>` Language of the application (`java/c/python`)
Default: `java`
- `-log_level=<level>`, - `-debug`, -`d` Set the debug level: `off` | `info` | `debug`
Default: `off`

Advanced options:

- `-comm=<path>` Class that implements the adaptor
`for` communications
Supported adaptors:
`integratedtoolkit.nio.master.NIOAdaptor` |
`integratedtoolkit.gat.master.GATAaptor`
Default:
 `integratedtoolkit.nio.master.NIOAdaptor`
- `-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
- `-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
- `-uuid=<int>` Preset an application UUID
Default: Automatic random generation
- `-PyObject_conversion=<bool>` Only `for` Python Binding. Enable the object
conversion 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.

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 \  
--queue_system=lsf \  
--tasks_per_node=16 \  
--master_working_dir=. \  
--worker_working_dir=gpfs \  
--lang=python \  
--log_level=debug \  
<APP> <APP_PARAMETERS>  
  
Num Nodes:      3  
Tasks per Node: 16  
Tasks in Master:0  
Master WD:      .  
Worker WD:      gpfs  
Exec-Time:      00:15  
Network:        default  
Library Path:   .  
To COMPSs:      --lang=python --log_level=debug <APP> <APP_PARAMETERS>  
Temp submit script is: /scratch/tmp/tmp.YPQKths559  
  
$ cat /scratch/tmp/tmp.YPQKths559  
#!/bin/bash  
#  
#BSUB -cwd .  
#BSUB -oo compss_3_%J.out  
#BSUB -eo compss_3_%J.err  
#BSUB -n 3  
#BSUB -R"span[ptile=1]"  
#BSUB -J COMPSs  
#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 *COMPSs User Manual: Application Execution* available at our webpage <http://compss.bsc.es> .

4 Enabling COMPSs Monitor

4.1 Configuration

As MareNostrum 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 the *COMPSs Installation Manual* available at our webpage <http://compss.bsc.es> . 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 (*MN_USER* stands for your MareNostrum user and the *TARGET_LOCAL_FOLDER* to the local folder where you wish to deploy the MareNostrum files):

```
compss@bsc:~$ scp $HOME/.ssh/id_dsa.pub ${MN_USER}@mn1.bsc.es:~/id_dsa_local.pub
compss@bsc:~$ ssh MN_USER@mn1.bsc.es
"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
MN_USER@mn1.bsc.es:~/COMPSs
TARGET_LOCAL_FOLDER/.COMPSs
```

Whenever you wish to unmount the sshfs directory please run:

```
compss@bsc:~$ sudo umount TARGET_LOCAL_FOLDER/.COMPSs
```

4.2 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 information please check the *COMPSs User Manual: Application Execution* available at our webpage <http://compss.bsc.es>.

Figure 1 illustrates how to login and Figure 2 shows the COMPSs Monitor main page for a MareNostrum application.



Figure 1: COMPSs Monitor login for MareNostrum

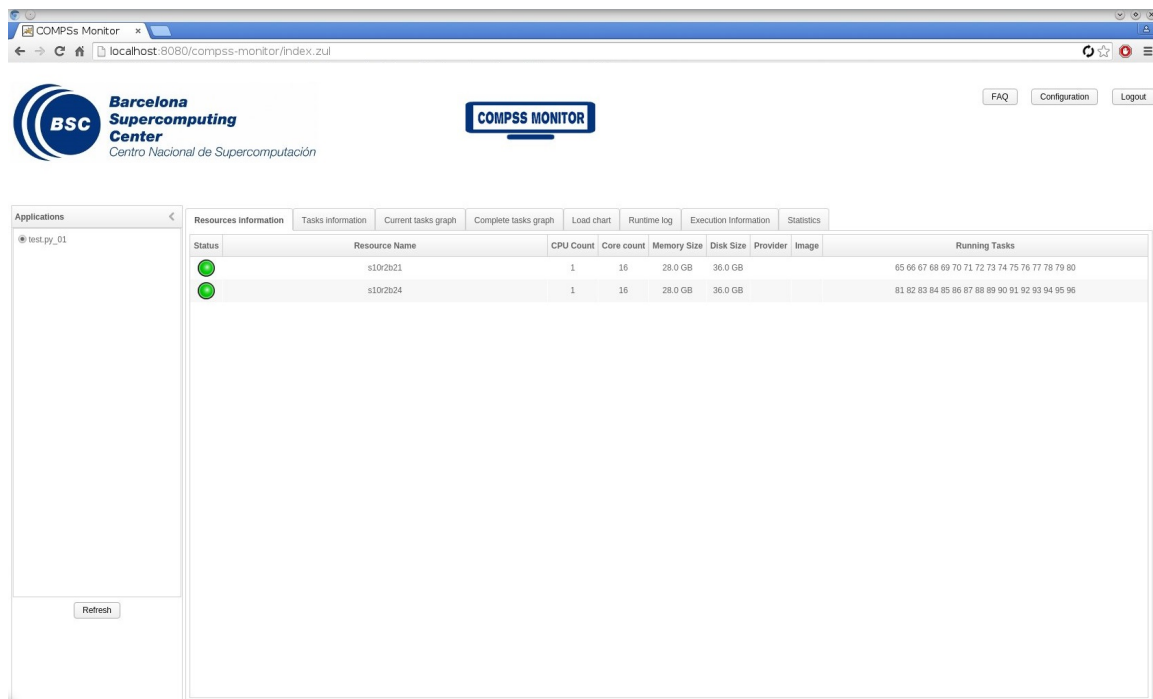


Figure 2: COMPSs Monitor main page for a test application at MareNostrum

Please find more details on the COMPSs framework at
`http://compss.bsc.es`