PyCOMPSs in MareNostrum4 INSTRUCTIONS

MareNostrum4 (MN4) is a supercomputer hosted at the Barcelona Supercomputing Center, composed of 3456 nodes (Intel Xeon Platinum processors), which has COMPSs installed and ready to be used.

These instructions describe how to use PyCOMPSs in MN4.

Index

Index **1**
1) How to connect to MN4 **1**
2) How to load COMPSs in MN4 **2**
3) How to execute PyCOMPSs applications in MN4 **2**
4) How to get the task dependency graph **3**
5) How to get the trace **4**
6) Sample application **5**
6.1) Get sample code 5
6.2) Launch sample code 6
6.2.1) my_workflow_multiple.py: application description 6
6.2.2) my_workflow_multiple.py: execution 6
6.2.3) my_main_gridsearch.py: tuning application parameters 7
6.2.3) my_main_gridsearch.py: execution 7
7) Further information **8**

1) How to connect to MN4

In order to connect to MN4, we use the secure-shell command from your laptop/pc as follows:

```
ssh -X nct01XXX@mn1.bsc.es
```

Where `nct01XXX` is the user provided for the tutorial.
Once executed, it will request the password, which will also be provided for the tutorial.

IMPORTANT: Be very careful introducing the password. Otherwise, your IP will be **banned** and you will not be able to connect to MN4.

2) How to load COMPSs in MN4

COMPSs is available in MN4 as a module, so it can be loaded with the following commands:

```
export COMPSS_PYTHON_VERSION=3
module load COMPSs/3.2
```

After them, COMPSs commands (e.g. “pycompss”, “enqueue_compss”, etc.) will be available.

TIP: It is suggested to add these two lines in your .bashrc, so that whenever you connect to MN4, COMPSs is available by default.

TIP2: Alternatively to add them to your .bashrc, it is also interesting to add them to any script intended to launch your PyCOMPSs application.

3) How to execute PyCOMPSs applications in MN4

In order to execute PyCOMPSs applications in MN4 it is necessary to submit them to the queuing system (SLURM). To this end, COMPSs provides a command able to deal with SLURM and ease the submission process.

For example, the following command represents a sample that submits “application.py” with a set of N parameters requesting 2 compute nodes, with a walltime of 10 minutes, using the indicated reservation, within the training queue:

```
pycompss job submit \
  --qos=training \
  --num_nodes=2 \
  --exec_time=10 \
  --reservation=<RESERVATION_NAME> \
  --lang=python \
  /path/to/application.py <param_1> <param_2> … <param_N>
```

TIP: “`pycompss job submit`” uses the latest cli interface, which replaces the “enqueue_compss” command (although keeps backwards compatibility).

TIP2: “`pycompss job submit`” has a large set of parameters that can be defined in order to configure the execution. They can be consulted by executing “`pycompss job submit -h`”.

Once the job has been submitted, SLURM will provide an identifier and will handle its execution. The status of all submitted jobs can be checked with the `squeue` command:

```
squeue
```

Which will show all submitted jobs and their status (e.g. pending, running, etc.)

In addition, another command that can become useful is to cancel a submitted job. It can be done with the `scancel` command followed by the:

```
scancel <JOB_ID>
```

4) How to get the task dependency graph

The task dependency graph can be achieved easily by adding the "`--graph=true`" flag to the "`pycompss job submit`" command.
The next command extends the previous example (shown in step 3) requesting the task dependency graph:

```
pycompss job submit \
  --qos=training \
  --num_nodes=2 \
  --exec_time=10 \
  --reservation=<RESERVATION_NAME> \
  --lang=python \
  --graph=true \
  /path/to/application.py <param_1> <param_2> … <param_N>
```

TIP: Alternatively, it is possible to use "`--graph`" or even shorter with "`-g`".

The task dependency graph generated after the application execution will be located within the logs folder:

```
$HOME/.COMPSs/<JOB_ID>/monitor
```

Where `JOB_ID` corresponds to the job identifier provided by SLURM when the job is launched.

The task dependency graph can be consulted with the following commands:

```
cd $HOME/.COMPSs/<JOB_ID>/monitor  # please, update <JOB_ID>
compss_gengraph complete_graph.dot
evince complete_graph.pdf
```

The "`compss_gengraph`" command will convert the complete graph in dot format to pdf, so that it can be displayed with evince.

CAUTION: COMPSs module needs to be loaded in order to have the "`compss_gengraph`" command available. If the command is not found, please load COMPSs as described in Section 2.

CAUTION: It is important to have connected to MN4 using the -X flag with ssh (shown in step 1), since this enables the X11 display. Otherwise, evince will display an error and will not show the task dependency graph. The solution is to disconnect and reconnect using -X flag with ssh and try again.

5) How to get the trace

Similarly to the task dependency graph, the trace can be achieved easily by adding the "`--trace=true`" flag to the "`pycompss job submit`" command.
The next command extends the previous example (shown in step 4) requesting the trace (as well as the task dependency graph):

```
pycompss job submit \
  --qos=training \
  --num_nodes=2 \
  --exec_time=10 \
  --reservation=<RESERVATION_NAME> \
  --lang=python \
  --graph=true \
  --tracing=true \
  /path/to/application.py <param_1> <param_2> … <param_N>
```

TIP: Alternatively, it is possible to use "`--trace`" or even shorter with "`-t`".

The trace generated after the application execution will be located within the logs folder:

```
$HOME/.COMPSs/<JOB_ID>/trace
```

Where `JOB_ID` corresponds to the job identifier provided by SLURM when the job is launched.
The trace can be consulted with the following commands:

```
cd $HOME/.COMPSs/<JOB_ID>/trace  # please, update <JOB_ID>
compss_gentrace
module load paraver
wxparaver <TRACE_NAME>.prv  # please, update the <TRACE_NAME>
```

The "`compss_gengraph`" command will convert the complete graph in dot format to pdf, so that it can be displayed with wxparaver.

Further information about trace analysis can be found in the COMPSs manual.

CAUTION: COMPSs module needs to be loaded in order to have the "`compss_gentrace`" command available. If the command is not found, please load COMPSs as described in Section 2.

CAUTION: It is important to have connected to MN4 using the -X flag with ssh (shown in step 1), since this enables the X11 display. Otherwise, wxparaver will display an error and will not show the trace. The solution is to disconnect and reconnect using -X flag with ssh and try again.

6) Sample application

This section will guide you through the execution of a sample application in MN4.

6.1) Get sample code

The first step is to get the code of the sample application. To this end, the following command will copy the sample application into the current folder.

```
cp -r /apps/COMPSs/TUTORIALS/2023_EGU11/example_egu11 .
```

The content of this folder is:

File name	Description
`gen_data.sh`	Script that generates input/initial data
`launch_gs.sh`	Script to launch the "my_main_gridsearch.py" application.
`launch.sh`	Script to launch the "my_workflow_multiple.py" application.
`my_analytic`	Binary able to perform an analytic
`my_analitic.c`	Source code of the analytic binary
`my_main_gridsearch.py`	Sample application. Extension of the "my_workflow_multiple" using dislib to perform a Grid Search.
`my_sim`	Binary able to perform a simulation
`my_sim.c`	Source code of the simulation binary
`my_workflow_multiple.py`	Sample application. PyCOMPSs workflow combining "gen_data.sh", "my_sim" and "my_analytic" binaries.

6.2) Launch sample code

There are two applications that can be launched in the sample code. This section describes how to launch both of them:

6.2.1) my_workflow_multiple.py: application description

The application illustrates a theoretical workflow that combines multiple steps of different nature (computation and data analytics):

- data_generation: PyCOMPSs task that generates an ensemble of input files. It emulates the source ensemble data generation phase of the workflow (file_1, file_2, file_3, etc).
- my_sim: Given a file_i input file generates a file_1.out file. For this simple example, it generates 10 random numbers that are written in the output file. It emulates the simulation step of the workflow and it will be invoked as many times as input files generated in the previous phase. This simulator is an external sequential binary. In a real case, it could be a parallel application (i.e., OpenMP or MPI).
- my_analytic: Gathers all output files of previous steps and performs a data analytics on them. The sample computation for illustrative purposes, averages the random numbers of each output file and computes the maximum between all of them. It emulates the data analytic step in a real workflow. In this example, it is again a sequential binary, while in real cases can be parallel, invoking external libraries (i.e., PyTorch, Spark, etc).
- get_result: a PyCOMPSs task that extracts the final result from the output file generated by the my_analytic step

6.2.2) my_workflow_multiple.py: execution

In order to launch the "`my_workflow_multiple.py`" application, we provide the "`launch.sh`" script. Please, check it out and do any modification to the flags required. Once checked, you can launch the application with the following command:

```
./launch.sh
```

It will display the job identifier, for example:

```
bsc19XXX@login3:~/example_egu11> ./launch.sh
load java/8u131 (PATH, MANPATH, JAVA_HOME, JAVA_ROOT, JAVA_BINDIR,
SDK_HOME, JDK_HOME, JRE_HOME)
load papi/5.5.1 (PATH, LD_LIBRARY_PATH, C_INCLUDE_PATH)
load PYTHON/3.7.4 (PATH, MANPATH, LD_LIBRARY_PATH, LIBRARY_PATH,
PKG_CONFIG_PATH, C_INCLUDE_PATH, CPLUS_INCLUDE_PATH, PYTHONHOME,
PYTHONPATH)
load COMPSs/3.2 (PATH, CLASSPATH, MANPATH, GAT_LOCATION, COMPSS_HOME,
JAVA_TOOL_OPTIONS, LDFLAGS, CPPFLAGS)
Job submitted: 28611180
```

And when the application is executed, some files will appear in the current folder (application results and execution logs).The execution logs are "compss-<JOB_ID>.out/err" files that show the standard output and error from the execution.

In addition, the logs folder within $HOME/COMPSs/<JOB_ID>/ will be created, and contains the execution logs (useful for debugging) as well as the task dependency graph (which is enabled in launch.sh).
Check section 4 to display the task dependency graph.

6.2.3) my_main_gridsearch.py: tuning application parameters

The previous step launches the execution of a single instance of the workflow. However, in many cases applications have input parameters that need to be tuned in order to work properly. For this purpose we can use the SimulationGridSearch method from the dislib. This method receives as input a set of parameters that we want to explore with a given application and the application name. On execution, an instance of the application is run for each of the parameters being provided and the results are gathered.

6.2.3) my_main_gridsearch.py: execution

Similarly to the "my_workflow_multiple.py", the "my_main_gridsearch.py" script can be launched with the "launch_gs.sh" script also provided. Please, check it out and do any modification to the flags required.
Once checked, you can launch the application with the following command:

```
./launch_gs.sh
```

It will display the job identifier, for example:

```
bsc19XXX@login3:~/example_egu11> ./launch_gs.sh
load java/8u131 (PATH, MANPATH, JAVA_HOME, JAVA_ROOT, JAVA_BINDIR,
SDK_HOME, JDK_HOME, JRE_HOME)
load papi/5.5.1 (PATH, LD_LIBRARY_PATH, C_INCLUDE_PATH)
load PYTHON/3.7.4 (PATH, MANPATH, LD_LIBRARY_PATH, LIBRARY_PATH,
PKG_CONFIG_PATH, C_INCLUDE_PATH, CPLUS_INCLUDE_PATH, PYTHONHOME,
PYTHONPATH)
load COMPSs/3.2 (PATH, CLASSPATH, MANPATH, GAT_LOCATION, COMPSS_HOME,
JAVA_TOOL_OPTIONS, LDFLAGS, CPPFLAGS)
Job submitted: 28611181
```

And when the application is executed, some files will appear in the current folder (application results and execution logs).The execution logs are "compss-<JOB_ID>.out/err" files that show the standard output and error from the execution.

In addition, the logs folder within $HOME/COMPSs/<JOB_ID>/ will be created, and contains the execution logs (useful for debugging) as well as the task dependency graph and the trace (which are enabled in launch_gs.sh).
Check Section 4 to display the task dependency graph and Section 5 to display the trace.

7) Further information

- Project page: http://www.bsc.es/compss
  - Virtual Appliance for testing & sample applications
  - Tutorials
- Documentation: https://compss.readthedocs.org
  - Installation
  - Programming model
  - Deployment
  - Tools
  - Troubleshooting
  - Tutorial
- Source Code: https://github.com/bsc-wdc/compss
- Docker Image: https://hub.docker.com/r/compss/compss/
- Applications:
  - https://github.com/bsc-wdc/apps
  - https://github.com/bsc-wdc/dislib