# Queue systems

## and how to use Torque/Maui

Piero Calucci

Scuola Internazionale Superiore di Studi Avanzati
Trieste

March 9th 2007
Advanced School
in High Performance Computing Tools
for e-Science

# Outline

# The User's Problem

- have dedicated resources
  *multitasking is Bad for HPC*
- have resources as soon as possible
  *you need to have your computation done by next week, right?*
- have jobs run unattended and results delivered back to you
  *what do you want to do at 4.30AM?*

# The Admin's Problem

- minimize resource waste
- promote fair share of resources
  *a.k.a. «avoid complaints from users»*
- monitor and account for everything

# The Resource Manager

At the core of a batch system there is a RM that:

- accepts job submissions from users
- tracks resource usage
- delivers jobs to execution nodes
- informs users about job status

# The TORQUE Resource Manager

The **T**erascale **O**pen-source **R**esource and **QUE**ue manager is deployed as

- a server component (`pbs_server`) on the masternode
- an execution mini-server (`pbs_mom`) on each execution node

There is also a scheduler component, but we will use the Maui Scheduler instead – more on this later

# A Job's Life

1. a job is a shell script that contains a description of the resources needed and the command you want to execute
2. you submit the job to the batch system
3. the batch system sends the job to an execution queue where it is executed without human intervention
4. job results are then delivered back to you

# Job Must Be a Shell Script

A job script contains a description of the resources you request and all the commands your job needs to perform.
Resource description always comes at the beginning of the script and is identified by the `#PBS` mark.

```
#!/bin/sh
#PBS -l walltime=1:00:00
#PBS -l nodes=1:ppn=2
#PBS -N MyTestJob
do_something_useful
do_more
exit 0
```

# Job Submission

Jobs are submitted to the batch system by means of the `qsub` command, as in

```
qsub job.sh
```

But you can also add resource description directly on the command line:

```
qsub -l nodes=4:ppn=4 job.sh
```

This is especially useful when you are experimenting with subtle variations of a job submission.

# Queues

Batch systems are usually configured with multiple queues.
Each queue can be configured to accept job from a certain group of users, or within specified resource limits, or simply on request from the user.
Be sure to select the right queue for your jobs.
Queue selection is performed with `-q queuename` on the `qsub` command line or with `#PBS -q queuename` in the job script.

# Simple Resource Specification

| | |
|---|---|
| `-l nodes=`*n* | request *n* execution nodes |
| `-l nodes=`*n*`:ppn=`*m* | request *n* execution nodes with *m* CPUs each |
| `-l walltime=`*n* | request *n* seconds of wallclock time (walltime can be specified also as *hours:minutes:seconds* |
| `-l nodes=`*n*`:`*feature* | request *n* nodes with a specific feature e.g. we use `:myri` for nodes with Myrinet cards |
| `-q` *name* | submit job to named queue |
| `-N` *name* | give job a name |

# Interactive Jobs

If resources are available right now you can run interactive jobs with `qsub -I`
In an interactive job you are given a shell on a computing node and are allowed to execute all your computation interactively, possibly on several nodes.

```
masternode $ qsub -I -q smp -l walltime=5:00 -l
nodes=1:ppn=2
qsub:  job 29506.cerbero.hpc.sissa.it ready

a211 $
```

# (No) Access to Computing Nodes

- no «normal» user access to computing nodes
- access permissions are created on the fly by the RM when (and where) needed for your job to run
- while a job is running you are granted interactive access to nodes allocated to your job
- at job completion access rights are cleared

# Node Access and Resource Limit Enforcement

- access right is granted only to nodes allocated to your job
  this enforces the limit on the number of nodes you can access and guarantees that no concurrent usage of a resource is possible
- access right is granted only for the walltime allocated to your job
  when your allocated walltime expires, you are given a short *grace time*, then all your processes on the computing node are killed
- you should arrange so that your jobs completes before the walltime limit, or save partial results before the job is killed

# Queue Status

| | |
|---|---|
| `qstat` | query queue status |
| `qstat -a` | alternate form |
| `qstat -r` | show only running jobs |
| `qstat -rn` | only running jobs, w/ list of allocated nodes |
| `qstat -i` | only idle jobs |
| `qstat -u` *`username`* | show jobs for named user |

# Job Trace

`tracejob` *id* show what happened today to job *id*

`tracejob -n` *d id* search last *d* days

searching the RM logs is a time-consuming operation, don't abuse it!

```
$ tracejob 29506
Job:  29506.cerbero.hpc.sissa.it
02/26/2007 10:12:39 S Job Queued at request of cxxxxx@cerbero.hpc.sissa.it, owner =
cxxxxxx@cerbero.hpc.sissa.it, job name = STDIN, queue = em64ts
02/26/2007 10:12:40 S Job Modified at request of maui@cerbero.hpc.sissa.it
02/26/2007 10:12:40 S Job Run at request of maui@cerbero.hpc.sissa.it
02/26/2007 10:12:40 S Job Modified at request of maui@cerbero.hpc.sissa.it

02/26/2007 10:19:36 S Exit_status=265 resources_used.cput=00:00:00

resources_used.mem=2940kb resources_used.vmem=89532kb

resources_used.walltime=00:06:51
```

# The Scheduler

The Maui Scheduler prioritizes jobs in the idle queue, according to admin-defined policies. The highest-priority job is run as soon as resources are available.

Jobs can be blocked if their requirements exceed available resources. Blocked jobs have an undefined priority.

Job priorities are recomputed at each scheduler iteration, so your job can move up and down the idle queue as an effect of resource usage by other jobs of yours.

# Queues as Seen by Maui

```
$ showq
ACTIVE JOBS--------------
 JOBNAME   USERNAME     STATE   PROC   REMAINING   STARTTIME

 29199        axxxx   Running     32     1:59:17     Wed ...
 29055     sxxxxxxx   Running      8     4:03:07     Tue ...
 28496     mxxxxxxx   Running      4     5:24:00     Sat ...
                ...
 27 Active Jobs   125 of   142   Processors Active   (88.03%)
                   52 of    58   Nodes Active        (89.66%)
IDLE JOBS----------------
 JOBNAME   USERNAME   STATE   PROC      WCLIMIT        QUEUETIME

 29069        sxxxx    Idle      4   1:21:00:00   Mon Feb 19 ...
 29019     kxxxxxxx    Idle      4   4:00:00:00   Mon Feb 19 ...
 29076      fxxxxxx    Idle      4   4:00:00:00   Mon Feb 19 ...
                ...
22 Idle Jobs
BLOCKED JOBS-----------
 JOBNAME   USERNAME       STATE   PROC      WCLIMIT        QUEUETIME

 28777     rxxxxxxx        Hold      8   2:00:00:00   Thu Feb 15
```

# The Backfill Window

| | node 1 | node 2 | node 3 |
|---|---|---|---|
| 0:00 | job1 | job1 | job3 |
| 1:00 | job1 | job1 | job3 |
| 2:00 | job2 | job2 | job2 |

- *job2* cannot run until *job1* is done
- if you submit a *job3* that requires only one node for two hours or less you can run before *job2*!

# Discovering Free Resources

The `showbf` command queries the scheduler and display resources that are available for immediate use.

| | |
|---|---|
| `showbf` | summary of free resources |
| `showbf -f myri` | select only nodes with a given feature |
| `showbf -p intel` | select only nodes in a given partition |

```
$ showbf
backfill window (user:  'cxxxxxx' group:
'bxxxxxxx' partition:  ALL) Mon Feb 26 13:46:16
5 procs available with no timelimit
$ showbf -f myri
backfill window (user:  'cxxxxxx' group:
'bxxxxxxx' partition:  ALL) Mon Feb 26 13:49:16
no procs available
$ showbf -p intel
```