OPEN GRID SCHEDULER SOFTWARE

using Gemini cluster, version September 2022

Frank van de Wiel,
ICTBèta

Table of contents

Open GRID Scheduler / Grid Engine 2011.11

On Gemini compute cluster there is a version of Open Grid Scheduler software installed. It is a distributed resource management (DRM) system which allows users workload in the form of batch files being distributed over available compute nodes.
The system determines on conditions like job queues when and how a job will be executed. Some queues are only available for particular research groups and running on their own compute nodes. At this moment there is no GPU available for compute jobs. Jobs submitted via qsub are non - interactive.
For jobs running for a long time it is advised to write intermediate results to a file and/or make use of checkpoints in your program. In that case system downtime necessary for system maintenance or hardware/network failure lets you continue your job(s) from the last saved intermediate file(s) or checkpoints and not have to start your job all over again.
As a bonus for running jobs via GRID job scheduling your jobs will run with a higher priority than interactive jobs do.

Who can submit jobs?

Every one with an account on Gemini is allowed to submit jobs to one or more GRID queues. Some queues are restricted only to members of specific research groups like ITF.
By default the environment module **sge/2011** is loaded when you are logged in on Gemini. This allow you to immediately use the grid scheduling software without any further actions by you.

Gemini specifications

Gemini is running Scientific Linux 7.9 on DELL PowerEdge R730 (2x) and R640 (6x).
Gemini-cluster has 320 CPU slots.

Preparing your program

Before submitting your program to a job queue you may need to answer one or more of the following questions.

- What queue to use?
- How long do I expect my program to run, for jobs running more than 24 hours you may need to take Kerberos tickets into account when accessing /science/projects/... location.
- Export environment variables to job queue if necessary
- What environment module do I need (if any)?
- What amount of memory do I need (available memory is not unlimited) ?
- How must storage is necessary for my output?
- Where should the output be saved, home directory, project space or local scratch space?
- What type of job (serial / parallel)?
- Do I want an email message from job system?
- Do I need intermediate files and/or checkpoints in my program to be able to continue after system maintenance or unexpected network/hardware failures?

Qsub (job submitting)

Qsub, the program to submit a job script has many options which allows you to set one or more SGE (Sun Grid Engine) directives. See man qsub for more information.

Here are some frequently used SGE directives

```
-V                # export your private environment variables to job system
                    PS> This will NOT export aliases.
-S /bin/bash      # run with this shell

-cwd              # run job using files in current working directory
-N job name       # name used for qstat

-q "queue name"   # submit to queue "queue name" e.g. all.q
-l h_rt=50:00:00  # need 50 hours runtime
-l mem_free=2G    # need 2GB free RAM

-pe mpich 4       # define parallel env and request 4 CPU cores

-o job_name.out   # name of output file
-e job_name.err   # name of error file
-o job_name.outerr -j y    # combine into one output file both -o and -e

-M user@domain    # use this <email> address
-m beas           # send a mail when job begins (b), ends (e), aborts (a) or suspends (s)
```

Submitting a job via the command line

In this example we create a very simple script just outputting the text "Hello World" and save it to the file *world*.

Create a file with content:

```
echo "Hello World"
```

and save it with the name *demo*

To submit this script to the job system enter the following command:

```
 $ qsub -q all.q -V -cwd -N hello -o world demo <return>
Your job 101089 ("hello") has been submitted
```

The result of this script after being executed is saved to the file ***world***.

Submitting job with a script

You can also use a script file to provide qsub with all the SGE directives you need and

call your program.

When you submit a job script, it is interpreted by both **bash** and **open grid scheduler**.

This is how is your script processed:

Want to use a bash command	don't use any prefix
Need a SGE directive	prefix it with characters **#$**
Want a comment	prefix it only with a **#**

Similar "Hello World" job example as above but now send via a script file.
Adding extra SGE directives **-M** and **-m** lets you receive an email when your job has finished.
Also prints start and stop time when job finishes and compute resources and environment settings used. Save this following lines to a file called *world.sh*

```
#!/bin/bash

# echo "Hello World" to a file "world"

#$ -V
#$ -cwd
#$ -o world
#$ -N hello

#$ -M user@domain
#$ -m e

echo STARTED at $(date)
echo "Hello World"

# Used resources
qstat -j $JOB_ID | awk 'NR==1,/^scheduling info:/'

echo FINISHED at $(date)
```

Submit this script with:

```
 $ qsub -q all.q world.sh <return>
Your job 101092 ("hello") has been submitted
```

After this job has finished you will receive an email and output is saved to the file ***world***.

Using environment modules for additional software

For many software packages "environment modules" are available on Gemini. You may have to load such a module before submitting your job to the queueing system.

Check what modules are installed on gemini:

```
 $ module avail
```

Example submitting a julia job via command line:

```
 $ module load julia/1.4      load environment settings for using julia

 $ qsub -V -m e -M solis-id@uu.nl -cwd -N testjulia your-julia-script
```

Explanation of command line options:

```
-V                          export private environment variables
-m e                        send email at end of job
-M user@domain              use email address
-cwd                        read and write files from current working
                            directory
-N testjulia                name of the job is testjulia
your-julia-script           name of julia script to execute
```

Remove this module if you don't want to use it anymore:

```
$ module remove julia/1.4
```

GRID environment variables

When a job starts, many SGE environment variables are set and will be available to your job script. To see what environment variables are available to you run the following script. Save this script as *environment.sh*

```
#!/bin/bash

#$ -V
#$ -o environment.outerr -j y
#$ -cwd

#$ -M user@domain
#$ -m e

printenv > environment
```

Run with:

```
$ qsub -q all.q environment.sh
```

A list of all environment variables available to you is saved to the file ***environment***

You will receive an email when the job is finished.

Here are some of the SGE variables you will see:

- SGE_O_WORKDIR - The working directory of the job submission command
- SGE_O_HOME - The home directory path of the job owner on the host from which the job was submitted
- SGE_O_LOGNAME - The login name of the job owner on the host from which the job was submitted
- SGE_O_PATH - The content of the PATH environment variable in the context of the job submission command
- SGE_O_HOST - The host from which the job was submitted
- SGE_O_SHELL - The content of the SHELL environment variable in the context of the job submission command
- SGE_STDERR_PATH – The path name of the file to which the standard error stream of the job is diverted. This file is commonly used for enhancing the output with error messages from prolog, epilog, parallel environment start and stop scripts, or checkpointing scripts.
- SGE_STDOUT_PATH – The path name of the file to which the standard output stream of the job is diverted. This file is commonly used for enhancing the output with messages from prolog, epilog, parallel environment start and stop scripts, or checkpointing scripts.

Job queues

Display a list of available queues on gemini.

```
 $ qconf -sql
all.q
itf-fat.q
itf.q
long.q
test.q
```

all.q	queue for jobs with max run time of 24 hour (available to everyone)
long.q	queue for jobs with run time of 24 hours or longer (available to everyone)
itf.q and **itf-fat.q**	only available to an ITF user list
test.q	only available to a test user group

Job queue specifications

Display properties of a specific queue

```
 $ qconf -sq all.q

qname                 all.q
hostlist              @shorthosts
seq_no                0
load_thresholds       np_load_avg=1.75
suspend_thresholds    NONE
nsuspend              1
suspend_interval      00:05:00
priority              0
min_cpu_interval      00:05:00
processors            UNDEFINED
qtype                 BATCH
ckpt_list             NONE
pe_list               mpich openmpi
rerun                 FALSE
slots                 48,[science-bs36.soliscom.uu.nl=32]
tmpdir                /tmp
shell                 /bin/bash
prolog                NONE
epilog                NONE
shell_start_mode      unix_behavior

<… skipped output>
```

Check queue status

A script **check_queue** is available to display information about running and pending jobs in a specific job queue. Call the script **check_queue** and you will see all options.

```
 $ check_queue

check_queue [all | test | long | itf | itf-fat] [0 | 1]

0 : pending jobs ; 1 : running jobs
```

Check queue information about running jobs in long.q

```
 $ check_queue long 1

Show Running 1
```

```
queuename                      qtype resv/used/tot. load_avg arch          states
---------------------------------------------------------------------------------
long.q@science-bs40.soliscom.u BP    0/0/48         47.94    linux-x64     d
---------------------------------------------------------------------------------
long.q@science-bs41.soliscom.u BP    0/0/48         0.01     linux-x64
---------------------------------------------------------------------------------
long.q@science-bs42.soliscom.u BP    0/0/24         18.73    linux-x64
Total Running Jobs : 0
```

Qstat state codes

See table below for many qstat state codes.

Category	State	SGE Letter Code
Pending	pending	qw
	pending, user hold	qw
	pending, system hold	hqw
	pending, user and system hold	hqw
	pending, user hold, re-queue	hRwq
	pending, system hold, re-queue	hRwq
	pending, user and system hold, re-queue	hRwq
Disabled	queue disabled	d
Running	running	r
	transferring	t
	running, re-submit	Rr
	transferring, re-submit	Rt
Suspended	job suspended	s, ts
	queue suspended	S, tS
	queue suspended by alarm	T, tT
	all suspended with re-submit	Rs, Rts, RS, RtS, RT, RtT
Error	all pending states with error	Eqw, Ehqw, EhRqw
Deleted	all running and suspended states with deletion	dr, dt, dRr, dRt, ds, dS, dT, dRs, dRS, dRT

Using (temporary) data storage

There are three places to put your data: your home dir, scratch space and project space.

Home dir

For your jobs you can use your home directory to read and save files to but for large datasets you may have some need of more temporary disk space when running your jobs or to store intermediate results.
If you don't use the directive **#$ -cwd** by default output is read from and saved into your home directory.

Scratch space

Each compute node has its own local disk **/scratch** which is not shared with other compute nodes. This location can be used for temporary file storage of your compute jobs.
You might also wish to have your job script check the *available space* first in order to avoid "File system full" or "No space left on device" errors. Here's a simple script that prints the available space in **/scratch** connected to the queue you want to use.

```
#$ -S /bin/bash
#
# Test for available disk space in /scratch
#
#$ -cwd
#
# Email job information
#$ -M "name"@uu.nl
#$ -m e

scratchdir=/scratch
freespace=`df -h $scratchdir | tail -1 | awk 'END {print $4}'`

echo "Available diskspace in /scratch on $HOSTNAME is $freespace"
```

Save this script as **scratch.sh** and start this script with:

```
$ qsub -q "queue name" scratch.sh
```

You will receive an email message when this job is finished. The output is saved in a file with a name like ***scratch.sh.o101069***.
In a similar way you can check available disk space on other locations.

Don't use /tmp to save files for your compute jobs. If **/tmp** is full it may result in a not responding or crashing compute node.

Project space

For large data files a project space may be available or created on request to you or your research group (/science/projects/…). In order to access "science"-project directories, you may need to first run '**kinit**' (for Kerberos authentication).

MATLAB

If you want to run jobs using MATLAB code, the accepted practice is to compile your MATLAB *.m* code to a native executable using the MATLAB compiler *mcc* and then submit that code, along with your data to an SGE queue. This approach does not require a MATLAB license.
Before you use any MATLAB utilities, you will have to load the MATLAB environment via the 'module' command

```
$ module load MATLAB/2020
```

Checkpoints

Check pointing is a facility to save the complete status of an executing program or job and to restore and restart from this so called checkpoint at a later point of time if the original program or job was halted, e.g. through a system crash.

See for more information: **man checkpoint**

Parallel jobs submit scripts

Submitting parallel jobs is very similar to submitting single node jobs (as shown above). A parallel job needs a **pe** (parallel environment) assigned to the script.
The important option is the **-pe** directive in the submit script. This variable must be set for the MPI environment for which you compiled your program.

The following parallel environments are defined:

mpich
openmpi
smp

These environment modules are available for parallel computing:

mpi/compat-openmpi16-x86_64
mpi/mpich-x86_64
mpi/openmpi-x86_64
nvhpc/20.9 # NVidia compiler set

Open MPI will automatically detect when it is running inside SGE and will just "do the Right Thing." Specifically, if you execute an mpirun command in a SGE job, it will automatically use the SGE mechanisms to launch and kill processes. There is no need to specify what nodes to run on — Open MPI will obtain this information directly from SGE and default to a number of processes equal to the slot count specified.

Example openmpi script

```
#!/bin/bash
# Replace < ... > with your settings and remove both < and >

#$ -N <job_name>
#$ -S /bin/sh
#$ -o <output file>
#$ -e <error file>
#$ -M <user@domain>
#$ -m es

# Execute the job from  the  current  working  directory
#$ -cwd

# Parallel environment request
#$ -pe openmpi

# Choose your queue
#$ -q long.q

# Load your environment module
module load mpi/openmpi-x86_64

# Start your program
#
mpirun -np $NSLOTS <put your code here>
```

Appendix

A simple compilation job with qsub via the command line.

```
$ qsub –q all.q –S /bin/bash –m e –M user@domain –cwd<enter>
gcc –o hello hello.c    <enter>
ctrl-D

Your job 587 ("STDIN") has been submitted
bash$
```

Description of qsub options:

-S /bin/bash	Use bash shell as command interpreter
-m e **-M** user@domain.nl	Send email message at end of job make sure your email address is valid)
–cwd	Use current working directory to read or write file. If you ommit **–cwd** then **$HOME** will be used as the working directory.

The file source file `hello.c` must be located in current working directory.

The batch system starts compilation of your job and at the end you will receive an email message. Your compiled program **hello** will be placed in your current working directory.

Use **check_queue all 1** to view queue status

Common problems

Program is not found by queue system.

- Check that you have loaded an environment module for your program (if needed)

before submitting your job (**module load "name of module"**)

- Use **#$ -V** in your job script or **qsub –V** submit command to export your environment settings to job queue system.

- Check **PATH** settings in your job script.

You have not received any email from job system

- Please check if the email address in your job-submission is valid!

Why won't my jobs run ?

There are several reasons why a job will not run. The first reason is due to the job resource requirements. It is possible that the cluster is full and you have to wait for available resources (processors etc.)

It is also possible the job may have experienced and error in the run script. In which case the status would be "Eqw". You can query a job's status by entering the following:

```
qstat -explain c -j _Job-ID_
```

where _Job-ID_ is the Grid Engine job number.

Fixing qsub errors

Occasionally, a script may stop and put your job into an error state. This can be seen by the qstat **state** output:

```
$ qstat -u '*'

job-ID  prior   name       user         state submit/start at     queue slots ja-task-ID
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   6868 0.62500 simple.sh  user1     E     06/08/2009 11:29:02 all.q
                                           ^^^
```

the E (^) means that the job is in an ERROR state.

You can either delete your job with qdel

```
qdel <Job ID> # deletes the job
```

or change it's status with the qmod command.

```
qmod -cj <Job ID> # clears the error state of the job
```

Support questions

ADDRESS QUESTIONS TO : gemini-admin@science.uu.nl

Please include the following information in your support request:

- Cluster name
- Job queue
- Job ID
- Submit/start time of job
- Contents of job submission script
- Attach output (log) files containing error descriptions.
- Provide commands that you were executing.
- When did the problem happen?

Getting help, support request sample

```
To: gemini-admin@science.uu.nl
Subject: Job 123456 gives errors on the Gemini cluster

Hello:

my name is User, user name user1. Today at 10:00 am, I submitted a job
123456 on the all.q on Gemini cluster. The Job script is located
$HOME/script/path. I have not changed it since submitting my job to
all.q.

A list of the following modules were loaded at the time follow:

module list
Currently Loaded Modulefiles:
  1) sge/2011    2) ferret/7    3) idl/8.6     4) julia/1.4

The job ran quickly and the myjob-123456.out and myjob-123456.err files
were created. There was no output in the myjob-123456.out file but
there was an message in the myjob-123456.err output

[user1@gemini scheduling]$ cat myjob-123456.err
error: *** JOB 123456 ON cdr692 CANCELLED AT 2018-09-06T15:19:16 DUE TO
TIME LIMIT ***

Can you tell me how to fix this problem?
```

Further documentation about using GRID engine

manual pages:

```
man sge_intro  - a facility for executing UNIX jobs on remote machines
man qsub       - submit a batch job to Grid Engine
man qdel       - delete batch jobs
man qhost      - show the status of Grid Engine hosts, queues, jobs
man checkpoint - Grid Engine checkpointing environment configuration file format
```

https://gridengine.eu/

https://gridengine.eu/mangridengine/manuals.html

http://gridscheduler.sourceforge.net/

https://www.open-mpi.org/faq/?category=running