Gtiit-Cluster User Guide

**Introduction**

Gtiit-Cluster nodes run CentOS 7. Regardless of your research workflow, **you'll need to master Linux basics** to use the system properly. This user guide does not address these topics. However, if you encounter a term or concept in this user guide that is new to you, a quick internet search should help you resolve the matter quickly.

**NOTE: During the cluster open test, there will a weekly maintenance on every Thursday afternoon 2-5pm. Please avoid running jobs in this period.**

**System Overview**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Seq** | **Node** | **hostname** | **Public IP** | **Private IP** |
| 1 | Mgmt Node - Frontend | einstein.gtiit.edu.cn | 10.150.0.101/24 | 10.150.6.65/26 |
| 2 | Mgmt Node - login | login-0-0.local | 10.150.0.102/24 | 10.150.6.126/26 |
| 3 | GPU Node1 | compute-0-0.local | N/A | 10.150.6.124/26 |
| 4 | GPU Node2 | compute-0-1.local | N/A | 10.150.6.123/26 |
| 5 | Compute Node1(192GB) | compute-0-2.local | N/A | 10.150.6.122/26 |
| 6 | Compute Node2(256GB) | compute-0-3.local | N/A | 10.150.6.121/26 |
| 7 | Compute Node3(512GB) | compute-0-4.local | N/A | 10.150.6.120/26 |
| 8 | Compute Node4(192GB) | compute-0-5.local | N/A | 10.150.6.119/26 |

**Accessing the System**

The "ssh" command (SSH protocol) is the standard way to connect to Gtiit-Cluster. SSH also includes support for the file transfer utilities scp and sftp. [Wikipedia](https://en.wikipedia.org/wiki/Secure_Shell) is a good source of information on SSH. SSH is available within Linux and from the terminal app in Mac OS. If you are using Windows, you will need a SSH client that supports the SSH-2 protocol: e.g. [Bitvise](http://www.bitvise.com/), [OpenSSH](http://www.openssh.com/), [PuTTY](http://www.putty.org/), or [SecureCRT](https://www.vandyke.com/products/securecrt/). You can initiate a ssh session using the following command:

$ ssh USERNAME@10.150.0.102

For the first time user (or whenever you want to change your password), you need to connect to the frontend node **10.150.0.101.** Changing initial password is enforced for user on the first-time login. After changing the password, you may want to wait for a few minutes before connecting to the login node **10.150.0.102**. For general use of the cluster, you should always connect to the login node **10.150.0.102**.

To connect with X11 support (usually required for applications with graphical user interfaces), use the "-X" or "-Y" switch:

$ ssh -Y USERNAME@10.150.0.102

**Avoid computationally intensive activity on login node or frontend node**. This means:

* Don't run research applications on the login nodes.
* Don't launch too many simultaneous processes for compiling code and visualization
* That script you wrote to check job status should probably do so every few minutes rather than several times a second.

**Using the System**

The Gtiit-cluster is controlled by SGE (Sun Grid Engine Software) that organizes the queues and resources. You need to submit your job script (which is a shell script) in login node and Sun Grid Engine will take care of the rest (as long as there is no error in your script).

SGE will do the "**job scheduling**". That means you can submit all your jobs and SGE will queue them and run them when resources you requested become available. SGE will also achieve "**load balancing**" which means, if one node has been loaded with many jobs, new jobs will be distributed to other nodes. In addition, SGE will allow you to do "**job monitoring and accounting**" which will be useful when you want to check if your job is running, and if it failed it will help you understand what went wrong.

Within the next sections, we will show how to use Sun Grid Engine for **job submission**, **monitoring**, and **troubleshooting**.

Job submission

QuickStart:

Login to login node: ssh [USERNAME@10.150.0.102](mailto:USERNAME@10.150.0.102).

$ qsub -cwd -V -l h\_rt=00:10:00 myjob.sh

-cwd: Run the job from the current working directory (default: Home directory).  
-V: passes all variables (-v var Pass the variable var).

-l h\_rt=00:10:00: request 10 minutes run time for the job.

Some other commonly used command line options are:

-o: file for standard output  
-e: file for standard error output  
-b y: If the value of -b is 'y', then command may be a binary or script. If the value of -b is 'n' then command needs to be a script and it will be handled as script. (default: -b 'y')

The submitted script file myjob.sh looks like the following

#!/bin/bash

#$ -q serial.q

./your\_commands

The option -q specifies the job queue you want to use (which is serial.q in this example). The last line “./your\_command” is the program you want to run.

Enforce time limit on a job:

In SGE, time limit is referred to as h\_rt (hard run-time), and is requested as a resource, using the -l switch to qsub. For example, to submit a job with a runtime of only 10 minutes:

$ qsub -q long.q -pe smp 2 -l h\_rt=80:00:00 myJob.sh

When h\_rt is not specified, SGE assumes h\_rt=00:00:00 (you need to specify a time limit in order to run it). Since h\_rt is a hard limit, any job that exceeds the requested run-time will be killed by SGE. To be safe, you should use checkpointing so if your job's runtime is underestimated and killed prematurely, you can restart it with no work lost. Since this job asked for 80 hours run time, it should be sent to a queue that allows long run time which is long.q in the system

Specifying resources:

If your program requires special resources to run, you can request resources following the example below.

$ qsub -q large.q -pe smp 4 -l “h\_vmem=4G,h\_rt=05:00:00“ myJob.sh

-pe <num\_cores>: Run in **smp** environment which is for shared-memory jobs with multi-threading (default: one cores per job).

-l h\_vmem=<size>: Specify the amount of maximum memory required. (Note: This is memory per process slot. e.g., total memory for the example command above will be 2\*4G. default: h\_vmem=2G in Gtiit-Cluster).

Since this job required smp parallel environment with 4 cores, is should be sent to a queue with parallel support which large.q in the system.

The resource attributes are defined as complex attributes in SGE. Use qconf -sc to show the complex configuration:

$ qconf -sc

#name shortcut type relop requestable consumable default urgency

#--------------------------------------------------------------------------------------

arch a STRING == YES NO NONE 0

calendar c STRING == YES NO NONE 0

cpu cpu DOUBLE >= YES NO 0 0

display\_win\_gui dwg BOOL == YES NO 0 0

gpu gpu INT <= YES JOB 0 0

h\_core h\_core MEMORY <= YES NO 0 0

h\_cpu h\_cpu TIME <= YES NO 0:0:0 0

The entry can be used in a qsub -l resource request if the **requestable** field is set to 'y' or 'yes'. If set to 'n' or 'no' this entry cannot be used by a user in order to request a queue or a class of queues. If the entry is set to 'forced' or 'f' the attribute has to be requested by a job or it is rejected.

When a job requests a **consumable** resource, the requested number will be subtracted from total available value. If not enough resource available, the job will be queued. In Gtiit-Cluster, we only defined host level consumable resource.

To view the defined consumable resource for a compute node, use the command below:

$ qconf -se compute-0-0 | grep complex

complex\_values h\_vmem=187.2G,slots=72,gpu=2

Specifying multiple resources:

Multiple resources can be requested simultaneously in -l option, just use “” to group them together (like -l “gpu=1,h\_vmem=4G,h\_rt=00:50:00”).

Specifying resources in script:

You can add qsub options to the beginning of your job script instead of providing as command line arguments. Take above example, the contents of myJob.sh would look like:

#!/bin/bash

#$ -q large.q

#$ -pe smp 4

#$ -l “h\_vmem=4G,h\_rt=00:30:00”

./your\_commands

Specifying GPU resources:

The **gpu.q** is dedicated to GPU resources. And you also need to specify how many GPU card you need. For each GPU node (compute-0-0 and compute-0-1), there are two Nvidia GPU card inside. Your qsub command should looks like:

$ qsub -q gpu.q -l gpu=1 gpujob.sh

Run MPI jobs:

qsub command option -pe mpi <number\_of\_processes> is used to specify the MPI **p**arallel **e**nvironment. The submit script “my\_mpi\_job.sh” for an MPI job is shown below

#!/bin/bash

#$ -q large.q

#$ -cwd

#$ -V

#$ -pe mpi 4

#$ -o output.my

#$ -e error.my

/opt/openmpi/bin/mpirun $HOME/mpi/my\_mpi\_job

You can then simply type qsub my\_mpi\_job.sh to submit the job. The -o output.my and -e error.my specified standard output (-o) and standard error (-e) files.

Viewing a job’s output

SGE creates stdout and stderr files in the current working directory (qsub -cwd) or user’s home directory (default). If any additional files are created during a job’s execution, they will also be located in the job’s working directory unless explicitly saved elsewhere.

For each job, SGE automatically creates two files: <job\_name>.o<job\_id> and <job\_name>.e<job\_id>. The **o** stands for standard output and **e** stands for standard error. The job name is the script name if not specify in qsub (qsub -N <job\_name>).

Interactive session

If you want to run an interactive session using software such as R, you need to use qrsh to start an interactive session and similar to qsub you need to ask for specific resources. In addition, if your software needs X11 forwarding (for example you will need this for R plots) you need to ssh to login-0-0 with -Y option first. The following qrsh command request 2 cpu cores and 5G per core maximum memory (2\*5=10G max memory):

$ qrsh -q interactive.q -pe smp 2 -l “h\_vmem=5G,h\_rt=10:00:00,high”

The last resource “high” is a job priority marker that can only be used in interactive and debug queues.

The large memory node (**compute-0-4** [512G]) can be requested by:

$ qrsh -q interactive -pe smp 2 -l "h\_vmem=5G,h=compute-0-4,h\_rt=10:00:00,high"

If you want to use X11 graphic interface on the large memory node (**compute-0-4** [512G]), you can use qsh instead of qrsh command as shown in the following

$ qsh -q interactive -pe smp 2 -l "h\_vmem=5G,h=compute-0-4,h\_rt=10:00:00,high"

Monitoring jobs in the queue

$ qstat

job-ID prior name user state submit/start at queue slots ja-task-ID

-----------------------------------------------------------------------------------------------------------------

6 0.55500 simple.sh cadmin r 10/15/2019 13:58:17 high.q@compute-0-3.local 10

7 0.00000 simple.sh cadmin qw 10/15/2019 13:58:21 10

From this output, we can see that the job is in the **qw** state which stands for queued and waiting. After a few seconds, the job state will become **r**, or running, which indicates the job is running. Once the job has finished, the job will be removed from the queue and will no longer appear in the output of "qstat" command.

By default, only users’ own jobs are showed. You can use the -u "\*" option to show all user jobs. Add the -q <queue\_name> to filter out jobs in a particular queue.

$ qstat -u "\*"

job-ID prior name user state submit/start at queue slots ja-task-ID

------------------------------------------------------------------------------------------------------------

29 0.55500 simple.sh cadmin r 10/21/2019 14:17:53 long.q@compute-0-3.local 10

-f option can be used to request a "full" display. Example output show as below:

$ qstat -f -u "\*"

queuename qtype resv/used/tot. load\_avg arch states

---------------------------------------------------------------------------------

gpu.q@compute-0-0.local BIP 0/0/72 0.01 lx-amd64

---------------------------------------------------------------------------------

gpu.q@compute-0-1.local BIP 0/0/72 0.01 lx-amd64

---------------------------------------------------------------------------------

long.q@compute-0-2.local BIP 0/0/72 0.01 lx-amd64

---------------------------------------------------------------------------------

long.q@compute-0-3.local BIP 0/10/72 0.01 lx-amd64

29 0.55500 simple.sh cadmin r 10/21/2019 14:17:53 10

long.q@compute-0-4.local BIP 0/0/72 0.01 lx-amd64

---------------------------------------------------------------------------------

long.q@compute-0-5.local BIP 0/0/72 0.01 lx-amd64

############################################################################

- PENDING JOBS - PENDING JOBS - PENDING JOBS - PENDING JOBS - PENDING JOBS

############################################################################

30 0.00000 simple.sh cadmin qw 10/21/2019 14:22:41 10

Deleting a job

In case your submitted job is in error state. To delete a job with the job identifier job-id, execute

$ qdel job\_id

Trouble Shooting

Pending jobs

Sometimes a pending job is obviously runnable but does not get dispatched. SGE can be asked for the reason:

$ qstat -j JOB\_ID

scheduling info: queue instance "gpu.q@compute-0-0.local" dropped because it is temporarily not available

queue instance "gpu.q@compute-0-1.local" dropped because it is temporarily not available

queue instance "serial.q@compute-0-0.local" dropped because it is temporarily not available

queue instance "serial.q@compute-0-1.local" dropped because it is temporarily not available

cannot run in queue "long.q" because it is not contained in its hard queue list (-q)

cannot run in queue "large.q" because it is not contained in its hard queue list (-q)

Job or queue errors are indicated by an uppercase "E" in the qstat output. You can query job error reason

$ qstat -j JOB\_ID | grep error

or query queue error reason

$ qstat -explain E

Available job queues

There are 5 job queues currently available in the system, and they should be used for different purposes.

Serial.q is for serial programs. Only 1 cpu core can be requested in this queue, and the maximum run time is 48 hours.

Large.q is for parallel programs that require multiple cpu cores. The maximum number of cores a job can request is 64, and the maximum run time is 48 hours.

Long.q is for long jobs. The maximum number of cores a job can request is 32, and the maximum run time is 96 hours.

Gpu.q is for qpu jobs. The maximum run time is 48 hours.

Debug.q is for debugging programs. Jobs in this queue are prioritized by system job scheduler, but job size is very limited in here (maximum 2 cores, maximum run time 30 minutes).

Interactive.q is for interactive jobs. It has the similar high priority in the system but limited resources (2 cores and 72 hours maximum).

All users are advised to only submit necessary jobs. System job scheduler will penalize excessive job submission behavior by lowering the running priority of your jobs.

Monitoring cluster usage

Use the qhost command:

$ qhost

HOSTNAME ARCH NCPU NSOC NCOR NTHR LOAD MEMTOT MEMUSE SWAPTO SWAPUS

----------------------------------------------------------------------------------------------

global - - - - - - - - - -

compute-0-0 lx-amd64 72 2 36 72 0.01 187.2G 1.8G 1024.0M 0.0

compute-0-1 lx-amd64 72 2 36 72 0.01 187.2G 1.8G 1024.0M 0.0

compute-0-2 lx-amd64 72 2 36 72 0.01 187.2G 1.9G 1024.0M 0.0

compute-0-3 lx-amd64 72 2 36 72 0.01 251.2G 2.2G 1024.0M 0.0

compute-0-4 lx-amd64 72 2 36 72 0.01 502.2G 3.6G 1024.0M 0.0

compute-0-5 lx-amd64 72 2 36 72 0.01 187.2G 1.9G 1024.0M 0.0

The output shows the hostname (**HOSTNAME**), architecture (**ARCH**), number of cpus (**NCPU**), number of cpu sockets (**NSOC**), number of cpu cores per socket (**NCOR**), number of threads (**NTHR**), the last 5 minutes load (**LOAD**), total memory (**MEMTOT**), and currently used memory (**MEMUSE**), swap space (**SWAPTO**) and used swap (**SWAPUS**) for each node.

To show all resource usages for hosts, use the qhost -F command:

$ qhost -F

HOSTNAME ARCH NCPU NSOC NCOR NTHR LOAD MEMTOT MEMUSE SWAPTO SWAPUS

----------------------------------------------------------------------------------------------

global - - - - - - - - - -

compute-0-0 lx-amd64 72 2 36 72 0.01 187.2G 2.2G 1024.0M 0.0

hl:arch=lx-amd64

hl:num\_proc=72.000000

hl:mem\_total=187.186G

hl:swap\_total=1023.996M

hl:virtual\_total=188.186G

hl:m\_topology=SCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTSCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTT

hl:m\_socket=2.000000

hl:m\_core=36.000000

hl:m\_thread=72.000000

hl:load\_avg=0.010000

hl:load\_short=0.000000

hl:load\_medium=0.010000

hl:load\_long=0.050000

hl:mem\_free=185.002G

hl:swap\_free=1023.996M

hl:virtual\_free=186.002G

hl:mem\_used=2.184G

hl:swap\_used=0.000

hl:virtual\_used=2.184G

hl:cpu=0.000000

hl:m\_topology\_inuse=SCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTSCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTT

hl:np\_load\_avg=0.000139

hl:np\_load\_short=0.000000

hl:np\_load\_medium=0.000139

hl:np\_load\_long=0.000694

hc:h\_vmem=187.200G

hc:slots=72.000000

hc:gpu=2.000000

Note that, not all resources are available on all nodes. For example, gpu resources are only available on node **compute-0-0** and **compute-0-1**. For each resource a single line is displayed with the following format:

<`h`ost resource | `g`lobal resource><`c`onsumable resource | `l`oad value>:<resource\_name>=<resource\_current\_value>

hl:mem\_free=185.002G means the host now have 185.002G free memory, and hc:gpu=2.000000 means there are 2 consumable gpu resources.

**Please refer to** [**https://arc.liv.ac.uk/SGE/**](https://arc.liv.ac.uk/SGE/) **for more information on SGE.**

**Managing your storage**

**$HOME** directory is the directory that you are first in after logging into the system. Its capacity is small, and it’s relatively slow. **Don’t put your data set in home directory**. It’s OK to put your scripts and source codes here as a backup. Your personal storage space is located in /share/data1/private/$username. This folder has a decent size quota and should be used for the data you want to keep on the cluster for an extended period of time. The majority of your work should be done in /share/data2/public/$username folder. This folder is a large shared common space for research users. To keep the size of this folder under control, there will be an automatic file cleaning every month during which all files that have not been accessed in the last month will be removed. A notice will be sent to all research users before the cleaning process. Backup of all disk spaces are automatically provided using RAID.

Note: Autofs is used to manage those shared disks. They are not mounted and ready by default. Instead, every time you access the folder, it will be automatically mounted. And if no one access the folder in last 20 minutes, it will be automatically unmounted.

**Installing third-party software**

You **CANNOT** use yum or any other installation process that requires elevated privileges, but this is almost never necessary. In most case you can download the source code and build the software in your own directory. Simple autotools build process looks like this:

$ export INSTALLDIR=$YOUR\_WORK\_DIR

$ ./configure --prefix=$INSTALLDIR

$ make

$ make install

Use the "--prefix" command line option to specify a non-default installation directory.

Other languages, frameworks, and build systems generally have equivalent mechanisms for installing software in user space. In most cases a web search like "Python Linux install local" will get you the information you need.

In Python, a local install will resemble one of the following examples:

$ pip install requests --user # install requests package to $ INSTALLDIR /.local

$ python setup.py install –user # install to $ INSTALLDIR /.local

$ pip install requests --prefix=$INSTALLDIR # custom location; add to PYTHONPATH

**Please refer to installation documents provided by the software for more information.**

**Pre-installed software list**

|  |  |  |
| --- | --- | --- |
| Name | Path | Notes |
| Gcc 4.8.5-16 | /usr/bin/gcc |  |
| GNU Fortran 4.8.5-16 | /usr/bin/gfortran |  |
| Python 2.7.5 | /usr/bin/python |  |
| Python 2.7.16 | /share/data2/apps/bin/python |  |
| Python 3.8.0 | /share/data2/apps/bin/python3 |  |
| Cuda 10.1 | /usr/local/cuda-10.1 | compute-0-0: v10.1.168  compute-0-1: v10.1.243 |
| nvcc 10.1 | /usr/local/cuda-10.1/bin/nvcc | compute-0-0: v10.1.168  compute-0-1: v10.1.243 |
| openmpi 1.10.6 | /usr/lib64/openmpi |  |
| BLAS 3.4.2 | /usr/lib64/libblas.so.3 |  |
| gsl 1.15 | /usr/lib64/libgsl.so.0 |  |
| cmake 2.8.12.2 | /usr/bin/cmake |  |
| Emacs 24.3.1 | /usr/bin/emacs |  |
| nasm 2.10.07 | /usr/bin/nasm |  |
| ffmpeg 4.2 | /usr/bin/ffmpeg |  |
| gimp 2.8.16 | /usr/bin/gimp |  |
| gnuplot 4.6.2 | /usr/bin/gnuplot-wx |  |
| ccrypt 1.10 | /usr/bin/ccrypt |  |
| jre 1.8.0\_231 | /usr/java/jre1.8.0\_231-amd64 |  |
| jdk 1.8.0\_101 | /usr/java/jdk1.8.0\_101 |  |
| scilab 6.0.2 | /share/data2/apps/scilab-6.0.2 |  |
| Vim | /usr/bin |  |
| Quantum Espresso | /share/data2/apps/bin |  |
| Paraview 5.7.0 | /share/data2/apps/ParaView |  |
| Paraview Server 5.7.0 | /share/data2/apps/ParaView-Server/ |  |
| Fiji | /share/data2/apps/Fiji.app |  |
| Maud 2.92 | /share/data2/apps/Maud-2.92 |  |
| Lammps 7Aug19 | /share/data2/apps/bin/lmp |  |
| fftw 3.3.8 | /share/data2/apps/include/fftw3.h  /share/data2/apps/include/fftw3-mpi.h |  |
| openfoam | /share/data2/apps/bin/openfoam7-linux |  |
| basilisk | /share/data2/apps/basilisk |  |
| sshsf | /usr/bin |  |
| xterm | /usr/bin |  |
|  |  |  |

Software was either installed into /usr (prebuild rpm) or /share/data2/apps (build from source code). Some prebuild software are just decompressed and put into the /share/data2/apps folder, e.g., /share/data2/apps/Maud-2.92. The /share/data2/apps/bin path is not included in the search path of the command line by default. You can add the path to PATH environment. For one-time use, type the command below

$ export PATH=/share/data2/apps/bin:$PATH

Otherwise, you need to type the full path of the executable binary to run, e.g., /share/data2/apps/bin/python3. To avoid typing above command each time, you can add it to your shell’s personal initialization file. For bash shell, you should add the command above to ~/.bashrc.

You can add more path to PATH environment. For example:

$ export PATH=/share/data2/apps/scilab-6.0.2/bin:/share/data2/apps/bin:$PATH