How to use the cluster with the *Son of Grid Engine*

v 0.9, June 2022

Tobias van Valkenhoef, Dan Dediu

Table of Contents

1. The cluster

This is currently (July 2022) composed of fifty Linux servers, named **gridnode001** to **gridnode052.** All of the nodes run the same software and have 20 or 24 CPU cores and 256 GB of memory. The **gridnode020** is special as it has 4 TB of memory and 96 CPU cores. Also the **gridnode017 and gridnode018** deviate from the default nodes as they include a RTX 8000 GPU for CUDA acceleration. The cluster is accessed with the machines called **gridmaster** and **gridportal1** used for submitting and monitoring jobs. The general-purpose machine **lux20** may also be used for some special purposes (see below). A full listing of the configuration of these machines can be found at the end of this document.

All nodes run the same operating system OpenSuSE 15.2 and have access to the same software also available to the general purpose Linux servers and clients in the form of environment modules. Some extra modules are available only for the cluster because they are optimized to make use of the cluster infrastructure.

Node **lux20** is special in that it has development files installed, which is important if you need to build some software from source. A very useful (and frequent) scenario is when you need an **R library** that is not currently installed. As an example, let's install package `ade4`: first, connect to lux20:

```
> ssh lux20
[TYPE YOUR PASSWORD]
>
```

start R and install the package required (with dependencies):

```
R

R version 4.0.3 (2020-10-10) -- "Bunny-Wunnies Freak Out"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
```

```
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
> install.packages("ade4",dep=TRUE)
```

and select a mirror near you (such as Utrecht in the Netherlands). If this is the first time you manually install a package, R will complain that it can't write to the global library directory (which is absolutely normal) and will ask if you want to use a personal library (say yes). After compilation (assuming everything is fine; if not ask Tobias van Valkenhoef) the new library will be installed and available on all nodes. Don't forget to quit R with `q()` and log off lux20 by typing `exit`.

The **gridmaster** is also special but in a different sense: it hosts the *Son of Grid Engine* (to be discussed in detail below; see also https://arc.liv.ac.uk/trac/SGE). As a general rule, ***don't use lux20, gridmaster or gridportal1 for heavy computational tasks: lux20 is for building software and the gridmaster and pridportal1 are for submitting calculation jobs into the cluster.***

Please note that this configuration (hardware and software) might change in the future; we will strive to keep this document updated.

2. Using the cluster

The point of having this cluster is to use it for computationally demanding tasks which would overwhelm your desktop/laptop machine. Such tasks might require **huge amounts of memory** (on a single node you have 256 Gb live RAM, and on the gridnode020 even 4TB) and/or **heavy computation**. But the main strength of the cluster is that you can **massively parallelize** you task.

However, the golden rule is to **always debug and optimize** your program/script on your local machine before launching 5000 of them and discovering after a month that they spit gigabytes of garbage. Debug by running clever test cases or portions of the actual data. Optimize to save memory and time (for R see useful guidelines: http://www.noamross.net/blog/2013/4/25/faster-talk.html http://stackoverflow.com/questions/7727435/how-can-i-optimize-the-performance-of-my-r-code or http://www.r-bloggers.com/speeding-up-r-computations or simply use a Famous Web Search Engine for more tips), but please see the notes on the use of parallelized code with the Grid Engine below.

Just one more observation before we proceed: **storage**. Normally, any input and output files will be read/written from/to your UNIX home directory (`/home/USERNAME` on Linux and `\\uhomes` or drive `U:` on Windows), which is easy to access and fine as long as you don't read or write huge amounts of data (such as large logfiles or outputs from Bayesian MCMC). However, there's another location which is faster (and does not clog the whole network with heaps of data), has 415TB space, and is accessible at `/data/clusterfs` on all Linux hosts (ask Tobias van Valkenhoef if you can't find it): what I usually do is transfer the input data there, start the job, wait until it is finished and then transfer back the results to my normal working directory.

Now, there's two basic ways to use the cluster: the **recommended** with batch jobs through the Grid Engine, and the **interactive** way by getting a shell directly on one of the node(s).

2.1. The recommended way: use batch jobs with the Grid Engine

This is very easy and you benefit from everything being taken care of transparently (so you don't need to worry about allocating your jobs, etc.). To begin, log in into the **gridmaster** host:

```
> ssh gridmaster
```

or, if you want access to the fancy Grid Engine GUI use[1]

```
> ssh -X gridmaster
```

Of course, if are outside the MPI network simply log in first into `ssh.mpi.nl` (and if you want GUI make sure you also use -X to log into this as well):

```
> ssh ssh.mpi.nl
```

or

```
> ssh -X ssh.mpi.nl
```

Now that you are on the **gridmaster** host, you can launch the fancy GUI of the Grid Engine (if you used -X when ssh'ing):

```
> qmon
```

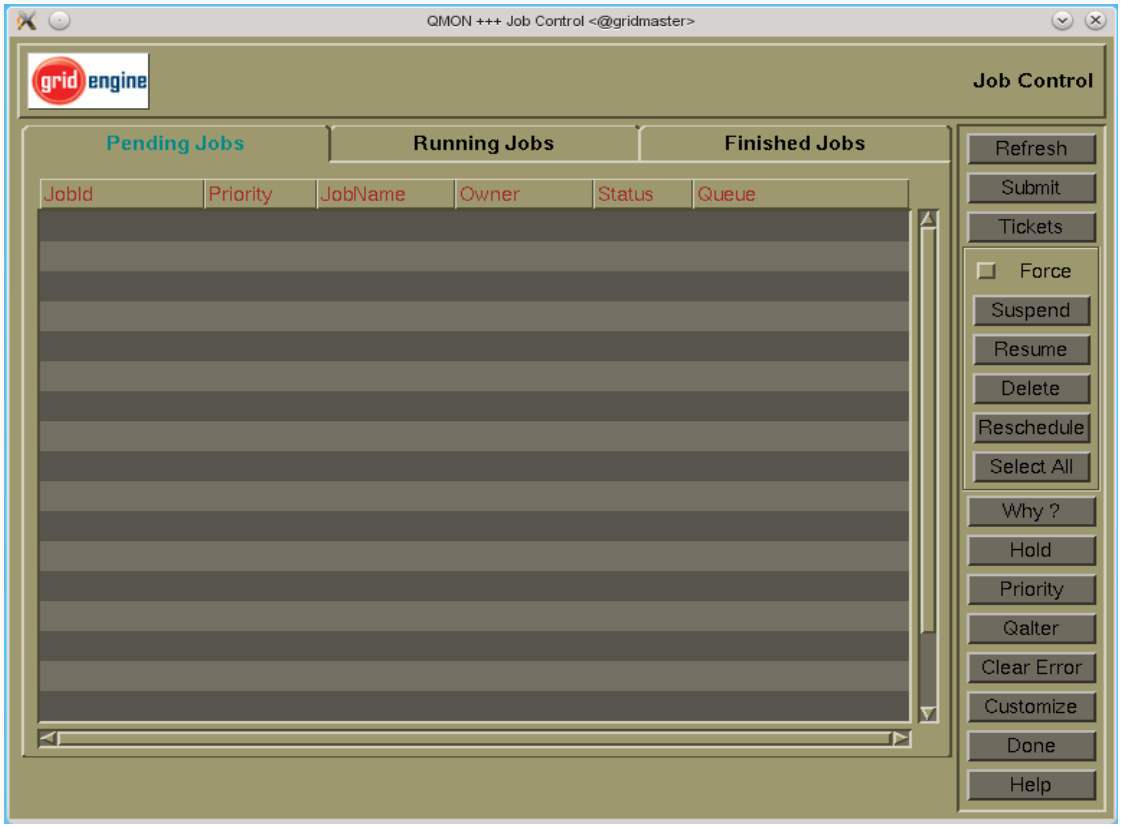
and you should see

If you get a “command not found” error on the **gridmaster** host, make sure you have the right environment, which normally gets loaded automatically on login variables by issuing the following command (note the leading dot '.' and space):

```
> . /opt/sge/default/common/settings.sh
```

If you click on the first button (top left; “Job Control”) you can see the jobs that are still pending, those that are running and those that have already finished:

1 For `ssh -X` (or X11 forwarding), on Windows you can use Xwin-32, Vncviewer or Cygwin/X (only the first two are officially supported); alternatively you can use OpenSuse 15.0 in the VMWare Player (there is an official image that you can use and this allows you to have a virtual Linux machines on your Windows desktop from which to access the cluster, etc.). On Mac OS X (10.9 or newer) you need to manually install Xquartz.



and you can now even submit a new job by clicking “Submit”.
However, probably the best way to go about it is by not using the GUI but some simple commands.

2.1.1. Create your script for the Grid Engine

The idea is to tell the Grid Engine something about your job. As an example, create a file called `test.sge`: in the directory grid-demo right in your Unix home directory (thus, `/home/USERNAME/grid-demo`):

```
#!/bin/sh
#$ -N grid-demo
#$ -cwd
#$ -q single.q
#$ -S /bin/bash
#$ -M youremail@server.com
#$ -m beas

Rscript grid-demo.r
```

The parameters are[2]:

- **N** the name for the job (also used as a prefix for stdin and stdout files, see below)
- **cwd** change working directory so that output goes in the same directory where the job is run
- **q** which queue to use
- **M** your e-mail address here; you'll receive an e-mail when the events defined by **-m** occur
- **m** the events for which e-mails will be sent (begin, end, aborted, suspended)
- **S** run the job in a shell (some contexts you might not need this)

Note: **-M** and **-m** really make sense only for very long jobs if you don't want to keep checking their status manually.

2 For more info on these and other parameters please see for example https://ctbp.ucsd.edu/computing/wiki/introduction_to_sge_the_queuing_system_on_the_clusters

Of course this simply wraps the real work done in the `grid-demo.r` R script which should be placed in the same directory (here `/home/USERNAME/grid-demo`) and which in this case looks like:

```
time.to.finish <- system.time( m <- mean( unlist( lapply( 1:10000, function(i){ rnorm(1); } ) ), na.rm=TRUE ) );
cat( paste( "Hello, World!\nMean is:", m), file="grid-demo-output.txt" );
cat( "I'm done here in ", time.to.finish, "\n" );
```

which does not do anything useful except compute the mean of 10,000 random numbers normally distributed around 0.0. However, it computes the time it takes the script to run, displays it to the standard output (last line), and also writes a stupid message and the computed mean to a file called `grid-demo-output.txt` to be placed in the same directory as the script itself.

Note that the `sge` file can wrap any sort of command and not just R scripts, such as compiled C++ programs, Java applications, python or BASH scripts, etc.

Also important is that this sge file has to be in a UNIX ASCII format. If you created this file on Windows you most likely need to convert it with the command “dos2unix test.sge”.

2.1.2. Tell the Grid Engine to run your job

We tell the Grid Engine to queue the job and run it by issuing the command:

```
> qsub test.sge
```

which should print a message such as:

```
Your job 44078 ("grid-demo") has been submitted
```

and if you mentioned **-M** and **-m beas** options you should also receive an e-mail about the job being started. There should now be at least two new files in the directory: `grid-demo.e44078` and `grid-demo.o44078`. The first one (extension **`.eXXXX`**) contains any **errors** your script might have produced, while the second (extension **`.oXXXX`**) is the **output** (what would usually go to the console, such as the `cat` command on the last line); these files are very important to watch as they might contain essential information on why a job misbehaved and on its progress (if you were careful enough to output such progress information in the first place).
Please note that the actual number 44078 will differ as it represents a unique identifier of a particular job (i.e. if you launch the exact same script at a different time it will have a different unique number assigned to it).

2.1.3. Check the status of your job

To check the status of your jobs please use

```
qstat -q 44078
```

which should produce a list of running (and queued) jobs, if any, such as:

```
job-ID  prior   name       user         state submit/start at     queue                          slots ja-task-ID
-----------------------------------------------------------------------------------------------------------------
  44078 0.00000 grid-demo  deddan       qw    06/20/2014 13:50:00                                    1
```

2.1.4. Your job is done!

Assuming all is well, your job should end (naturally or not) at some point (if you mentioned **-M** and **-m beas** options you should also receive an e-mail about the job being completed or stopped) and it will not appear anymore in the list produced by `qsub`.

Now you should have not only the `grid-demo.e44078` and `grid-demo.o44078` files in the `/home/USERNAME/grid-demo` directory, but also the file `grid-demo-output.txt` that we explicitly created on line 2; in a particular run of this job they contained:

`grid-demo.e44078`:	NOTHING
`grid-demo.o44078`:	I'm done here in 0.045 0 0.045 0 0
`grid-demo-output.txt`:	Hello, World!
	Mean is: -0.00670801927305977

The files .eNNNN and .oNNNN can be safely deleted when the job is done if they don't contain information you might need in the future (otherwise they will tend to accumulate there making it hard to know which are old and which new).

2.1.5. Different queues for different jobs

The jobs you want to submit to the cluster have different characteristics, and to reflect this diversity there are **multiple** queues (as the name suggests, a queue is where your jobs accumulate in order to be sent for processing to the right processor), each appropriate for a particular type of job. Currently these queues are: **single15.q**, **multi15.q**, **fullnode15.q**, **sfari.q**, **cudatest.q**, **cuda.q**. **big.q**, **interactive.q**, and **all.q**.

Single.q

The **single15.q** queue is the default queue for *single threaded batch jobs* (i.e., jobs that do not require a graphical interface to run – that's what batch means – and each is run sequentially on a single processor; – that's what a single thread means; if you don't know what this is please see for example https://en.wikipedia.org/wiki/Thread_%28computing%29#Multithreading) and has been assigned the most compute resources: it can run hundreds of concurrent jobs!

In most cases this is what you want and the system automatically allocates your jobs to this queue (so nothing special for you to do). If you have one (or a couple) jobs to run through the Grid Engine, you can do it manually as described in the above Sections, but if you have **tens** or **hundreds**, the manual option is out.

In this case the smart move is to write a little script (BASH, python, R, whatever you like) that generates `sge` wrappers, one wrapper for eah little job you have. Say you want to compute 10 means of 10,000 normally distributed random numbers: in this case write a small script that generated 10 `test.sge` files, each wrapping (in this case) the same R script (but of course each R script could be slightly different, reading different input data for example), `test01.sge`, `test02.sge`,... `test10.sge`. Then `qsub` each individually or, even better, generate a BASH script that does it for you, something like:

```
#!/bin/sh
qsub test00.sge
qsub test01.sge
qsub test02.sge
qsub test03.sge
qsub test04.sge
qsub test05.sge
qsub test06.sge
```

```
qsub test07.sge
qsub test08.sge
qsub test09.sge
qsub test10.sge
```

Of course, the same advice of debugging applies here as well: don't generate a run with all 10 (thousand) jobs at once, test two first and make sure everything works as expected!

Multi15.q

The **multi15.q** queue is designed for *multi threaded applications*, that is, it allows one application to use at least 10 available cpu cores on a server (an examples of such cases is when you use `mclapply()` from library `parallel` in `R`). Or if your job uses more than 13 GB of memory.

A final word on **parallelism**: for example the R package `parallel` makes it very easy to distribute an identical computation across multiple CPUs using `mclapply` (http://stat.ethz.ch/R-manual/R-devel/library/parallel/html/mclapply.html), which is amazing on a multi-CPU or the multi-core processors such as Core i3, i5 and i7. However, when using the Grid Engine there's a snag: the Grid Engine is a very smart piece of software that can keep track of which jobs run where and allocate them optimally to free CPUs, but is has no way of knowing that one of these jobs forks to multiple CPUs using for example `mclapply`: from the Grid Engine's point of view it is still a single job using one CPU and this mismatch might interfere with its ability to optimally manage the jobs. **Thus the best advice is to use `mclapply` (or other types of parallelism) with the Grid Engine only in the multi15.q submitted to the Grid Engine as explained above.**

Fullnode15.q

The fullnode15.q is used for extremely multithreaded applications that are capable of using all available resources of a compute node. Often these jobs don't play nice with each other when having to time share on a multi15.q node. This queue is only available on special request.

cuda.q/cudatest.q

The cuda.q and cudatest.q are used for applications with CUDA support. This queue is only available on special request.

big.q

The big.q is used for direct access to the big node (gridnode020), useful when your job requires more than 256 GB of RAM. This queue is only available on special request and for temporary use.

Interactive.q

The **interactive.q** is a dedicated queue for managing *interactive jobs.*

Sfari.q

The **sfari.q** is a dedicated queue for the Sfari project not available for others.

All.q

The **all.q** is normally not accessible and only used for testing purposes.

2.1.6. A note on computational performance using R

R comes with a standard library for doing linear algebra (aka **BLAS** https://en.wikipedia.org/wiki/Basic_Linear_Algebra_Subprograms) which is essential for many of the computations R has to do. However, this standard implementation of BLAS is quite slow, and there are alternatives such as Intel's MKL (https://software.intel.com/en-us/intel-mkl), ATLAS (http://math-atlas.sourceforge.net/) and **OpenBLAS** (http://www.openblas.net/), which result in dramatic jumps in performance, especially for some types of computations (e.g., http://www.brodrigues.co/2014/11/11/benchmarks-r-blas-atlas-rro/ or http://edustatistics.org/nathanvan/2013/07/09/for-faster-r-use-openblas-instead-better-than-atlas-trivial-to-switch-to-on-ubuntu/ or http://brettklamer.com/diversions/statistical/faster-blas-in-r/).

While, in principle, using these alternative BLAS implementations should have no visible effects, it *might* happen that there are compatibility issues with libraries, so it is important to keep this in mind and do some testing and internet searches before you decide to use such alternative libraries (anecdotally, I [DD] have been using OpenBLAS with my R installations at home [Ubuntu 14.04, CentOS 7, OpenSuse 13.1] for about a year now and I have seen no ill effects so far).

Luckily, besides the fully "vanilla" **R** (using the built-in BLAS) which is accessed using the command

```
> R
```

on the cluster, there's also an implementation that uses OpenBLAS which is invoked using the command

```
> R-OpenBLAS
```

From your point of view, it is completely transparent which BLAS R is using (the behaviour is the same), but this is much faster. However, please note that it may result in **"hidden" multithreading** in the sense that OpenBLAS by default uses several cores in parallel. This is important as you may want to avoid overloading the CPUs without the Grid Engine realizing it (see **section 2.1.5** about the proper way of dealing with multithreading).

2.2. The interactive way.

The interactive way will give you an active shell on one of the gridnodes. This is especially useful when you're testing your scripts or have a big job requiring interactive access. (Matlab).

So, how do you do it? Simple: log in to the gridmaster

```
> ssh gridmaster
```

And issue the following command:

```
> qrsh -q interactive.q
```

This will give you a shell on a random gridnode with a free interactive.q job slot. You can choose any queue where you have permissions to with the exception of the cuda.q at the moment. If you want to select a specific node use the following command:

```
> qrsh -q multi15.q@gridnode040.mpi.nl
```

It will give you a shell on the gridnode040 if there are job-slots available on that server.

However, please note that as soon as you disconnect from the node (say because of a network problem or because you shut down your own computer) the task **stops**! To prevent this you can use the `screen` command as follows: ssh into **gridmaster** (either from you machine or by first ssh'ing into **ssh.mpi.nl**) and start

```
> screen
```

This will create a virtual terminal into which you can run your program in the usual way:

```
> qrsh -q interactive.q
```

except that now you can close the connection and/or shut down your computer and the task continues to run. To check on its status ssh back into **gridmaster**, list the available screens with:

```
> screen -list
```

pick the one you want, and reconnect to it using its unique identifier:

```
> screen -r rNNNN
```

Now you see the progress of your task and any output message it might have generated. When the task is done don't forget to close the screen session by simply existing from it:

```
> exit
```

For more info on `screen` please see http://www.rackaid.com/blog/linux-screen-tutorial-and-how-to/ http://www.mattcutts.com/blog/a-quick-tutorial-on-screen/ or http://www.howtoforge.com/linux_screen

Please note that the interactive sessions take up job slots on the cluster even if they are idle. So please close the interactive sessions after you finished with it. Systeadmins might kill of your interactive sessions not on the interactive.q when idling for over a week.

3. Contact info

For more info, questions, suggestions and warnings on launching big jobs please contact Tobias van Valkenhoef (Tobias.vanValkenhoef@mpi.nl),or helpdesk@mpi.nl.