Speed: The GCS ENCS Cluster

Serguei A. Mokhov Gillian A. Roper
Network, Security and HPC Group*
Gina Cody School of Engineering and Computer Science
Concordia University
Montreal, Quebec, Canada
rt-ex-hpc~AT~encs.concordia.ca
Version 6.6-dev-03

Abstract

This document primarily presents a quick start guide to the usage of the Gina Cody School of Engineering and Computer Science compute server farm called "Speed" – the GCS ENCS Speed cluster, managed by HPC/NAG of GCS ENCS, Concordia University, Montreal, Canada.

Contents

*The group acknowledges the initial manual version VI produced by Dr. Scott Bunnell while with us.

1 Introduction

This document contains basic information required to use "Speed" as well as tips and tricks, examples, and references to projects and papers that have used Speed. User contributions of sample jobs and/ or references are welcome. Details are sent to the `hpc-ml` mailing list.

Resources:

- Our public GitHub page where the manual and sample job scripts are maintained (pull-requests (PRs), subject to review, are welcome):
  `https://github.com/NAG-DevOps/speed-hpc`
  `https://github.com/NAG-DevOps/speed-hpc/pulls`

- Our official Concordia page for the "Speed" cluster:
  `https://www.concordia.ca/ginacody/aits/speed.html`
  which includes access request instructions.

- All registered users are subscribed to the `hpc-ml` mailing list upon gaining access.

- Introductory slides of Speed presented to departments [10].

1.1 What Speed Comprises

- Twenty four (24) 32-core compute nodes, each with 512 GB of memory and approximately 1 TB of volatile-scratch disk space.

- Twelve (12) NVIDIA Tesla P6 GPUs, with 16 GB of memory (compatible with the CUDA, OpenGL, OpenCL, and Vulkan APIs).

- One AMD FirePro S7150 GPUs, with 8 GB of memory (compatible with the Direct X, OpenGL, OpenCL, and Vulkan APIs).

1.2 What Speed Is Ideal For

- Jobs that are too demanding for a desktop, but that are not worth the hassles associated with the provincial and national clusters.

- Single-core batch jobs; multithreaded jobs up to 32 cores (i.e., a single machine).

- Anything that can fit into a 500-GB memory space and a scratch space of approximately 1 TB.

- CPU-based jobs.

- CUDA GPU jobs (`speed-05`, `speed-17`).

- Non-CUDA GPU jobs using OpenCL (`speed-19` and `speed-05|17`).

1.3 Speed's Intent

- To Design and Develop, test and run parallel, batch, etc. algorithms, scripts with partial data sets.
- Prepare them for big clusters:
  - Calcul Quebec
  - Compute Canada
  - Cloud platforms

1.4 Available Software

We have a great number of open-source software available and installed on Speed – various Python, CUDA versions, C++/Java compilers, OpenGL, OpenFOAM, OpenCV, TensorFlow, OpenMPI, OpenISS, MARF [15], etc. There are also a number of commercial packages, subject to licensing contributions, available, such as MATLAB [5, 14], Abaqus [1], Ansys, Fluent [2], etc.

To see the packages available, run `ls -al /encs/pkg/` on `speed.encs`.

In particular, there are over 2200 programs available in `/encs/bin` and `/encs/pkg` under Scientific Linux 7 (EL7).

- Popular concrete examples:
  - MATLAB (R2016b, R2018a, R2018b)
  - Fluent (19.2)
  - Singularity (Docker-like container), can run other OS's apps, like Ubuntu's.
- We do our best to accommodate custom software requests. Python environments can be used to have user-custom installs in the scratch directory.
- A number of specific environments are available, too.
- Popular examples mentioned (loaded with, `module`):
  - Python (2.3.0 - 3.5.1)
  - Gurobi (7.0.1, 7.5.0, 8.0.0, 8.1.0)
  - Ansys (16, 17, 18, 19)
  - OpenFOAM (2.3.1, 3.0.1, 5.0, 6.0)
  - Cplex 12.6.x to 12.8.x
  - OpenMPI 1.6.x, 1.8.x, 3.1.3

1.5 Requesting Access

The first step to using the "Speed" cluster is to request access.

Please email your access request to, `rt-ex-hpc AT encs.concordia.ca`, including your ENCS account's username.

If you are a student, please include the name of your Supervisor or instructor as well as a statement from that person which indicates that you may have access to "Speed".

2 Job Management

In these instructions, anything bracketed like so, <>, indicates a label/value to be replaced (the entire bracketed term needs replacement).

2.1 Getting Started

Once your ENCS account has been granted access to "Speed", use your ENCS account credentials to create an SSH connection to `speed` (an alias for `speed-submit.encs.concordia.ca`).

2.1.1 SSH Connections

If you are connecting from home, and have a Mac or Linux system, in a terminal, this will connect you (on a single line):

```
ssh -o ProxyCommand="ssh <ENCSusername>@login.encs.concordia.ca nc speed 22"
  <ENCSusername>@speed.encs.concordia.ca
```

Windows users can create SSH connections via PuTTY (or MobaXterm).

All users are expected to have a basic understanding of Linux and its commonly used commands.

2.1.2 Environment Set Up

After creating an SSH session to "Speed", you will need to source the scheduler file:

```
source /local/pkg/uge-8.6.3/root/default/common/settings.csh
```

If sourcing the settings file has been successful, you will now have access to the scheduler commands. For example, if, `qstat -f -u "*"`, returns something non-error related, you are in business.

The next step is to copy a job template to your home directory and to set up your cluster-specific storage. Execute the following command from within your home directory. (To move to your home directory, type `cd` at the Linux prompt and press `Enter`.)

```
cp /home/n/nul-uge/template.sh . && mkdir /speed-scratch/$USER
```

Tip: Add the source command to your shell-startup environment.

For **new ENCS Users**, and/ or those who don't have an `.tcshrc` file, move a copy of the `.tcshrc` in `nul-uge`'s. home directory to your home directory with:

```
cp /home/n/nul-uge/.tcshrc .
```

Users who already have an `.tcshrc` file, use a text editor, such as `vim` or `emacs`, to add the source request to your existing shell-startup environment (i.e., to the `.tcshrc` file in your home directory).

Note that you will need to either log out and back in, or execute a new shell, for the environment changes in the updated `.tcshrc` file to be applied (**important**).

If you use `bash`, or another shell, please contact `rt-ex-hpc AT encs.concordia.ca`.

2.2 Job Submission Basics

Preparing your job for submission is fairly straightforward. Editing a copy of the `template.sh` you moved into your home directory during Section 2.1.2 is a good place to start. You can also use a job script example from our GitHub's (`https://github.com/NAG-DevOps/speed-hpc`) "src" directory and base your job on it.

Job scripts are broken into four main sections:

- Directives
- Module Loads
- User Scripting

2.2.1 Directives

Directives are comments included at the beginning of a job script that set the shell and the options for the job scheduler.

The shebang directive is always the first line of a script. In your job script, this directive sets which shell your script's commands will run in. On "Speed", we recommend that your script use a shell from the `/encs/bin` directory.

To use the `tcsh` shell, start your script with: `#!/encs/bin/tcsh`

For `bash`, start with: `#!/encs/bin/bash`

Directives that start with `"$#"`, set the options for the cluster's "Altair Grid Engine (AGE)" scheduler. The script template, `template.sh`, provides the essentials:

```
#$ -N <jobname>
#$ -cwd
#$ -m bea
#$ -pe smp <corecount>
#$ -l h_vmem=<memory>G
```

Replace, `<jobname>`, with the name that you want your cluster job to have; `-cwd`, makes the current working directory the "job working directory", and your standard output file will appear here; `-m bea`, provides e-mail notifications (begin/end/abort); replace, `<corecount>`, with the degree of (multithreaded) parallelism (i.e., cores) you attach to your job (up to 32), be sure to delete or comment out the `#$ -pe smp` parameter if it is not relevant; replace, `<memory>`, with the value (in GB), that you want your job's memory space to be (up to 500), and all jobs MUST have a memory-space assignment.

If you are unsure about memory footprints, err on assigning a generous memory space to your job so that it does not get prematurely terminated (the value given to `h_vmem` is a hard memory ceiling). You can refine `h_vmem` values for future jobs by monitoring the size of a job's active memory space on `speed-submit` with:

```
qstat -j <jobID> | grep maxvmem
```

Memory-footprint values are also provided for completed jobs in the final e-mail notification (as, "Max vmem").

Jobs that request a low-memory footprint are more likely to load on a busy cluster.

2.2.2 Module Loads

As your job will run on a compute or GPU "Speed" node, and not the submit node, any software that is needed must be loaded by the job script. Software is loaded within the script just as it would be from the command line.

To see a list of which modules are available, execute the following from the command line on `speed-submit`.

```
module avail
```

To list for a particular program (`matlab`, for example):

```
module -t avail matlab
```

Which, of course, can be shortened to match all that start with a particular letter:

```
module -t avail m
```

Insert the following in your script to load the `matlab/R2020a`) module:

```
module load matlab/R2020a/default
```

Use, `unload`, in place of, `load`, to remove a module from active use.
To list loaded modules:

```
module list
```

To purge all software in your working environment:

```
module purge
```

Typically, only the `module load` command will be used in your script.

2.2.3 User Scripting

The last part the job script is the scripting that will be executed by the job. This part of the job script includes all commands required to set up and execute the task your script has been written to do. Any Linux command can be used at this step. This section can be a simple call to an executable or a complex loop which iterates through a series of commands.

Every software program has a unique execution framework. It is the responsibility of the script's author (e.g., you) to know what is required for the software used in your script by reviewing the software's documentation. Regardless of which software your script calls, your script should be written so that the software knows the location of the input and output files as well as the degree of parallelism. Note that the cluster-specific environment variable, `NSLOTS`, resolves to the value provided to the scheduler in the `-pe smp` option.

Jobs which touch data-input and data-output files more than once, should make use of `TMPDIR`, a scheduler-provided working space almost 1 TB in size. `TMPDIR` is created when a job starts, and exists on the local disk of the compute node executing your job. Using `TMPDIR` results in faster I/O operations than those to and from shared storage (which is provided over NFS).

An sample job script using `TMPDIR` is available at `/home/n/nul-uge/templateTMPDIR.sh`: the job is instructed to change to `$TMPDIR`, to make the new directory `input`, to copy data

from `$SGE_O_WORKDIR/references/` to `input/` (`$SGE_O_WORKDIR` represents the current working directory), to make the new directory `results`, to execute the program (which takes input from `$TMPDIR/input/` and writes output to `$TMPDIR/results/`), and finally to copy the total end results to an existing directory, `processed`, that is located in the current working directory. TMPDIR only exists for the duration of the job, though, so it is very important to copy relevant results from it at job's end.

2.3 Sample Job Script

Now, let's look at a basic job script, `tcsh.sh` in Figure 1 (you can copy it from our GitHub page or from `/home/n/nul-uge`).

```
#!/encs/bin/tcsh

#$ -N qsub-test
#$ -cwd
#$ -l h_vmem=1G

sleep 30
module load gurobi/8.1.0
module list
```

Figure 1: Source code for `tcsh.sh`

This script sleeps on a node for 30 seconds, uses the `module` command to load the `gurobi/8.1.0` environment, and then prints the list of loaded modules into a file. Concentrating on the first four lines, the first line is the shell declaration; the next three lines are submission options passed to the scheduler. The first, `-N`, attaches a name to the job (otherwise it is called what the job script is called), the second, `-cwd`, tells the scheduler to execute the job from the current working directory, and not to use the default of your home directory (potentially important for output-file placement), and the third, `-l h_vmem`, requests and assigns a 1GB of memory space to the job (this is an upper bound, and jobs that attempt to use more will be terminated). Note that this third option is *not* optional (if you do not specify a memory space, submission of the job will fail). Also notice the syntax that denotes a scheduler option, the, `#$`.

The scheduler command, `qsub`, is used to submit (non-interactive) jobs. To submit this job: `qsub ./tcsh.sh`. You will see, `"Your job X ("qsub-test") has been submitted"`. The command, `qstat`, can be used to look at the status of the cluster: `qstat -f -u "*"`. You will see something like this:

```
queuename                      qtype resv/used/tot. load_avg arch          states
---------------------------------------------------------------------------------
a.q@speed-01.encs.concordia.ca BIP   0/0/32         0.01     lx-amd64
---------------------------------------------------------------------------------
a.q@speed-03.encs.concordia.ca BIP   0/0/32         0.01     lx-amd64
---------------------------------------------------------------------------------
a.q@speed-25.encs.concordia.ca BIP   0/0/32         0.01     lx-amd64
---------------------------------------------------------------------------------
a.q@speed-27.encs.concordia.ca BIP   0/0/32         0.01     lx-amd64
---------------------------------------------------------------------------------
```

```
g.q@speed-05.encs.concordia.ca BIP   0/0/32         0.02     lx-amd64
     144   100.00000 qsub-test nul-uge     r     12/03/2018 16:39:30    1
     62624 0.09843 case_talle x_yzabc      r     11/09/2021 16:50:09    32
---------------------------------------------------------------------------------
g.q@speed-17.encs.concordia.ca BIP   0/0/32         0.01     lx-amd64
---------------------------------------------------------------------------------
s.q@speed-07.encs.concordia.ca BIP   0/0/32         0.04     lx-amd64
---------------------------------------------------------------------------------
s.q@speed-08.encs.concordia.ca BIP   0/0/32         0.01     lx-amd64
---------------------------------------------------------------------------------
s.q@speed-09.encs.concordia.ca BIP   0/0/32         0.01     lx-amd64
---------------------------------------------------------------------------------
s.q@speed-10.encs.concordia.ca BIP   0/32/32        32.72    lx-amd64
     62624 0.09843 case_talle x_yzabc      r     11/09/2021 16:50:09    32
---------------------------------------------------------------------------------
s.q@speed-11.encs.concordia.ca BIP   0/32/32        32.08    lx-amd64
     62679 0.14212 CWLR_DF    a_bcdef      r     11/10/2021 17:25:19    32
---------------------------------------------------------------------------------
s.q@speed-12.encs.concordia.ca BIP   0/32/32        32.10    lx-amd64
     62749 0.09000 CLOUDY     z_abc        r     11/11/2021 21:58:12    32
---------------------------------------------------------------------------------
s.q@speed-15.encs.concordia.ca BIP   0/4/32         0.03     lx-amd64
     62753 82.47478 matlabLDPa b_bpxez      r     11/12/2021 08:49:52     4
---------------------------------------------------------------------------------
s.q@speed-16.encs.concordia.ca BIP   0/32/32        32.31    lx-amd64
     62751 0.09000 CLOUDY     z_abc        r     11/12/2021 06:03:54    32
---------------------------------------------------------------------------------
s.q@speed-19.encs.concordia.ca BIP   0/32/32        32.22    lx-amd64
---------------------------------------------------------------------------------
...
---------------------------------------------------------------------------------
s.q@speed-35.encs.concordia.ca BIP   0/32/32        2.78     lx-amd64
     62754 7.22952 qlogin-tes a_tiyuu      r     11/12/2021 10:31:06    32
---------------------------------------------------------------------------------
s.q@speed-36.encs.concordia.ca BIP   0/0/32         0.03     lx-amd64
etc.
```

Remember that you only have 30 seconds before the job is essentially over, so if you do not see a similar output, either adjust the sleep time in the script, or execute the `qstat` statement more quickly. The `qstat` output listed above shows you that your job is running on node `speed-05`, that it has a job number of 144, that it was started at 16:39:30 on 12/03/2018, and that it is a single-core job (the default).

Once the job finishes, there will be a new file in the directory that the job was started from, with the syntax of, `"job name".o"job number"`, so in this example the file is, qsub `test.o144`. This file represents the standard output (and error, if there is any) of the job in question. If you look at the contents of your newly created file, you will see that it contains the output of the, `module list` command. Important information is often written to this file.

Congratulations on your first job!

2.4 Common Job Management Commands Summary

Here are useful job-management commands:

- `qsub ./<myscript>.sh`: once that your job script is ready, on `speed-submit` you can submit it using this
- `qstat -f -u <ENCSusername>`: you can check the status of your job(s)
- `qstat -f -u "*"`: display cluster status for all users.
- `qstat -j [job-ID]`: display job information for [job-ID] (said job may be actually running, or waiting in the queue).
- `qdel [job-ID]`: delete job [job-ID].
- `qhold [job-ID]`: hold queued job, [job-ID], from running.
- `qrls [job-ID]`: release held job [job-ID].
- `qacct -j [job-ID]`: get job stats. for completed job [job-ID]. `maxvmem` is one of the more useful stats.

2.5 Advanced `qsub` Options

In addition to the basic `qsub` options presented earlier, there are a few additional options that are generally useful:

- `-m bea`: requests that the scheduler e-mail you when a job (b)egins; (e)nds; (a)borts. Mail is sent to the default address of, `"username@encs.concordia.ca"`, unless a different address is supplied (see, `-M`). The report sent when a job ends includes job runtime, as well as the maximum memory value hit (`maxvmem`).
- `-M email@domain.com`: requests that the scheduler use this e-mail notification address, rather than the default (see, `-m`).
- `-v variable[=value]`: exports an environment variable that can be used by the script.
- `-l h_rt=[hour]:[min]:[sec]`: sets a job runtime of HH:MM:SS. Note that if you give a single number, that represents *seconds*, not hours.
- `-hold_jid [job-ID]`: run this job only when job [job-ID] finishes. Held jobs appear in the queue. The many `qsub` options available are read with, `man qsub`. Also note that `qsub` options can be specified during the job-submission command, and these *override* existing script options (if present). The syntax is, `qsub [options] /PATHTOSCRIPT`, but unlike in the script, the options are specified without the leading `#$` (e.g., `qsub -N qsub-test -cwd -l h_vmem=1G ./tcsh.sh`).

2.6 Array Jobs

Array jobs are those that start a batch job or a parallel job multiple times. Each iteration of the job array is called a task and receives a unique job ID.

To submit an array job, use the `t` option of the `qsub` command as follows:

```
qsub -t n[-m[:s]] <batch_script>
```

-t Option Syntax:

- `n`: indicates the start-id.
- `m`: indicates the max-id.
- `s`: indicates the step size.

Examples:

- `qsub -t 10 array.sh`: submits a job with 1 task where the task-id is 10.
- `qsub -t 1-10 array.sh`: submits a job with 10 tasks numbered consecutively from 1 to 10.
- `qsub -t 3-15:3 array.sh`: submits a jobs with 5 tasks numbered consecutively with step size 3 (task-ids 3,6,9,12,15).

Output files for Array Jobs:

The default and output and error-files are `job_name.[o|e]job_id` and `job_name.[o|e]job_id.task_id`. This means that Speed creates an output and an error-file for each task generated by the array-job as well as one for the super-ordinate array-job. To alter this behavior use the `-o` and `-e` option of `qsub`.

For more details about Array Job options, please review the manual pages for `qsub` by executing the following at the command line on speed-submit `man qsub`.

2.7 Requesting Multiple Cores (i.e., Multithreading Jobs)

For jobs that can take advantage of multiple machine cores, up to 32 cores (per job) can be requested in your script with:

```
#$ -pe smp [#cores]
```

Do not request more cores than you think will be useful, as larger-core jobs are more difficult to schedule. On the flip side, though, if you are going to be running a program that scales out to the maximum single-machine core count available, please (please) request 32 cores, to avoid node oversubscription (i.e., to avoid overloading the CPUs).

Core count associated with a job appears under, "states", in the, `qstat -f -u "*"`, output.

2.8 Interactive Jobs

Job sessions can be interactive, instead of batch (script) based. Such sessions can be useful for testing and optimising code and resource requirements prior to batch submission. To request an interactive job session, use, `qlogin [options]`, similarly to a `qsub` command-line job (e.g., `qlogin -N qlogin-test -l h_vmem=1G`). Note that the options that are available for `qsub` are not necessarily available for `qlogin`, notably, `-cwd`, and, `-v`.

2.9 Scheduler Environment Variables

The scheduler presents a number of environment variables that can be used in your jobs. Three of the more useful are `TMPDIR`, `SGE_O_WORKDIR`, and `NSLOTS`:

- `$TMPDIR`=the path to the job's temporary space on the node. It *only* exists for the duration of the job, so if data in the temporary space are important, they absolutely need to be accessed before the job terminates.

- `$SGE_O_WORKDIR`=the path to the job's working directory (likely an NFS-mounted path). If, `-cwd`, was stipulated, that path is taken; if not, the path defaults to your home directory.
- `$NSLOTS`=the number of cores requested for the job. This variable can be used in place of hardcoded thread-request declarations.

In Figure 2 is a sample script, using all three.

```
#!/encs/bin/tcsh

#$ -N envs
#$ -cwd
#$ -pe smp 8
#$ -l h_vmem=32G

cd $TMPDIR
mkdir input
rsync -av $SGE_O_WORKDIR/references/ input/
mkdir results
STAR --inFiles $TMPDIR/input --parallel $NSLOTS --outFiles $TMPDIR/results
rsync -av $TMPDIR/results/ $SGE_O_WORKDIR/processed/
```

Figure 2: Source code for `tmpdir.sh`

2.10 SSH Keys For MPI

Some programs effect their parallel processing via MPI (which is a communication protocol). An example of such software is Fluent. MPI needs to have 'passwordless login' set up, which means SSH keys. In your NFS-mounted home directory:

- `cd .ssh`
- `ssh-keygen -t ed25519` (default location; blank passphrase)
- `cat id_ed25519.pub >> authorized_keys` (if the `authorized_keys` file already exists) *OR* `cat id_ed25519.pub > authorized_keys` (if does not)
- Set file permissions of `authorized_keys` to 600; of your NFS-mounted home to 700 (note that you likely will not have to do anything here, as most people will have those permissions by default).

2.11 Example Job Script: Fluent

The job script in Figure 3 runs Fluent in parallel over 32 cores. Of note, we have requested e-mail notifications (`-m`), are defining the parallel environment for, `fluent`, with, `-sgepe smp` (**very important**), and are setting `$TMPDIR` as the in-job location for the "moment" `rfile.out` file (in-job, because the last line of the script copies everything from `$TMPDIR` to a directory in the user's NFS-mounted home). Job progress can be monitored by examining the standard-out file (e.g., `flu10000.o249`), and/or by examining the "moment" file in `/disk/nobackup/<yourjob>` (hint: it starts with your job-ID) on the node running the job. **Caveat:** take care with journal-file file paths.

```
#!/encs/bin/tcsh

#$ -N flu10000
#$ -cwd
#$ -m bea
#$ -pe smp 8
#$ -l h_vmem=160G

module load ansys/19.0/default
cd $TMPDIR

fluent 3ddp -g -i $SGE_O_WORKDIR/fluentdata/info.jou -sgepe smp > call.txt

rsync -av $TMPDIR/ $SGE_O_WORKDIR/fluentparallel/
```

Figure 3: Source code for `fluent.sh`

2.12 Java Jobs

Jobs that call `java` have a memory overhead, which needs to be taken into account when assigning a value to `h_vmem`. Even the most basic `java` call, `java -Xmx1G -version`, will need to have, `-l h_vmem=5G`, with the 4-GB difference representing the memory overhead. Note that this memory overhead grows proportionally with the value of `-Xmx`. To give you an idea, when `-Xmx` has a value of 100G, `h_vmem` has to be at least 106G; for 200G, at least 211G; for 300G, at least 314G.

2.13 Scheduling On The GPU Nodes

The primary cluster has two GPU nodes, each with six Tesla (CUDA-compatible) P6 cards: each card has 2048 cores and 16GB of RAM. Though note that the P6 is mainly a single-precision card, so unless you need the GPU double precision, double-precision calculations will be faster on a CPU node.

Job scripts for the GPU queue differ in that they do not need these statements:

```
#$ -pe smp <threadcount>
#$ -l h_vmem=<memory>G
```

But do need this statement, which attaches either a single GPU, or, two GPUs, to the job:

```
#$ -l gpu=[1|2]
```

Single-GPU jobs are granted 5 CPU cores and 80GB of system memory, and dual-GPU jobs are granted 10 CPU cores and 160GB of system memory. A total of *four* GPUs can be actively attached to any one user at any given time.

Once that your job script is ready, you can submit it to the GPU queue with:

```
qsub -q g.q ./<myscript>.sh
```

And you can query `nvidia-smi` on the node that is running your job with:

```
ssh <username>@speed[-05|-17] nvidia-smi
```

Status of the GPU queue can be queried with:

```
qstat -f -u "*" -q g.q
```

Very important note regarding TensorFlow and PyTorch: if you are planning to run TensorFlow and/or PyTorch multi-GPU jobs, do not use the `tf.distribute` and/or `torch.nn.DataParallel` functions, as they will crash the compute node (100% certainty). This appears to be the current hardware's architecture's defect. The workaround is to either manually effect GPU parallelisation (TensorFlow has an example on how to do this), or to run on a single GPU.

Important

Users without permission to use the GPU nodes can submit jobs to the `g.q` queue but those jobs will hang and never run.

There are two GPUs in both `speed-05` and `speed-17`, and one in `speed-19`. Their availability is seen with, `qstat -F g` (note the capital):

```
queuename                      qtype resv/used/tot. load_avg arch          states
---------------------------------------------------------------------------------
...
---------------------------------------------------------------------------------
g.q@speed-05.encs.concordia.ca BIP   0/0/32         0.04     lx-amd64
        hc:gpu=6
---------------------------------------------------------------------------------
g.q@speed-17.encs.concordia.ca BIP   0/0/32         0.01     lx-amd64
        hc:gpu=6
---------------------------------------------------------------------------------
...
---------------------------------------------------------------------------------
s.q@speed-19.encs.concordia.ca BIP   0/32/32        32.37    lx-amd64
        hc:gpu=1
---------------------------------------------------------------------------------
etc.
```

This status demonstrates that all five are available (i.e., have not been requested as resources). To specifically request a GPU node, add, `-l g=[#GPUs]`, to your `qsub` (statement/script) or `qlogin` (statement) request. For example, `qsub -l h_vmem=1G -l g=1 ./count.sh`. You will see that this job has been assigned to one of the GPU nodes:

```
queuename                      qtype resv/used/tot. load_avg arch          states
---------------------------------------------------------------------------------
g.q@speed-05.encs.concordia.ca BIP 0/0/32 0.01 lx-amd64  hc:gpu=6
---------------------------------------------------------------------------------
g.q@speed-17.encs.concordia.ca BIP 0/0/32 0.01 lx-amd64  hc:gpu=6
---------------------------------------------------------------------------------
s.q@speed-19.encs.concordia.ca BIP 0/1/32 0.04 lx-amd64  hc:gpu=0 (haff=1.000000)
       538 100.00000 count.sh   sbunnell     r     03/07/2019 02:39:39     1
---------------------------------------------------------------------------------
etc.
```

And that there are no more GPUs available on that node (`hc:gpu=0`). Note that no more than two GPUs can be requested for any one job.

3 Creating Virtual Environments

The following documentation is specific to the **Speed** HPC Facility at the Gina Cody School of Engineering and Computer Science.

3.1 Anaconda

To create an anaconda environment in your speed-scratch directory, use the `prefix` option when executing `conda create`. For example, to create an anaconda environment for `ai_user`, execute the following at the command line:

```
conda create --prefix /speed-scratch/a_user/myconda
```

Note: Without the `prefix` option, the `conda create` command creates the environment in texttta_user's home directory by default.

3.1.1 List Environments

To view your conda environments, type: `conda info --envs`

```
# conda environments:
#
base                  *  /encs/pkg/anaconda3-2019.07/root
                         /speed-scratch/a_user/myconda
```

3.1.2 Activate an Environment

Activate the environment `speedscratcha_usermyconda` as follows

```
conda activate /speed-scratch/a_user/myconda
```

After activating your environment, add `pip` to your environment by using

```
conda install pip
```

This will install `pip` and `pip`'s dependencies, including python, into the environment.

Important Note: `pip` (and `pip3`) are used to install modules from the python distribution while `conda install` installs modules from anaconda's repository.

3.2 CUDA

When calling `CUDA` within job scripts, it is important to create a link to the desired `CUDA` libraries and set the runtime link path to the same libraries. For example, to use the `cuda-11.5` libraries, specify the following in your Makefile.

```
-L/encs/pkg/cuda-11.5/root/lib64 -Wl,-rpath,/encs/pkg/cuda-11.5/root/lib64
```

In your job script, specify the version of `gcc` to use prior to calling cuda. For example: `module load gcc/8.4` or `module load gcc/9.3`

3.2.1 Special Notes for sending CUDA jobs to the GPU Queue

It is not possible to create a `qlogin` session on to a node in the **GPU Queue** (`g.q`). As direct logins to these nodes is not available, jobs must be submitted to the **GPU Queue** in order to compile and link.

We have several versions of CUDA installed in:

```
/encs/pkg/cuda-11.5/root/
/encs/pkg/cuda-10.2/root/
/encs/pkg/cuda-9.2/root
```

For CUDA to compile properly for the GPU queue, edit your Makefile replacing `usrlocalcuda` with one of the above.

3.3 efficientdet

The following steps describing how to create an efficientdet environment on *Speed*, were submitted by a member of Dr. Amer's research group.

- Enter your ENCS user account's speed-scratch directory `cd /speed-scratch/<encs_username>`
- load python `module load python/3.8.3` create virtual environment `python3 -m venv <env_name>` activate virtual environment `source <env_name>/bin/activate.csh` install DL packages for Efficientdet

```
pip install tensorflow==2.7.0
pip install lxml>=4.6.1
pip install absl-py>=0.10.0
pip install matplotlib>=3.0.3
pip install numpy>=1.19.4
pip install Pillow>=6.0.0
pip install PyYAML>=5.1
pip install six>=1.15.0
pip install tensorflow-addons>=0.12
pip install tensorflow-hub>=0.11
pip install neural-structured-learning>=1.3.1
pip install tensorflow-model-optimization>=0.5
pip install Cython>=0.29.13
pip install git+https://github.com/cocodataset/cocoapi.git#subdirectory=PythonAPI
```

4 Conclusion

The cluster is, "first come, first served", until it fills, and then job position in the queue is based upon past usage. The scheduler does attempt to fill gaps, though, so sometimes a single-core job of lower priority will schedule before a multi-core job of higher priority, for example.

4.1 Important Limitations

- New users are restricted to a total of 32 cores: write to `rt-ex-hpc@encs.concordia.ca` if you need more temporarily (256 is the maximum possible, or, 8 jobs of 32 cores each).

- Job sessions are a maximum of one week in length (only 24 hours, though, for interactive jobs).
- Scripts can live in your NFS-provided home, but any substantial data need to be in your cluster-specific directory (located at `/speed-scratch/<ENCSusername>/`).

  NFS is great for acute activity, but is not ideal for chronic activity. Any data that a job will read more than once should be copied at the start to the scratch disk of a compute node using `$TMPDIR` (and, perhaps, `$SGE_O_WORKDIR`), any intermediary job data should be produced in `$TMPDIR`, and once a job is near to finishing, those data should be copied to your NFS-mounted home (or other NFS-mounted space) from `$TMPDIR` (to, perhaps, `$SGE_O_WORKDIR`). In other words, IO-intensive operations should be effected locally whenever possible, saving network activity for the start and end of jobs.
- Your current resource allocation is based upon past usage, which is an amalgamation of approximately one week's worth of past wallclock (i.e., time spent on the node(s)) and CPU activity (on the node(s)).
- Jobs should NEVER be run outside of the province of the scheduler. Repeat offenders risk loss of cluster access.

4.2 Tips/Tricks

- Files/scripts must have Linux line breaks in them (not Windows ones).
- Use `rsync`, not `scp`, when moving data around.
- If you are going to move many many files between NFS-mounted storage and the cluster, `tar` everything up first.
- If you intend to use a different shell (e.g., `bash` [13]), you will need to source a different scheduler file, and will need to change the shell declaration in your script(s).
- The load displayed in `qstat` by default is `np_load`, which is load/#cores. That means that a load of, "1", which represents a fully active core, is displayed as 0.03 on the node in question, as there are 32 cores on a node. To display load "as is" (such that a node with a fully active core displays a load of approximately 1.00), add the following to your `.tcshrc` file: `setenv SGE_LOAD_AVG load_avg`
- Try to request resources that closely match what your job will use: requesting many more cores or much more memory than will be needed makes a job more difficult to schedule when resources are scarce.
- E-mail, `rt-ex-hpc AT encs.concordia.ca`, with any concerns/questions.

4.3 Use Cases

- HPC Committee's initial batch about 6 students (end of 2019):
  - 10000 iterations job in Fluent finished in < 26 hours vs. 46 hours in Calcul Quebec
- NAG's MAC spoofer analyzer [9, 8], such as `https://github.com/smokhov/atsm/tree/master/examples/flucid`

  - compilation of forensic computing reasoning cases about false or true positives of hardware address spoofing in the labs

- S4 LAB/GIPSY R&D Group's:

  - MARFCAT and MARFPCAT (OSS signal processing and machine learning tools for vulnerable and weak code analysis and network packet capture analysis) [11, 6, 3]
  - Web service data conversion and analysis
  - Forensic Lucid encoders (translation of large log data into Forensic Lucid [7] for forensic analysis)
  - Genomic alignment exercises

- Parna Niksirat, Adriana Daca, and Krzysztof Skonieczny. The effects of reduced-gravity on planetary rover mobility. *International Journal of Robotics Research*, 39(7):797–811, 2020

A History

A.1 Acknowledgments

The first 6 versions of this manual and early job script samples, Singularity testing and user support were produced/done by Dr. Scott Bunnell during his time at Concordia as a part of the NAG/HPC group. We thank him for his contributions.

A.2 Phase 1

Phase 1 of Speed was of the following configuration:

- Sixteen, 32-core nodes, each with 512 GB of memory and approximately 1 TB of volatile-scratch disk space.
- Five AMD FirePro S7150 GPUs, with 8 GB of memory (compatible with the Direct X, OpenGL, OpenCL, and Vulkan APIs).

B Frequently Asked Questions

B.1 "Disk quota exceeded" Error

B.1.1 Probably Cause

The `"Disk quota exceeded"` Error occurs when your application has run out of disk space to write to. On Speed this error can be returned when:

1. The `/tmp` directory on the speed node your application is running on is full and cannot be written to.
2. Your NFS-provided home is full and cannot be written to.

B.1.2 Possible Solutions

1. Use the **-cwd** job script option to make the directory that the job script is submitted from the `job working directory`, e.g., the directory to store all output files in.

2. As previously mentioned in this manual, it is recommended to use local disk space for IO intensive operations, however, as the `/tmp` on the speed machines is `1GB` in size it may be necessary for scripts to store temporary data elsewhere. Review the documentation for the module(s) you are calling within your script to determine how to set working directories for that application. The basic steps for this solution are:

   - Review the documentation for the module(s) you are calling within your script to determine how to set working directories for that application.
   - Create a directory in speed-scratch for your temporary files. For example, to create a subdirector called `tmp` for user **a_user**, in that user's speed-scratch directory execute: **mkdir m 750 speedscratcha_usertmp**
   - If necessary, create a subdirectory for recovery files as well: **mkdir m 750 speedscratcha_userrecovery**
   - Use the documentation to update your script(s) and command calls to use the working directories created in the previous steps.

B.1.3 Example of setting working directories for `COMSOL`

- Create directories for recovery, temporary, and configuration files. For example, to create these directories for **a_user**:

  ```
  mkdir -m 750 -p /speed-scratch/a_user/comsol/{recovery,tmp,config}
  ```

- Add the following command switches to the COMSOL command to use the directories created for **a_user** above:

  ```
  -recoverydir /speed-scratch/a_user/comsol/recovery
  -tmpdir /speed-scratch/a_user/comsol/tmp
  -configuration/speed-scratch/a_user/comsol/config
  ```

C Sister Facilities

Below is a list of resources and facilities similar to Speed at various capacities. Depending on your research group and needs, they might be available to you. They are not managed by HPC/NAG of AITS, so contact their respective representatives.

- `computation.encs` CPU only 3-machine cluster running longer jobs without a scheduler
- `apini.encs` cluster for teaching and MPI programming (see the corresponding course)
- Computer Science and Software Engineering (CSSE) Virya GPU Cluster (2 nodes totalling 16 V100 NVIDIA GPUs), contact Alexander Aric at `gpu-help AT encs` to request access if you are a CSSE member

- Dr. Maria Amer's VidPro group's nodes in Speed with additional V100 and P6 GPUs (use `a.q` for those nodes).
- Dr. Hassan Rivaz's `impactlab.encs` Lambda Labs station
- Dr. Ivan Contreras' servers
- Compute Canada / Calcul Quebec

References

[1] 3DS. Abaqus. [online], 2019–2021. https://www.3ds.com/products-services/simulia/products/abaqus/.

[2] ANSYS. FLUENT. [online], 2000–2012. http://www.ansys.com/Products/Simulation+Technology/Fluid+Dynamics/ANSYS+FLUENT.

[3] Amine Boukhtouta, Nour-Eddine Lakhdari, Serguei A. Mokhov, and Mourad Debbabi. Towards fingerprinting malicious traffic. In *Proceedings of ANT'13*, volume 19, pages 548–555. Elsevier, June 2013.

[4] Amy Brown and Greg Wilson, editors. *The Architecture of Open Source Applications: Elegance, Evolution, and a Few Fearless Hacks*, volume I. aosabook.org, March 2012. Online at http://aosabook.org.

[5] MathWorks. MATLAB. [online], 2000–2012. http://www.mathworks.com/products/matlab/.

[6] Serguei A. Mokhov. The use of machine learning with signal- and NLP processing of source code to fingerprint, detect, and classify vulnerabilities and weaknesses with MARFCAT. Technical Report NIST SP 500-283, NIST, October 2011. Report: http://www.nist.gov/manuscript-publication-search.cfm?pub_id=909407, online e-print at http://arxiv.org/abs/1010.2511.

[7] Serguei A. Mokhov. *Intensional Cyberforensics*. PhD thesis, Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada, September 2013. Online at http://arxiv.org/abs/1312.0466.

[8] Serguei A. Mokhov, Michael J. Assels, Joey Paquet, and Mourad Debbabi. Automating MAC spoofer evidence gathering and encoding for investigations. In Frederic Cuppens et al., editors, *Proceedings of The 7th International Symposium on Foundations & Practice of Security (FPS'14)*, LNCS 8930, pages 168–183. Springer, November 2014. Full paper.

[9] Serguei A. Mokhov, Michael J. Assels, Joey Paquet, and Mourad Debbabi. Toward automated MAC spoofer investigations. In *Proceedings of C3S2E'14*, pages 179–184. ACM, August 2014. Short paper.

[10] Serguei A. Mokhov and Scott Bunnell. Speed server farm: Gina Cody School of ENCS HPC facility. [online], 2018–2019. https://docs.google.com/presentation/d/1bWbGQvYsuJ4U2WsfLYp8S3yb4i7OdU7QDn3l_Q9mYis.

[11] Serguei A. Mokhov, Joey Paquet, and Mourad Debbabi. The use of NLP techniques in static code analysis to detect weaknesses and vulnerabilities. In Maria Sokolova and Peter van Beek, editors, *Proceedings of Canadian Conference on AI'14*, volume 8436 of *LNAI*, pages 326–332. Springer, May 2014. Short paper.

[12] Parna Niksirat, Adriana Daca, and Krzysztof Skonieczny. The effects of reduced-gravity on planetary rover mobility. *International Journal of Robotics Research*, 39(7):797–811, 2020.

[13] Chet Ramey. The Bourne-Again Shell. In Brown and Wilson [4]. http://aosabook.org/en/bash.html.

[14] Rob Schreiber. MATLAB. *Scholarpedia*, 2(6):2929, 2007. http://www.scholarpedia.org/article/MATLAB.

[15] The MARF Research and Development Group. The Modular Audio Recognition Framework and its Applications. [online], 2002–2014. http://marf.sf.net and http://arxiv.org/abs/0905.1235, last viewed May 2015.