Table of Contents

RES - Red Española de Supercomputación **2**
Altamira 2
Running Jobs 2
Submitting Jobs 2
Job Directives 2
Job Examples 3
MareNostrum 4
Running Jobs 4
Submitting Jobs 4
LSF Commands 4
Job Directives 4
Job Examples 5
Tirant 6
Running Jobs 6
Classes 6
Submitting Jobs 6
Job Directives 7
Job Examples 7
?DKRZ **8**
Blizzard 8
Lizard 8
National Computational Infrastructure (Australia) **8**
VAYU 8
ECMWF **8**
HPCF 8
Ecgate 8

RES - Red Española de Supercomputación

Altamira

```
ssh  <userid>@altamira1.ifca.es
```

Running Jobs

SLURM is the utility used at Altamira for batch processing support, so all jobs must be run through it. This document provides information for getting started with job execution at Altamira.

In order to keep the login nodes in a propper load, a 10 minutes limitation in the cpu time is set for processes running interactively in these nodes. Any execution taking more than this limit should be carried out through the queue system.

Submitting Jobs

A job is the execution unit for the SLURM. A job is defined by a text file containing a set of directives describing the job, and the commands to execute.

These are the basic directives to submit jobs:

mnsubmit <job_script> submits a job script to the queue system (see below for job script directives).

mnq shows all the jobs submitted.

mncancel <job_id> removes his/her job from the queue system, canceling the execution of the job if it was already running.

Job Directives

A job must contain a series of directives to inform the batch system about the characteristics of the job. These directives appear as comments in the job script, with the following syntax:

```
#@ directive = value
```

Additionally, the job script may contain a set of commands to execute. If not, an external script must be provided with the 'executable' directive. Here you may find the most common directives:

```
#@ class = class_name
```

The queue where the job is to be submitted. Let this field empty unless you need to use "debug" or special queues.

```
#@ wall_clock_limit = HH:MM:SS
```

The limit of wall clock time. This is a mandatory field and you must set it to a value greater than the real execution time for your application and smaller than the time limits granted to the user. Notice that your job will be killed after the elapsed period.

```
#@ initialdir = pathname
```

The working directory of your job (i.e. where the job will run). If not specified, it is the current working directory at the time the job was submitted.

```
#@ error = file
```

The name of the file to collect the stderr output of the job.

```
#@ output = file
```

The name of the file to collect the standard output (stdout) of the job.

```
#@ total_tasks = number
```

The number of processes to start.

```
#@ cpus_per_task = number
```

The number of cpus allocated for each task. This is useful for hybrid MPI+OpenMP applications, where each process will spawn a number of threads. The number of cpus per task must be between 1 and 16, since each node has 16 cores (one for each thread).

```
#@ tasks_per_node = number
```

The number of tasks allocated in each node. When an application uses more than 3.8 GB of memory per process, it is not possible to have 16 processes in the same node and its 64GB of memory. It can be combined with the cpus_per_task to allocate the nodes exclusively, i.e. to allocate 2, processes per node, set both directives to 2. The number of tasks per node must be between 1 and 16.

```
# @ gpus_per_node = number
```

The number of GPU cards assigned to the job. This number can be [0,1,2] as there are 2 cards per node.

Job Examples

In the examples, the %j part in the job directives will be sustitute by the job ID.

Sequential job:

```
#!/bin/bash
#@ job_name = test_serial
#@ initialdir = .
#@ output = serial_%j.out
#@ error = serial_%j.err
#@ total_tasks = 1
#@ wall_clock_limit = 00:02:00

./serial_binary
```

Parallel job:

```
#!/bin/bash
#@ job_name = test_parallel
#@ initialdir = .
#@ output = mpi_%j.out
#@ error = mpi_%j.err
#@ total_tasks = 32
#@ wall_clock_limit = 00:02:00

srun ./parallel_binary
```

GPGPU job:

```
#!/bin/bash
#@ job_name = test_gpu
#@ initialdir = .
#@ output = gpu_%j.out
#@ error = gpu_%j.err
#@ total_tasks = 1
#@ gpus_per_node = 1
#@ wall_clock_limit = 00:02:00

./gpu_binary
```

The jobs with GPU should execute module load CUDA in order to set the library paths before running mnsubmit.

For more information about ALTIMARIA see ?https://moin.ifca.es/wiki/Supercomputing/Userguide

MareNostrum

```
ssh  <userid>@mn1.bsc.es
```

Running Jobs

LSF is the utility used at MareNostrum III for batch processing support, so all jobs must be run through it. This document provides information for getting started with job execution at the Cluster.

Submitting Jobs

A job is the execution unit for LSF. A job is defined by a text file containing a set of directives describing the job, and the commands to execute. Please, bear in mind that there is a limit of 3600 bytes for the size of the text file.

LSF Commands

These are the basic directives to submit jobs:

bsub < job_script submits a ?job script? to the queue system (see below for job script directives). Remember to pass it through STDIN '<'

bjobs [-w][-X][-l job_id] shows all the submitted jobs.

bkill <job_id> remove the job from the queue system, canceling the execution of the processes, if they were still running.

Job Directives

A job must contain a series of directives to inform the batch system about the characteristics of the job. These directives appear as comments in the job script, with the following syntax:

```
#BSUB -option value
```

```
#BSUB -J job_name
```

The name of the job.

```
#BSUB -q debug
```

This queue is only intended for small tests, so there is a limit of 1 job per user, using up to 64 cpus (4 nodes), and one hour of wall clock limit.

```
#BSUB -W HH:MM
```

NOTE: take into account that you can not specify the amount of seconds in LSF. The limit of wall clock time. This is a mandatory field and you must set it to a value greater than the real execution time for your application and smaller than the time limits granted to the user. Notice that your job will be killed after the elapsed period.

```
#BSUB -cwd pathname
```

The working directory of your job (i.e. where the job will run). If not specified, it is the current working directory at the time the job was submitted.

```
#BSUB -e/-eo file
```

The name of the file to collect the stderr output of the job. You can use %J for job_id. -e option will APPEND the file, -eo will REPLACE the file.

```
#BSUB -o/-oo file
```

The name of the file to collect the standard output (stdout) of the job. -o option will APPEND the file, -oo will REPLACE the file.

```
#BSUB -n number
```

The number of processes to start.

```
#BSUB -R"span[ptile=number]"
```

The number of processes assigned to a node.

We really encourage you to read the manual of bsub command to find out other specifications that will help you to define the job script.

```
man bsub
```

Job Examples

Sequential job :

```
#!/bin/bash
#BSUB -n 1
#BSUB -oo output_%J.out
#BSUB -eo output_%J.err
#BSUB -J sequential
#BSUB -W 00:05

./serial.exe
```

Sequential job using OpenMP :

```
#!/bin/bash
#BSUB -n 1
#BSUB -oo output_%J.out
#BSUB -eo output_%J.err
#BSUB -J sequential_OpenMP
#BSUB -W 00:05

export OMP_NUM_THREADS=16

./serial.exe
```

Parallel job :

```
#!/bin/bash
#BSUB -n 128
#BSUB -o output_%J.out
#BSUB -e output_%J.err
# In order to launch 128 processes with 16 processes per node:
#BSUB -R"span[ptile=16]"
#BSUB -x # Exclusive use
#BSUB -J parallel
#BSUB -W 02:00
# You can choose the parallel environment through modules

module load intel openmpi
mpirun ./wrf.exe
```

Parallel job using threads:

```
#!/bin/bash
# The total number of MPI processes:
#BSUB -n 128
```

```
#BSUB -oo output_%J.out
#BSUB -eo output_%J.err
# It will allocate 4 MPI processes per node:
#BSUB -R"span[ptile=4]"
#BSUB -x # Exclusive use
#BSUB -J hybrid
#BSUB -W 02:00
# You can choose the parallel environment through
# modules

module load intel openmpi
# 4 MPI processes per node and 16 cpus available
# (4 threads per MPI process):

export OMP_NUM_THREADS=4

mpirun ./wrf.exe
```

For more information about MareNostrum III see ?http://www.bsc.es/support/MareNostrum3-ug.pdf

Tirant

```
ssh username@tirant1.uv.es
```

Running Jobs

Slurm+MOAB is the new utility used at Tirant for batch processing support. We moved from LoadLeveler and all the jobs must be run through this new batch system. We tried to keep it as simple as possible keeping the syntax from the LoadLeveler to make the transition easier to those who used !Loadleveler in the past. This document provides information for getting started with job execution at Tirant.

Classes

The user?s limits are assigned automatically to each particular user (depending on the resources granted by the Access Committee) and there is no reason to explicitly set the #@class directive. Anyway you are allowed to use the special class: "debug" in order to performe some fast short tests. To use the "debug" class you need to include the #@class directive

Class	Max CPUs	CPU Time	Wall time limit
debug	64	10 min	10 min

The specific limits assigned to each user depends on the priority granted by the access committee. Users granted with ?high priority hours? will have access to a maximum of 1024 CPUs and a maximum wall clock limit of 72 hours. For users with ?low priority hours? the limits are 1024 CPUs and 24 hours. If you need to increase these limits please contact the support group.

Local users of Tirant have access to a new local queue called "class t". This queue has same parameters as "class a", but the priority is modified to restrict local time consumption to 20% of total computation time.

- debug: This class is reserved for testing the applications before submitting them to the 'production' queues. Only one job per user is allowed to run simultaneously in this queue, and the execution time will be limited to 10 minutes. The maximum number of nodes per application

is 32. Only a limited number of jobs may be running at the same time in this queue. The specifications for each class may be adjusted in the future to adapt to changing requirements.

Submitting Jobs

A job is the execution unit for the SLURM. We have created wrappers to make easier the adaptation to the new batch system to those users who have already used Tirant and MareNostrum in the past. So the commands are quite similar to the former !Loadleveler commands. A job is defined by a text file containing a set of directives describing the job, and the commands to execute.

These are the basic directives to submit jobs:

- **mnsubmit <jobscript>** submits a 'job script' to the queue system (see below for job script direc-tives).

- **mnq** shows all the jobs submitted.
- **mncancel <jobid>** remove his/her job from the queue system, canceling the execution of the processes, if they were already running.
- **checkjob <jobid>** obtains detailed information about a specific job, including the assigned nodes and the possible reasons preventing the job from running.
- **mnstart** shows information about the estimated time for the specified job to be executed.

Job Directives

A job must contain a series of directives to inform the batch system about the characteristics of the job. These directives appear as comments in the job script, with the following syntax:

```
# @ directive = value
```

Additionally, the job script may contain a set of commands to execute. If not, an external script must be provided with the 'executable' directive. Here you may find the most common directives:

```
# @ class = class\_name
```

The partition where the job is to be submitted. Let this field empty unless you need to use "interactive" or "debug" partitions.

```
# @ wall_clock_limit = HH:MM:SS
```

The limit of wall clock time. This is a mandatory field and you must set it to a value greater than the real execution time for your application and smaller than the time limits granted to the user. Notice that your job will be killed after the elapsed period.

```
# @ initialdir = pathname
```

The working directory of your job (i.e. where the job will run). If not specified, it is the current working directory at the time the job was submitted.

```
# @ error = file
```

The name of the file to collect the stderr output of the job.

```
# @ output = file
```

The name of the file to collect the standard output (stdout) of the job.

```
# @ total_tasks = number
```

The number of processes to start.

Job Examples

Serial job:

```
#!/bin/bash
# @ job_name = testserial
# @ initialdir = .
# @ output = serial%j.out
# @ error = serial%j.err
# @ total_tasks = 1
# @ wall_clock_limit = 00:02:00

./serialbinary > serial.out
```

Parallel job :

```
#!/bin/bash
# @ job_name = test_parallel
# @ initialdir = .
# @ output = mpi%j.out
# @ error = mpi%j.err
# @ total_tasks = 56
# @ wall_clock_limit = 00:02:00

srun ./parallelbinary > parallel.output
```

?DKRZ

How to use DKRZ facilities?

Workflows in climate modelling research are complex and comprise, in general, a number of different tasks, such as model formulation and development (including debugging, platform porting, and performance optimization), generation of input data, performing model simulations, postprocessing, visualization and analysis of output data, long-term archiving of the data, documentation and publication of results. The **DKRZ** hardware and software infrastructure is optimally adapted to accomplish these tasks in an efficient way. In the graphic below we give a schematic overview on the **DKRZ** systems.

For a more detailed description of the different systems shown in the picture and basic software installed on these systems ?click here.

Blizzard

?http://www.dkrz.de/Nutzerportal-en/doku/blizzard

```
ssh  <userid>@blizzard.dkrz.de
```

Lizard

?http://www.dkrz.de/Nutzerportal-en/doku/blizzard/lizard

```
ssh  <userid>@lizard.dkrz.de
```

National Computational Infrastructure (Australia)

?http://nf.nci.org.au/facilities/

VAYU

?http://nf.nci.org.au/facilities/vayu/system_doc.php

ECMWF

```
ssh <user>@ecaccess.ecmwf.int
```

HPCF

?http://www.ecmwf.int/services/computing/overview/ibm_cluster.html

Ecgate