0. Welcome to Gadi

Gadi is Australia's most powerful supercomputer, a highly parallel cluster comprising more than 150,000 processor cores on ten different types of compute nodes. Gadi accommodates a wide range of tasks, from running climate models to genome sequencing, from designing molecules to astrophysical modelling. To start using Gadi, you should read this page which covers most of the basics you need to know before submitting your first job.

Making the most of the Gadi compute systems requires users to parallelise their tasks to achieve the best scalability. To run these computationally intensive tasks within one compute node across N processors or across N compute nodes, where N>=1, users need to wrap the tasks into a job and submit the job to a scheduler. In this job submission, users, request the amount of compute resources their job needs to be exclusively accessible by the tasks in the job for a certain specified period of time.

User applications run under a customised CentOS operating system on Gadi and by default users interact with the system through the command line interface. For users unfamiliar with Linux commands, please keep the below 'Linux Command Quick Reference' page handy when reading through this welcome page.

- Linux Command Quick Reference

HOW TO USE THIS PAGE

This page should take new users around an hour to read through.

- The first two sections explains how to log in to Gadi and some important environment variables at login.
- The third section 'Gadi Resources' summarises the various resources available on Gadi and briefly covers all the basics of each type of resource.
- The fourth section provides suggestions about moving data in and out of Gadi.
- The fifth sections explains how jobs work on Gadi and includes three job submission examples.
- The last section 'Job Monitoring' shows the commands that can be used to monitor jobs on Gadi with example return messages explained.

Don't worry if some of the sections don't mean anything to you, yet. If you think they are not relevant to your workflow, simply go on to the next section. If later you run into an error, you can always come back and read the skipped section then.

Logging In & Login Nodes

To use Gadi, you need to be registered as an NCI user first. Read our notes here on User Account Registration.

To register and get a username, please follow the instructions at the MyNCI portal.

To run jobs on Gadi, you need to first log in to the system. Users on Mac/Linux can use the built-in terminal. For Windows users, we recommend using MobaXterm as the local terminal. Logging in to Gadi happens through a Gadi login node.

For example, user aaa777 would run

```
ssh aaa777@gadi.nci.org.au
```

in the local terminal. Once the password is accepted, the ssh connection will be established on one of the ten Gadi login nodes, gadi-login-xx, where xx= [01,10]. Note that Gadi login nodes are separate from the Gadi compute nodes on which jobs actually run.

For example, the following connection is established on the login node gadi-login-02, as shown in the prompt in the last line.

```
$ ssh aaa777@gadi.nci.org.au
aaa777@gadi.nci.org.au's password:
################################################################################
#                  Welcome to the NCI National Facility!                       #
#      This service is for authorised clients only. It is a criminal           #
#      offence to:                                                             #
#                - Obtain access to data without permission                    #
#                - Damage, delete, alter or insert data without permission     #
#      Use of this system requires acceptance of the Conditions of Use         #
#      published at http://nci.org.au/conditions/                              #
################################################################################
|         gadi.nci.org.au - 155,232 processor InfiniBand x86_64 cluster       |
================================================================================

Jan 8 2020 Storage Flags Enforced
   Storage flags are now enforced for all PBS jobs. You must request the
   storage you wish to use in your PBS jobs. For example:

       -l storage=scratch/ab12+gdata/yz98

   Any storage not requested will not be visible in jobs.

================================================================================

[aaa777@gadi-login-02 ~]$
```

Message of the Day

When logging in, the first thing you see under the welcome text is the Mesage of the Day. You can see an example in the above block. Please consider it a noticeboard and read it on every login. News and status updates relevant to Gadi users will be posted to it.

Usage Limit on Login Nodes

Users are encouraged to submit and monitor jobs, prepare scripts for job submissions, compile small applications, and transfer a small amount of data on the login nodes. To ensure fair usage of login node resources amongst all users, any processes that cumulatively use more than 30 minutes CPU time and /or instantaneously use more than 4 GiB memory on any login node will be killed immediately.

Login Environment

At login, your landing point is your home directory whose path is set in the environment variable HOME in the login shell. This also sets the default project in PROJECT and your default shell in SHELL according to the file $HOME/.config/gadi-login.conf.

For example, the user aaa777 enters the home directory `/home/777/aaa777/` right after the ssh connection is established on the Gadi login node gadi-login-05. The default project is set to ‘a00’ and the shell is set to `/bin/bash` according to the default value in the file `/home/777/aaa777/.config/gadi-login.conf`.

```
[aaa777@gadi-login-05 ~]$ pwd
/home/777/aaa777/
[aaa777@gadi-login-05 ~]$ echo $HOME
/home/777/aaa777/
[aaa777@gadi-login-05 ~]$ cat $HOME/.config/gadi-login.conf
PROJECT a00
SHELL /bin/bash
[aaa777@gadi-login-05 ~]$ echo $PROJECT
a00
[aaa777@gadi-login-05 ~]$ echo $SHELL
/bin/bash
```

Gadi also supports the shells ksh, zsh and tcsh. Please edit the file $HOME/.config/gadi-login.conf to set the one that suits you as the default.

To add more default functions, aliases, modules that are available in your login shell and/or the PBS job shell, please edit the file $HOME/.bashrc in the corresponding `IF` clause(s) in `in_interactive_shell` for both interactive PBS jobs and login shells, `in_login_shell` for only login shell, and `in_pbs_job` for non-interactive PBS jobs.

Gadi Resources

Users can perform various tasks on Gadi, for example, to run computationally intensive jobs on compute nodes, build their own applications on login nodes /data-mover nodes and manage storage on different filesystems. A summary of the available resources of all kinds on Gadi is shown below.

Resource Name		Owner	Accessible from	Size Limit	Allocation Valid Until	Resource Specific Comments
Compute Hours		project	n.a.	amount set by scheme manager	end of quarter	
storage	**$HOME**	user	PBS jobs / login nodes	10 GiB with no possible extension	user account deactivation	• with backups in $HOME/.snapshot
	/scratch /$PROJECT	project	PBS jobs† / login nodes	72 GiB by default, more on jobs' demand	project retirement/job demand changes	• designed for jobs with large data IO • data expires in 90 days since creation [tbc when details available] • number-of-files limit applied
	/g/data /$PROJECT	project	PBS jobs† / login nodes	amount set by scheme manager	project retirement	• designed for hosting persistent data • number-of-files limit applied • also accessible from other NCI services, like cloud
	mdss	project	PBS copyq jobs† / login nodes	amount set by scheme manager	project retirement	• tape-based archival data sto • two copies created in two different buildings
	$PBS_JOBFS	user	PBS jobs *	disk space available on the job's hosting node(s)	job termination	• designed for jobs with frequent and small IO
software applications		NCI	PBS jobs / login nodes	n.a.	n.a.	• built from source on Gadi when possible • more can be added on request ‡
license		software group owner	PBS jobs / login nodes	available seats on the licensing server	license expiry date	• access controlled by software group membership †† • NCI owned licenses are for academic use only • projects, institutions and universities can bring in their own licenses

† need to be explicitly mounted using the PBS directive `-lstorage`, see details here[a link to PBS jobs, submission script explained]

‡ According to demand, dependencies and scalability, applications can be requested to be added to the Gadi /apps repository.

* job owner can access the folder through commands like `qls` and `qcp` on the login node during the job.

†† module file and PBS `-lsoftware` directive are used when controlling access to license [example link]

Compute Hours

To run jobs on Gadi, users need to have sufficient allocated compute hours available. Importantly, compute allocations are granted to **projects** instead of directly to users. Only members of a project can look up and use its compute allocation. To look up how much compute allocation is available in your project, run the command 'nci_account'. For example, to check the grant/usage of the project a00, user aaa777 would run

```
$ nci_account -P a00

Usage Report: Project=a00 Period=2020.q2
============================================================
    Grant:   785.00 KSU
     Used:   403.18 KSU
 Reserved:    23.04 KSU
    Avail:   358.78 KSU
```

This shows the total, used, reserved, and available compute grant of the project in the current billing period at the time of the query. **SU** is short for Service Unit, the unit that measures Gadi compute hours. Jobs run on the Gadi normal queue are charged 2 SUs to run for an hour on a single core with a memory allocation of up to 4 GiB. Jobs on Gadi are charged for the proportion of a node's total memory that is used: see more examples of job cost here. In addition, different Gadi queues have different charge rates: see the breakdown of charge rates here.

The Grant amount listed is always equal to the sum of Used, Reserved and Avail. Every time a project has a job submitted, its potential cost, calculated according to the requested walltime, is reserved from the total Grant and reduces the amount of Avail. The actual cost based on walltime usage is determined only after the job's completion. If it finishes within the requested walltime limit, the over-reserved allocation is returned to the Avail amount and the actual cost goes to Used amount. When the project has no jobs that are waiting to run or running, Reserved SU returns to zero.

If there are not enough SUs available for a job to run according to its requested resource amounts, the job will be waiting in the queue indefinitely. The project lead CI should contact their allocation scheme manager to apply for more.

The Home Folder $HOME

Each user has a project-independent home directory. The storage limit of the home folder is fixed at 10 GiB. We recommend to use it as the folder where you host any scripts you want to keep to yourself. Users are encouraged to share their data elsewhere, see our Data Sharing Suggestions. All data on /home is backed up. In the case of 'accidental' deletion of any data inside the home folder, you can retrieve the data by following the example here.

Project Folder on Lustre Filesystems /scratch and /g/data

Users get access to storage space on the /scratch filesystem, and on the /g/data filesystems if the project folder exists, through project memberships. For example, the user jjj777 is a member of the projects a00 and b11, therefore, jjj777 has permission to read and create files/folders in the folders /scratch/a00/jjj777 and /scratch/b11/jjj777.

```
[jjj777@gadi-login-05 ~]$ mkdir -p /scratch/a00/jjj777/test
[jjj777@gadi-login-05 ~]$ ls -ltrah /scratch/a00/jjj777
total 48K
drwxr-s---  2 jjj777 a00 16K Nov 14  2019 tmp
drwxrws--- 71 root   a00 16K Nov 14  2019 ..
drwxr-sr-x  2 jjj777 a00 16K Sep  1 15:44 test
drwxr-s---  4 jjj777 a00 16K Sep  1 15:44 .
```

The first column in the output shows the permissions set for the folder/file. For more information on unix file permissions, see this page.

To look up how much storage you have access to through which projects, run the command 'lquota' on the login node. It prints out the storage allocation info together with its live usage data. For example, the return message

```
------------------------------------------------------------------------
          fs      Usage     Quota     Limit   iUsage   iQuota   iLimit
------------------------------------------------------------------------
   a00 scratch    3.17GB     1.0TB     2.0TB    21622   202000   404000
  xy11 scratch  438.76GB     1.0TB     2.0TB   204000   202000   404000 Over inode limit
   a00   gdata  345.89GB     2.0TB     4.0TB     3957   312000   624000
------------------------------------------------------------------------
```

shows the user has access to two project folders on /scratch and one on /g/data. The project a00 has 1.0 TiB storage quota on /scratch and used 3.17 GiB of it.

There is also a quota called `iQuota` applied to the storage allocation on /scratch and /g/data. It sets the maximum number of files allowed in the project. For example, it shows in the above return message the project xy11 has 438.76 GiB data in 204k files which is exceeding the file-number limit iQuota set at 202000. Please keep the number of files under control as this can significantly affect the I/O performance. Lustre filesystems are good at handling large scale parallel I/O but the performance degrades significantly when doing frequent small IO operations.

To look up which user owns how much data and in how many files, please use the command `nci-files-report`. It pulls out the lastest snapshot of the storage usage from the data collected throughout the filesystem every 24 hours.

For example, to take a look at the snapshot of the usage by the project xy11 on the filesystem /scratch, run

```
nci-files-report -f scratch -g xy11
```

In its return message, it shows in which project folder which user owns how much data and in how many files, as shown in the column `count`. If there is a difference between the numbers in the column `space used` and `file size`, it generally suggests the user has a very big folder containing a large number of files.

```
------------------------------------------------------------------------------
         project             user     space used      file size          count
------------------------------------------------------------------------------
            xy11           jjj777         14.3GB         13.2GB         112086
             ...
------------------------------------------------------------------------------
```

Job Folder $PBS_JOBFS

All Gadi jobs by default have 100MB of storage space allocated on the hosting compute node(s). The path to the storage space is set in the environment variable PBS_JOBFS in the job shell. We encourage users to use the folder $PBS_JOBFS in their jobs that generate large numbers of small IO operations. It not only boosts the job's performance by saving the time spent in those frequent and small IO operations but also saves a lot of inode usage for your project on the shared filesystems like /scratch and /g/data. Note that the folder $PBS_JOBFS is physically deleted upon job completion/failure, therefore, it is crucial for users to copy the data in $PBS_JOBFS back to the local directory on the shared filesystem while the job is still running.

To request storage space on the local disk on the compute node(s) explicitly, please use the PBS directive `-ljobfs`. For example, to request 100 GiB local disk space on the compute node for your job, please add the following line to its submission script. See the Job Submission section below for all the details about submitting jobs to Gadi.

```
#PBS -l jobfs=100GB
```

If the job runs on multiple compute nodes, this request of 100 GiB of space will be equally distributed among all nodes. If the job requests more than the available disk space on the compute node(s), the submission would fail. Please browse the Gadi Queue Structure and Gadi Queue Limit pages to look up how much local disk is available on each type of compute node.

Tape Filesystem massdata

NCI operates a tape filesystem called massdata to provide an archive service for projects that need to back up their data. Massdata has two large tape libraries in separate machine rooms in two separate buildings. Projects have their data in their own path massdata/<*project*> on massdata, but the path is not directly accessible from Gadi: data requests must be launched from Gadi login nodes or from within copyq jobs. Here is a video clip showing the tape robot at work.

NCI provides the `mdss` utility for users to manage the migration and retrieval of files between multiple levels of a storage hierarchy: from on-line disk cache to offline tape archival. It connects to massdata and launches the corresponding requests. For example, `mdss get` first launches the requests to stage the remote files from the massdata repository into the disk cache, once the data gets online it then transfers the data back to your local directory, for example, a project folder on /scratch or /g/data.

To look up the status of the data within massdata owned by your project, run `mdss dmls` on a Gadi login node. For example,

```
$ mdss -P a00 dmls -ltrh aaa777
total 64K
-rw-r-----   1 aaa777    a00         77G 2016-12-01 10:10 (OFF) test.tar.gz
-rw-r--r--   1 aaa777    a00        100G 2020-09-08 14:48 (DUL) test_archive.tar
drwxr-sr-x   3 aaa777    a00          23 2020-09-08 14:49 (REG) test_dir
```

This example shows that in the directory massdata/a00/aaa777 the file test.tar.gz is offline in the tape archive (OFF), the file test_archive.tar is both in disk cache and on tape (DUL), and the directory test_dir is in cache (REG). If the project a00 doesn't have any storage allocation on massdata, it will say `No mass data store hosts available for project a00` in the return message. To look up the manual of more `mdss` commnands, run `man mdss` on the Gadi login node.

We encourage users to archive only infrequently used data to massdata as it generally takes a longer time to recall it than data transfers between lustre filesystems. Depending on how many requests are launched before yours on the tape filesystem, it may take more than a day to get your request(s) executed on massdata. To save times on data migration and retrieval, please keep the average file size around 100 GiB.

If you need access to massdata or more space on it, the project Lead CI should contact their allocation scheme manager to apply for it.

Software Applications and Licenses

Gadi has many software applications installed in the directory /apps and uses `Environment Modules` to manage them. To run any of them, load the corresponding module first. If the application requires licenses, join the corresponding software group through my.nci.org.au like you would join other projects.

If the applications or packages you need are not centrally installed on Gadi, please contact help@nci.org.au to discuss whether it is suitable to install the missing ones centrally on Gadi. We do not install all requested software applications but we are happy to assist with the installation inside of your own project folder.

Applications on /apps

The command `module avail` prints out the complete list of software applications centrally available on Gadi. To look for a specific application, please run `module avail <app_name>`. For example, `module avail open` prints out all the available versions for applications whose name starts with `open`.

```
$ module avail open
------------------------------------------------------------ /apps/Modules/modulefiles
------------------------------------------------------------
openbabel/2.4.1      openmpi/2.1.6-mt        openmpi/3.0.4-debug  openmpi/4.0.1           openmpi/4.0.2-debug
openquake/3.9.0
openmpi/2.1.6        openmpi/2.1.6-mt-debug  openmpi/3.1.4        openmpi/4.0.1-debug     openquake/3.7.1
openmpi/2.1.6-debug  openmpi/3.0.4           openmpi/3.1.4-debug  openmpi/4.0.2(default)  openquake/3.8.0
```

Before running the application, load the module into your current shell environment by running `module load <app_name>/<version>`. It sets the environment variables such that the shell knows where to find the relevant executables, libraries and header files and how to run the application. For example,

```
$ module load openmpi/3.1.4
$ which mpirun
/apps/openmpi/3.1.4/bin/mpirun
$ echo $C_INCLUDE_PATH
/apps/openmpi/3.1.4/include
```

To remove a module from your environment, run ‘module unload <app_name>‘. For example,

```
$ module list
Currently Loaded Modulefiles:
 1) openmpi/3.1.4
$ module unload openmpi
$ module list
No Modulefiles Currently Loaded.
```

where ‘module list’ prints out currently loaded modules. Because two versions of the same application cannot be loaded at the same time, it is not necessary to specify which version to remove.

To read more about `module` commands, please read the page Environment Modules.

Access to Software Licenses

Software group membership enables access to a particular licensed application. To join a software group, login to my.nci.org.au, navigate to ‘Projects and Groups‘, search for the software name.
Once you've identified the corresponding software group, read all the content under the ‘Join’ tab carefully before sending out the membership request. It may take a while for the software group lead CI to approve the membership request, send an email to help@nci.org.au or submit a ticket at help.nci.org.au if you need the approval immediately. Once the membership is approved, the access to the application will be enabled.

For example, after joining the software group matlab_unsw, in the return message of ‘module avail matlab’ it prints out the licence module ’matlab_licence /unsw’ together with the application module ‘matlab/R2019b’. You need to load both of them to run matlab jobs successfully on Gadi.

```
$ module avail matlab
------------------------------------------- /apps/Modules/restricted-modulefiles/matlab_unsw
-------------------------------------------
matlab_licence/unsw

----------------------------------------------------------- /apps/Modules/modulefiles
-----------------------------------------------------------
matlab/R2019b
```

If there is no licence module in the return message, it means it doesn't require you to load the license module to run the application. The access control is imposed on its installation directory permission.

To reserve enough license seats for your PBS jobs, please add the PBS directive `-lsoftware=<license_name>` in your submission script. To look up the right <license_name> for the license owned by the software group you joined, have a look at the live license usage page. This is where we publish the details about how many of which licenses are used in, and reserved for, which job. Please use it when you need to know the status of licenses your jobs are requesting.

File Transfer to/from Gadi

Gadi has six designated data-mover nodes with the domain name `gadi-dm.nci.org.au`` Please use it when transferring files to and from Gadi.

For example, aaa777 runs the following command line in the local terminal

```
scp input.dat aaa777@gadi-dm.nci.org.au:/home/777/aaa777
```

to transfer the file input.dat in the current directory to the home folder on Gadi.

If the transfer is going to take a long time, there is a possibility that it could be interrupted by network instability. For that reason, it is better to start the transfer in a resumable way. For example, the following command line allows user aaa777 to download data in the folder /scratch/a00/aaa777/test_dir on Gadi onto the current directory on their local machine using ‘rsync’.

```
rsync -avPS aaa777@gadi-dm.nci.org.au:/scratch/a00/aaa777/test_dir ./
```

If the download is interrupted, run the same command again to resume the download from where it left off.

Gadi Jobs

To run compute tasks such as simulations, weather models, and sequence assemblies on Gadi, users need to submit them as ‘jobs’ to ‘queues’. Job submission enables users to specify the queue, duration and resources needs of their jobs. Gadi uses PBSPro to schedule all submitted jobs and keeps nodes that have different hardware in different queues. See details about the hardware available in the different queues on the Gadi Queue Structure page. Users submit jobs to a specific queue to run jobs on the corresponding type of node.

When wrapping your tasks up inside a job for submission, an estimate of the amount of computational resources the job will use and for what duration is required, so that the job scheduler knows what resources to reserve for the job. If the job uses more than it requested, it will be terminated immediately. See the Gadi Queue Limit page for more details on resource limits and costs on each type of the nodes.

Once the requested amount of resources becomes available to the system, the job is sent to the scheduled hosting node(s). If the job requests more than a single node, it is first started on the head node where all the commands in the job submission script are executed one by one. Whether the job can utilise the requested amount of resources or not depends on the commands and scripts/binaries called in the submission script.

If some tasks in the job need access to the internet at any stage, they have to be separately packed into a job on a copyq node as none of the standard compute nodes have external network access outside of Gadi.

On job completion, by default, the contents of the job's standard output/error stream gets copied to a file in the working directory with the name in the format <jobname>.o<jobid>/<jobname>.e<jobid>. For example, when the job 12345678 finishes, there are two files created with the names ***job.sh.o12345678*** and ***job.sh.e12345678*** as the record of its STDOUT and STDERR, respectively, and these two log files are located inside the same folder where the job was submitted.

We recommend users check these two log files before proceeding with the post-processing of any output/result from the corresponding job.

Job Submission

To submit a job defined in a submission script, called for example 'job.sh', run

```
qsub job.sh
```

on the login node. Once the submission is accepted, the above command returns the jobID, for example, 12345678.gadi-pbs, which can be used for tracking its status.

> Because Gadi jobIDs always end with the string '.gadi-pbs' they are often referred by using the numbered part only.

The submission script consists of three sections. The first line specifies which shell to use. The second section includes all the PBS directives that define the resources the job needs and the last section contains all the command lines that you would use in an interactive shell to run the compute task.

Submission Script Example

Here is an example job submission script to run the python script 'main.py' which is assumed to be located inside the same folder where you run 'qsub job.sh'.

```
#!/bin/bash

#PBS -l ncpus=48
#PBS -l mem=190GB
#PBS -l jobfs=200GB
#PBS -q normal
#PBS -P a00
#PBS -l walltime=02:00:00
#PBS -l storage=gdata/a00+scratch/a00
#PBS -l wd

module load python3/3.7.4
python3 main.py $PBS_NCPUS > /g/data/a00/$USER/job_logs/$PBS_JOBID.log
```

The first section '#!/bin/bash' specifies that the job should use the 'bash' shell.

The second section with all the lines that start with '#PBS' specifies how much of each resource the job will need. It requests an allocations of 48 CPU cores, 190 GiB memory, and 200 GiB local disk on a compute node from the normal queue for its exclusive access for 2 hours. It also requests the system to mount both the a00 project folders on the filesystems /scratch and /g/data inside the job, and to enter the working directory once the job is started. Please see more PBS directives explained in here. Note that a '-lstorage' directive must be included if you need access to /g/data, otherwise these folders will not be accessible when the job is running.

To find the right queue for your jobs, please browse the Gadi Queue Structure and Gadi Queue Limit pages.

Users are encouraged to request resources to allow the task(s) to run around the ‘sweet spot’ where the job benefits from parallelism and achieves shorter execution time while utilising at least 80% of the requested compute capacity.

While searching for the sweet spot, please be aware that it is common to see components in a task that run only on a single core and cannot be parallelised. These sequential parts drastically limit the parallel performance. For example, having 1% sequential parts in a certain workload limits the overall CPU utilisation rate of the job when running in parallel on 26 cores to less than 80%. Moreover, parallelism adds overhead which in general scales up with the increasing core count and, when beyond the ‘sweet spot’, results in a waste of time on unnecessary task coordinations.

In the third section of the submission script, it loads the module python3/3.7.4 into the job shell and calls python3 to execute the script main.py with the argument $PBS_NCPUS and redirects the output from the python script to a log file into the /g/data project folder with the name of the jobID.

Users are encouraged to run their tasks if possible in bigger jobs to take advantage of the potential massive parallelism that can be achieved on Gadi. However, depending on the application, it may not be possible for the job to run on more than a single core/node. For applications that do run on multiple cores/nodes, the commands and scripts/binaries called in the third section determine whether the particular job can utilise the requested amount of resources or not. Users need to edit the script/input files which define the compute task to allow it, for example, to run on multiple cores/nodes. It may take several iterations to find the ideal details for sections two and three of the submission script when exploring around the job's sweet spot.

Interactive Jobs

We recommend users try their workflow on Gadi in an *interactive job* before submitting the tasks using the prepared submission script. This is because, when the interactive job starts, the user runs commands in an interactive shell on the head compute node, which allows one to quickly test out possible solutions. For example, this can be used to debug large jobs that run many parallel processes on many nodes or install applications that have to be built when GPUs are available.

To submit an interactive job, run ‘qsub -I’ on the login node. For example, to start an interactive job on Gadi’s gpuvolta queue through project a00 with the request of 48 CPU cores, 4 GPUs, 380 GiB memory, and 200 GiB local disk for 5 minutes on one gpu compute node, run

```
qsub -I -qgpuvolta  -Pa00 -lwalltime=00:05:00,ncpus=48,ngpus=4,mem=380GB,jobfs=200GB,storage=gdata/a00,wd
```

Once the job starts, it mounts the folder /g/data/a00 to the job and enters the job's working directory in which the job was submitted.

Example

Here is a minimum example of an interactive job. You can tell the interactive job has been started by the prompt changing from something like [aaa777@gadi-login-03 ~] to [aaa777@gadi-gpu-v100-0079 ~]. The job shell doesn't have any modules loaded by default. If you need to repeatedly load modules inside interactive jobs, please edit the file ~/.bashrc to load them automatically. Once you're done with the interactive job, run ‘exit’ to terminate the job.

```
[aaa777@gadi-login-03 ~]$ qsub -I -lwalltime=00:05:00,ncpus=48,ngpus=4,mem=380GB,jobfs=200GB,wd -qgpuvolta
qsub: waiting for job 11029947.gadi-pbs to start
qsub: job 11029947.gadi-pbs ready

[aaa777@gadi-gpu-v100-0079 ~]$ module list
No Modulefiles Currently Loaded.
[aaa777@gadi-gpu-v100-0079 ~]$ exit
logout

qsub: job 11029947.gadi-pbs completed
```

Copyq Jobs

For transfer of bulk data (say more than 500 GiB), it is recommended to do it in a job submitted to the queue ‘copyq’ because long data transfer processes running on the login node will be terminated when reaching the 30-minute cumulative CPU time usage limit. Long software installations that require an internet connection are also recommended to be run inside copyq jobs.

To submit a copyq job, called for example job.sh, define the tasks in the submission script, specify the queue to be copyq, and run

```
qsub job.sh
```

on a login node.

Example

An example copyq job submission script job.sh could be

```
#!/bin/bash

#PBS -l ncpus=1
#PBS -l mem=2GB
#PBS -l jobfs=2GB
#PBS -q copyq
#PBS -P a00
#PBS -l walltime=02:00:00
#PBS -l storage=gdata/a00+massdata/a00
#PBS -l wd

tar -cvf my_archive.tar /g/data/a00/aaa777/work1
mdss -P a00 mkdir -p aaa777/test/
mdss -P a00 put my_archive.tar aaa777/test/work1.tar
mdss -P a00 dmls -ltrh aaa777/test
```

This script defines a job that creates a tar archive of the data inside the folder /g/data/a00/aaa777/work1 and puts the tar file onto the tape filesystem massdata inside the folder aaa777/test owned by the project a00.

To compile code inside a copyq job, it may be necessary to load modules such as intel-compiler and request more jobfs to allow enough disk space to host data written to $TMPDIR.

Job Monitoring

Once a job submission is accepted, its jobID is shown in the return message and can be used to monitor the job's status. Users are encouraged to keep monitoring their own jobs at every stage of their lifespan on Gadi. Note however, that excessive polling of the PBS servers for monitoring purposes will be considered attacks. A frequency of one monitoring query every 10 minutes is more than enough.

The PBS server which manages job submissions and scheduling answers all submissions, queries, and requests related to jobs. To ensure its quick response to essential requests, don't launch frequent job monitoring queries.

Queue Status

To look up the status of a job in the queue, run the command 'qstat'. For example, to lookup the job 12345678 in the queue, run

```
qstat -swx 12345678
```

If the job is running, you would see something like

```
gadi-pbs:
                                                                                                  Req'd
Req'd   Elap
Job ID                          Username        Queue           Jobname         SessID   NDS  TSK   Memory Time
S Time
------------------------------ --------------- --------------- --------------- -------- ---- ----- ------ -----
- -----
12345678.gadi-pbs               aaa777          normal-exec     job.sh          2545114    1    48  190gb  02:00
R 00:35:21
   Job run at Fri Sep 04 at 10:38 on (gadi-cpu-clx-2697:ncpus=48:mem=199229440kb:jobfs=209715200kb)
```

It shows that the job was submitted by the user aaa777 to the normal queue, requested 48 cores and 190 GiB memory for 2 hours and has been running (S = "R") for 35 minutes and 21 seconds. The line at the bottom says the job started at 10:38 am on 4 Sept on the compute node gadi-cpu-clx-2697 where it has exclusive access to 48 cores, 190 GiB memory and 200 GiB jobfs local disk.

CPU and Memory Utilisation

We encourage users to keep monitoring their own jobs' utilisation rate at every stage because, if they run into errors and fail to exit, this shows up clearly in the drop in utilisation rate.

While a low utilisation rate is sufficient for spotting the underuse of compute time, a 100% utilisation rate doesn't necessarily confirm the efficient use of requested resources. Further profiling is required to tell whether performance improvements are possible.

To see how much CPU and memory the job actually has been using, run the command ‘nqstat_anu’. For example, to look up the info of the job 12345678, run

```
nqstat_anu 12345678
```

The output shows the CPU utilisation rate in the column ‘%CPU’ and the peak memory usage in the column ‘RSS’ and ‘mem’. The example output below says the job used only 23% of the compute capacity of the requested 48 cores in the elapsed 36 minutes and 47 seconds. Depending on the tasks running inside the job, this percentage may increase while the job proceeds further. We normally recommend users aim for at least 80% overall CPU utilisation rate.

```
                                %CPU  WallTime  Time Lim     RSS    mem  memlim  cpus
 12345678 R aaa777  a00 job.sh  23    00:36:47   2:00:00  5093MB 5093MB 190.0GB    48
```

Process Status and Files in Folder $PBS_JOBFS

To monitor the status of processes inside a job, run the command ‘qps’. For example, to take a snapshot of the process status of the job 12345678, run

```
$ qps 12345678
```

To list the files inside the folder $PBS_JOBFS on the compute node from the login node, run the command ‘qls’. For example,

```
$ qls 12345678
Node 0: gadi-cpu-clx-0027:/jobfs/12345678.gadi-pbs:
testjob_outdir

$ qls -l 12345678/testjob_outdir
Node 0: gadi-cpu-clx-0027:/jobfs/12345678.gadi-pbs:
total 4
drwxr-xr-x 2 aaa777 a00   6 Sep  1 10:54 tmp
drwxr-xr-x 3 aaa777 a00  34 Sep  1 10:54 short_read
-rw-r--r-- 1 aaa777 a00 649 Sep  1 10:54 job.timing
```

To copy the file in the folder $PBS_JOBFS into your current folder, run the command ‘qcp’ on the login node. For example,

```
$ qcp -n 0 12345678/testjob_outdir/job.timing ./job.timing.bk1
```

Congratulations! Hopefully you are now ready to get started on Gadi. Please keep hold of the cheatsheet below which provides most of the Gadi specifics you may need to look up again in the future.

- Gadi Quick Reference Guide