



COMP SUPERSCALAR

---

# COMPSS at BSC

MareNostrum 3 Manual

---

VERSION: 2.0

December 21, 2016



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

<b>1</b>	<b>COMP Superscalar (COMPSs)</b>	<b>1</b>
<b>2</b>	<b>COMPSs Modules</b>	<b>2</b>
2.1	Available modules . . . . .	2
2.2	Configuration . . . . .	2
<b>3</b>	<b>COMPSs Jobs</b>	<b>4</b>
3.1	Submitting COMPSs jobs . . . . .	4
3.2	Tracking COMPSs jobs . . . . .	6
<b>4</b>	<b>Enabling COMPSs Monitor</b>	<b>8</b>
4.1	Configuration . . . . .	8
4.2	Execution . . . . .	8

# 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/1.1.2_gpfs
COMPSs/1.1.2_scratch
COMPSs/1.2
COMPSs/1.3
COMPSs/1.4
COMPSs/2.0

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, IT_HOME)
```

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

```
$ runcompss --version
COMPSs version <version>
```

### 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/1.8.0u66 (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, IT_HOME)

$ runcompss --version
COMPSs version <version>
```

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 -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
  --gpus_per_node=<int>    Number of desired GPUs per node.
                           Leave this field empty if your application doesn't use GPUs.
                           Default:
  --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
  --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:
  --tasks_per_node=<int>   Maximum number of simultaneous tasks running on a node
                           Default: 16
  --node_memory=<MB>      Maximum node memory: disabled | <int> (MB)
                           Default: disabled
  --network=<name>        Communication network for transfers: default | ethernet
                           | infiniband | data.
                           Default: ethernet

  --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_tasks=<int>	Maximum number of tasks that the master node can run as worker. Cannot exceed tasks_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 separated by "," and without blank spaces (Notice the quotes) Default: ""
 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: /opt/COMPSs/Runtime/configuration/xml/projects/default_project.xml
--resources=<path>	Path to the resources XML file Default: /opt/COMPSs/Runtime/configuration/xml/resources/default_resources.xml
--lang=<name>	Language of the application (java/c/python) Default: 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=<path>	Class that implements the adaptor for communications Supported adaptors: integratedtoolkit.nio.master.NIOAdaptor   integratedtoolkit.gat.master.GATAdaptor Default: integratedtoolkit.nio.master.NIOAdaptor
--conn=<path>	Path of the connector jar/s that should be loaded.



```

--scheduler=<path>
--library_path=<path>
--classpath=<path>
--appdir=<path>
--base_log_dir=<path>
--specific_log_dir=<path>
--uuid=<int>
--master_name=<string>
--master_port=<int>
--jvm_master_opts="<string>"
--jvm_workers_opts="<string>"
--task_count=<int>
--pythonpath=<path>
--PyObject_serialize=<bool>

```

You can use multiple by splitting with ':'  
Supported connectors: compss-{CONN}-connector.jar  
(where {CONN} can be: "jclouds", "amazon", "docker",  
"one", "rocci", "vmm", etc.)

Class that implements the Scheduler for COMPSs  
Supported schedulers:  
integratedtoolkit.components.impl.TaskScheduler  
| integratedtoolkit.scheduler.readyscheduler.ReadyScheduler  
Default:  
integratedtoolkit.scheduler.readyscheduler.ReadyScheduler

Non-standard directories to search for libraries (e.g. Java  
JVM library, Python library, C binding library)  
Default: Working Directory

Path for the application classes / modules  
Default: Working Directory

Path for the application class folder.  
Default: /home/cramonco

Base directory to store COMPSs log files (a .COMPSs/ folder  
will be created inside this location)  
Default: User home

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

Preset an application UUID  
Default: Automatic random generation

Hostname of the node to run the COMPSs master  
Default:

Port to run the COMPSs master communications.  
Only for NIO adaptor  
Default: [43000,44000]

Extra options for the COMPSs Master JVM. Each option separated  
by "," and without blank spaces (Notice the quotes)  
Default:

Extra options for the COMPSs Workers JVMs. Each option separated  
by "," and without blank spaces (Notice the quotes)  
Default: -Xms1024m,-Xmx1024m,-Xmn400m

Only for C/Python Bindings. Maximum number of different  
functions/methods, invoked from the application, that  
have been selected as tasks  
Default: 50

Additional folders or paths to add to the PYTHONPATH  
Default: /home/cramonco

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.

## 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 \
--tasks_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
  --tasks_per_node=16
  --master_working_dir=.
  --worker_working_dir=gpfs
  --lang=python
  --classpath=.
  --library_path=.
  --comm=integratedtoolkit.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 track 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.

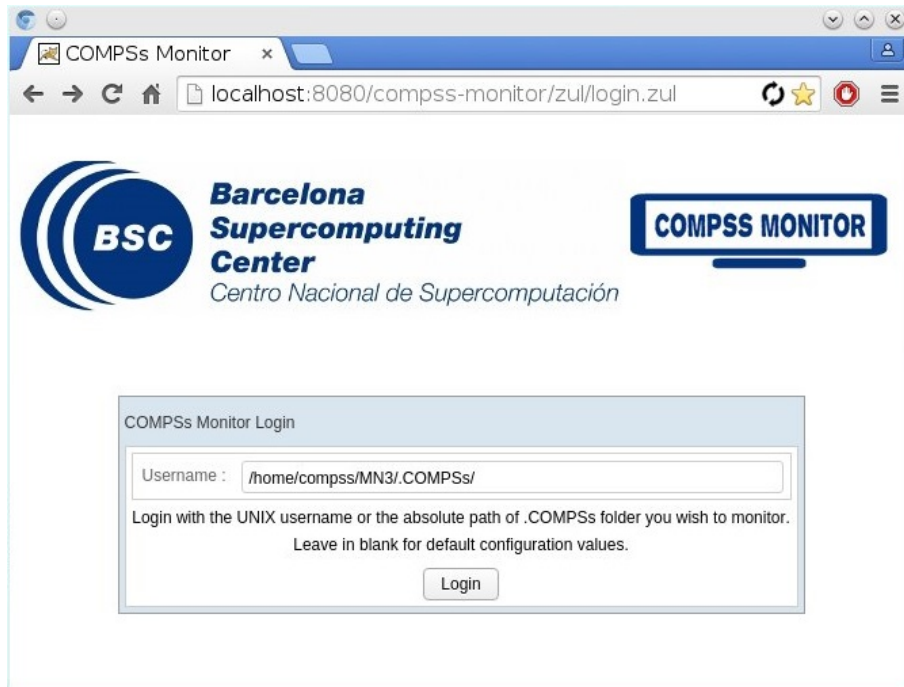


Figure 1: COMPSs Monitor login for MareNostrum

Applications <

2435288



Resources information		Tasks information	Current tasks graph	Complete tasks graph	Load chart	Runtime log	Execution information	Statistics		
Status	Resource Name	CPU Computing Units	GPU Computing Units	FPGA Computing Units	OTHER Computing Units	Memory Size	Disk Size	Provider	Image	Running Actions
	s04z883	16	-	-	-	28.0 GB	- GB	-	-	195 203 287 295 255 443 411 402 379 324 263 427 435 371 419 348
	s04z880	16	-	-	-	28.0 GB	- GB	-	-	226 210 218 275 319 299 339 307 283 259 267 331 243 251 235 292

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

Please find more details on the COMPSs framework at

<http://comps.bsc.es>