A brief introduction to DEI IBM POWER clusters

Login to the server:

`ssh username@power7a.dei.unipd.it`

Maybe the system will ask you to change your password the first time you log in, otherwise use the `passwd` command to change it after the first usage.

Basic usage:

Using the system might be tricky if you haven't ever used a batch system. You cannot have an interactive access to the computing resources: instead you have to request them through a 'job file' which goes along with your executable.
This job file have a particular syntax and must be *submitted* to the system scheduler. In the IBM world the scheduler is called LoadLeveler.

LoadLeveler micro-primer:

Create your job file (simulation.job):

```
#---CUT HERE (simulation.job)---
#!/bin/bash
#@ job_type = parallel
#@ initialdir = /home/username/simulations
#@ input = /dev/null
#@ output = /home/username/simulations/my_output.txt
#@ error = /home/username/simulations/my_error.txt
#@ notification = error
#@ class = short
#@ blocking = UNLIMITED
#@ total_tasks = 20
#@ queue

./mysimulation -my_parameters
#---ENDS HERE---
```

Dissection of simulation.job:

```
#!/bin/bash
#@ job_type = parallel
```

The first line is the 'command interpreter': the lines of your job file beginning with `'#'` are treated as comments (but those starting with `#@` are intercepted by LoadLeveler). Every other line is executed by the interpreter (bash in this case) as if it was typed on the command line.

`job_type` could be '`parallel`' or '`serial`'.

```
#@ initialdir = /home/username/simulations
```

This is your 'base' directory containing at least *simulation.job* and the executable and where you would normally put your outputs. If the job file is in the same directory of all the other files you can also write

`initialdir = .` (the period meaning 'my current directory').

```
#@ input = /dev/null
```

Normally it's easier to specify the input on the command line (see below) so you're stating here that you don't need any input.

```
#@ output = /home/username/simulations/my_output.txt
```

This is one of many ways to get your output: if your simulation writes on its own files it may be possible that you don't need this either. Some error messages coming from the system might go here also.

Tip: if you specified `initialdir = .` you can change the line in simply

```
#@ output = my_output.txt
```

```
#@ output=/home/username/simulations/my_error.txt
```

This is where errors you normally see on the screen go.

Tip: if you specified `initialdir = .` you can change the line in simply

```
#@ error = my_error.txt
```

```
#@ notification = error
```

The system will mail you in case something goes wrong.

```
#@ class = short
```

This is where LoadLeveler comes in: the '`class`' directive tells the scheduler that your job will run for up to 2 minutes. These limits (2 minutes, maximum number of processors...) are hardcoded on the definition of the class. There are other classes you can use: you can list them all with the '`llclass`' command (try it at least once!).

```
#@ blocking = UNLIMITED
#@ total_tasks = 20
```

You can safely ignore the meaning of the '`blocking`' directive and use it as is.

The '`total_tasks`' is where you ask the system to reserve the CPUs for your

MPI/parallel programs.

```
#@ queue
```

Mandatory: it ends the block of requirements.

```
./mysimulation -my_parameters
```

This is the command line you would normally type at the console. The path is relative to `initialdir`.

Submitting your job:

At the console, type

```
$ llsubmit simulation.job
```

The system will answer something like:

```
llsubmit: The job "power7a.dei.unipd.it.28" has been submitted.
```

The number '28' is your '*job id*': the scheduler will queue your job and execute it as soon as possible. The global execution queue is listed by using the '`llq`' command. As an example:

```
$ llq
Id                  Owner      Submitted   ST PRI Class        Running On
----------------- ---------- ----------- -- --- ------------ -----------
power7a.29.0       paolo      12/20 12:20 R  50  verylong     power7l

1 job step(s) in queue, 0 waiting, 0 pending, 1 running, 0 held, 0 preempted
```

Above you see that my job is running since the `ST` (state) column says '`R`' and that I'm using the '`verylong`' class.
If something goes wrong it may be possible that your remains stuck in the '`I`' (idle) state: in that case you can use

```
llq -l -s your_job_id
```

to ask the scheduler what's happening.

Cancel any job you don't want to run anymore using

```
llcancel your_job_id
```

Getting further documentation:

- man *command_i_know_nothing_of*

- IBM Cluster information center:

http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/index.jsp

look for Tivoli Workload Scheduler LoadLeveler pages.

Improving your job file:

Frequently you will do many repetitions of your simulations. Using the 'basic' version of *simulation.job* ends up in overwriting your output and error files on every run: can we do better? The answer is yes and the solution consists in 'numbering' automatically your files. You can use for this purpose your *jobid* or the current date and time:

- using *jobid*

```
#@ output = simulation_${jobid}.out.txt
#@ error = simulation_${jobid}.err.txt
```

- using the current date and time:

```
#!/bin/bash
...
...  (other LoadLeveler directives)
...
now = `date +%Y%m%d-%H%M%S`        (notice that ` are backticks characters)
#@ output = simulation_${now}.out.txt
#@ error = simulation_${now}.err.txt
```

Interactive execution:

For a quick trial or for debugging purposes, you may want to launch your program like you would do with a serial one in an interactive session. This can be done by using a *host.list* file: this is a simple text file where you name the hosts (one per line) where you want your program to be launched. Since our Power cluster consist of only one (big) host, your host.list will be made of N or more 'power7a' equal lines l, N being the number of tasks you need. Of course any number M>N of lines will do.

Example: you want to run myprogram on 4 tasks on 'power7a' cluster.

- contents of host.list:

```
power7a
power7a
power7a
power7a
```

- from the shell:

```
$ ./myprogram -procs 4
```

Remember to put *host.list* in the same directory of *myprogram*