Torque implementation at Leiden Observatory

Quick start

To submit a job, use the qsub command. Default queue is 'para', which executes on (any number of) the para cluster nodes. To choose another queue, use the -q queuename option, e.g. qsub -q lofar jobscript See below for queue names, and for links to general documentation of the software.

To view the queue, use the qstat command, e.g. qstat -q para. A few useful options to qstat:

- -n shows the list of nodes on which the job is executing (unless it is only queued and waiting to be executed, in which case the list of nodes is not known yet).
- -f gives a full status display, with information like the time the job was queued, the name of error and output files, variables set by Torque.

Column S denotes the current status of your job: Q = queued, R = running, C = completed. Completed jobs will be shown in the overview for 1 hour after completion. Note that qstat only shows your own jobs, unless you are root. If you want to see some basic information about the jobs of other users, use the command showq. To remove a job from the queue, use qdel. If that doesn't work, qdel -p removes the job even if it cannot be killed, but this may leave leftover jobs and files around, so use this only if a normal qdel fails.

Job files

It is usually convenient to store options for your job together with the command instructions, so that you can simply submit a new job like 'qsub jobscript' without having to type all the options all the time. This can be done by having the options in shell-comments (starting with #) in the script, e.g.

```
#!/bin/tcsh -f
#PBS -S /bin/tcsh
#PBS -j oe
#PBS -q para
#PBS -l nodes=2:ppn=4
#PBS -l walltime=01:00:00
#
cd $PBS_O_WORKDIR
setenv PATH $PBS_O_PATH
#
/disks/para33/user/src/program
```

Running this job script is equivalent to specifying -q para -l nodes=2:ppn=4 -l walltime=01:00:00 -j oe on the qsub commandline. In this case that means:

- -j oe: join output and errors in one file
- -q para: use the para queue (default)

- -l nodes=2:ppn=4: a resource list, specifying you want the job to run on 2 nodes, and 4 processors per node
- -l walltime=01:00:00 : a limit on the “wall time” (real clock time). See section on scheduling
- -S /bin/tcsh : specifies that the job script is executes through the tcsh shell

See 'man qsub' for an explanation of all the options. Note also the cd and setenv commands. As described in the torque documentation, very little of the environment is preserved in jobs so these commands make sure you start in the directory where the job was submitted, and sets the PATH to the old value (not always necessary, but it may prevent strange problems, in case you run programs that are not in the detfault path, e.g. MPI utilities).

A note before submitting a job: not only inside the jobscript do you need to use network-visiable disk names (eg /net/para33/data2/user), but this is often also needed in the environment from where you submit the job. E.g. if your job script is on para33 in /data2/user and you go there directly with the cd command, the job will not be able to write output files in the right location. So go there using `cd /net/para33/data2/user` or specify a full path name for output and error files, and for each file and executable referenced in the script.

Queues

para: default and generally accessible queue, access to para33 and para35 - para37 machines
lofarq: The special purpose queue for LOFAR processing only. This queue is avaialble for lofar group members only
In the future, there may be additional queues added to the system, such as a queue to make use of the cpu power of fast desktop machines, or a high priority queue limited to just a few cpus (as opposed to lower priority work using many or all available cpus)

MPI

Both MPICH and OpenMPI are available on the para cluster (as they are on all our Linux machines). MPICH is initialized by default in our standard environment. Switching to OpenMPI can be done using

```
module switch mpi/openmpi-x86_64
```

To switch back to MPICH (in case you have changed your account's default):

```
module switch mpi/mpich-x86_64
```

See also Sfinx modules.

To make MPI work nicely with PBS (Torque), you can make use of the special environment variables that are set during job execution. A list of assigned nodes is available in `$PBS_NODEFILE` (one entry per line, and if your job runs on multpile cores on the same host, that host is listed multiple times). Other useful variables are `$PBS_NUM_PPN` (processors per node), `$PBS_NUM_NODES` (number of nodes), and `$PBS_NP` (total number of processors). These can be matched to the options expected by mpirun, e.g.:

```
mpirun -f $PBS_NODEFILE -np $PBS_NP $PBS_O_WORKDIR/programname
```

Note: Replace `-f` by `-hostfile` when using OpenMPI. Of course you should make sure that your program was compiled with the MPI version you are using!

Scheduling

A few notes on the scheduling algorithm used by the Maui scheduler. Understanding the basics of this may help you make better use of the queuing system and get your jobs processed faster.

When a job is submitted, or when the scheduler has to recalculate priorities for any other reason, it will check if the resources requested by a job are available. If so, it will allocate these resources and start the job. If not (e.g. your job asks for 32 cpu cores and only 20 are available), the scheduler will check the next job in the queue and see if that job can be executed, and so on. This means, that some jobs will have to wait longer than others, especially if they want many or very specific resources. As a first step at more fairness, the scheduler uses a first in, first out policy: jobs that have been in the queue longer, will get higher prority so they will eventually get executed.

But imagine there is a huge job waiting until 64 cpu cores are available. It would be a waste if one of the 64-core nodes cannot start new jobs, and just waits for running jobs to finish. After all, one of these jobs may be taking a couple of days, and all this time, nothing else is going to happen on this node. Luckily, the scheduler has a `backfill` policy. This makes use of limits set in jobs, especially the walltime. So to continue the example used above, where the scheduler needs to have all 64 cores on one node to be available to start a big job. The scheduler has selected a node to be allocated, and it is waiting for one more job to finish. Suppose the job that is running, has specified that it takes a maximum of 48 hours, of which 16 have elapsed so far. And another job in the queue has specified a maximum walltime of 4 hours. Then the scheduler knows, that it can safely execute this new job, and still have all the cores available once the other running job is finished. So your small job gets executed immediately, and a couple of hours later, you have your results. Now suppose you had submitted this exact same job, but without a walltime specification. The scheduler would not have known how long your job would take, assuming a set default for the maximum job duration (e.g. 16 days). So obviously with these settings, the job would not have been started, and the node would have continued to run just the one old job, until that was finished and the new big 64-core job could start. And only after that job has finished (e.g. another 48 hours later), your small job could be reconsidered for execution.

Of course setting a maximum walltime has a downside as well: it really sets the maximum amount of time your job is allowed to take, and the job will be terminated once the maximum is exceeded. So choose wisely!

Time and number of cpu cores are not the only resources that need to be considered; the maximum memory in use by the job is another. If a node has plenty of free cores, but only 1 GB free memory, it cannot start new jobs, unless those jobs have told the scheduler, that they only require a small amount of memory. as with walltime, exceeding your self-imposed limits will be the end of the job, but specifying too much (or specifying no limit, defaulting to unlimited memory use) may mean, that your job has to wait longer for the requested resources to be available.

Disk usage

MPI seems to be very sensitive to disk waits. If a disk becomes unresponsive, MPI jobs hang or crash. Combined with the fact that the normal configuration of our desktop systems only allows 8 simultaneous nfs clients, and becomes less responsive even before that number is reached, there is only one conclusion: The queue system only works, if program and data are local on the disk of one of the para cluster machines; disks of the central servers like ijsselmeer (/disks/strw1 etc) should probably be fine too, albeit slow, but running jobs or accessing data on a disk of a desktop machine is not entirely stable.

We may look into methods to enforce this policy (ie make desktop disks invisible to the para nodes). Until then, use common sense and use the /net/para*/data* data areas for all file access during your program run, including the program itself and any input and output files.

Note: The para cluster disks are there for the data storage in use by the jobs queued on the cluster nodes, they are not intended for general storage purposes!

A logical conclusion is of course, that if everyone needs the local disks on the cluster, users should clean up on those disks as well, or soon we will be out of space. There is a total of 50 TB on those machines, but without frequent cleanups, that will fill up in no time. iIf this becomes a problem, we might put a scratch policy in place on those disks.

Links

For more information about Torque and related software, see:
Torque website + documentation
Maui Cluster Scheduler
man pages of qsub, qstat and other utilities