Speed: The GCS ENCS Cluster

Scott Bunnell
Concordia University
Montreal, Quebec, Canada
sbunnell@encs.concordia.ca

Serguei A. Mokhov
Concordia University
Montreal, Quebec, Canada
serguei@encs.concordia.ca

Gillian Roper
Concordia University
Montreal, Quebec, Canada
groper@encs.concordia.ca

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

1 Introduction

We include basic information and tips and tricks in this document, as well as examples and references, including to the projects or papers the use(d) Speed. Feel free to contribute either sample jobs or references. Details are sent to the `hpc-ml` mailing list.

Some 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 GCS:
  `https://www.concordia.ca/ginacody/aits/speed.html`
  that 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 [9].

1.1 What It Comprises

- Twenty four (24), 32-core 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 It Is Ideal For

- Jobs that are too demanding of 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 [13], etc. There are also a number of commercial packages, a subject to licensing contributions, available, such as MATLAB [4, 12], Ansys, Fluent [1], 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
- A number of specific environments 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

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

To use the cluster you will need to be added to the LDAP group that governs access to `speed-submit.encs.concordia.ca`. Please submit your request to, `rt-ex-hpc AT encs.concordia.ca`, detailing who you are, your username (e.g., what you would use to access `login.encs.concordia.ca`, for example), and the lab that you are associated with. That same username will then be given a scheduler account. Once that you can SSH to `speed` (an alias for `speed-submit.encs.concordia.ca`), you will need to source the scheduler file:

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

You may consider adding the source request to your shell-startup environment (i.e., to your `.tcshrc` file). If sourcing has been successful, you have access to the scheduler commands. For example, if, `qstat -f -u "*"`, returns something non-error related, you are in business.

As a new user, **iff** you don't have yet `.tcshrc`, please execute the following the first time that you log in, in your home directory, *exactly* as it is written (it is all one line):

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

If you do have `.tcshrc` already, you may need to edit it instead using, e.g., `vim`.

That command prepares your environment (again, be careful overwriting an existing `.tcshrc`), provides you with a job template, and sets up your cluster-specific storage. Note that you need to either log out and back in, or execute a new shell, for the environment changes to be applied (important). And if you use `bash`, or another shell, please contact `rt-ex-hpc AT encs.concordia.ca`.

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 connect via PuTTY (or MobaXterm). All users are expected to have a basic understanding of Linux and its commonly used commands.

2.2 Job Submission Basics

Preparing your job for submission is fairly straightforward. Job scripts are broken into four main sections: make a copy of `template.sh` or any existing job script example on our GitHub, and base your job on that. The first section is the shell call, and that can be left alone, unless you want to use a different shell. The next section contains the options provided to the cluster scheduler, and `template.sh` provides the essentials:

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

```
#!/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`

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`).

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 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. np_load  arch  states
---------------------------------------------------------------------------------
l.q@speed-05.encs.concordia.ca BIP 0/0/32 0.00 lx-amd64
   144 100.00000 qsub-test nul-uge r 12/03/2018 16:39:30 1
---------------------------------------------------------------------------------
l.q@speed-17.encs.concordia.ca BIP 0/0/32 0.00 lx-amd64
---------------------------------------------------------------------------------
l.q@speed-19.encs.concordia.ca BIP 0/0/32 0.00 lx-amd64
---------------------------------------------------------------------------------
l.q@speed-20.encs.concordia.ca BIP 0/0/32 0.00 lx-amd64
---------------------------------------------------------------------------------
l.q@speed-21.encs.concordia.ca BIP 0/0/32 0.00 lx-amd64
---------------------------------------------------------------------------------
l.q@speed-22.encs.concordia.ca BIP 0/0/32 0.00 lx-amd64
---------------------------------------------------------------------------------
l.q@speed-31.encs.concordia.ca BIP 0/0/32 0.00 lx-amd64
---------------------------------------------------------------------------------
l.q@speed-32.encs.concordia.ca BIP 0/0/32 0.00 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.

Congratulations on your first job!

2.3 Job Management Commands

Here are useful job-management commands:

- `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.

2.4 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`).

2.5 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.

2.6 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`.

2.7 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 a 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.

2.8 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 rsa` (default location; blank passphrase)
- `cat id_rsa.pub >> authorized_keys` (if the `authorized_keys` file already exists) *OR* `cat id_rsa.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).

```
#!/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`

2.9 Example Job Script: Fluent

```
#!/encs/bin/tcsh

#$ -N flu10000
#$ -cwd
#$ -m bea
#$ -pe smp 32
#$ -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`

The job script in Figure 3 runs Fluent in parallel over 32 cores. Of note, I have requested e-mail notifications (`-m`), am defining the parallel environment for, `fluent`, with, `-sgepe smp` (very important), and am 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 my 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.

2.10 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.

2.11 Scheduling On The GPU Nodes

The 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
---------------------------------------------------------------------------------
l.q@speed-05.encs.concordia.ca BIP 0/0/32 0.01 lx-amd64  hc:gpu=2
---------------------------------------------------------------------------------
l.q@speed-17.encs.concordia.ca BIP 0/0/32 0.03 lx-amd64  hc:gpu=2
---------------------------------------------------------------------------------
l.q@speed-19.encs.concordia.ca BIP 0/0/32 0.01 lx-amd64  hc:gpu=1
---------------------------------------------------------------------------------
l.q@speed-20.encs.concordia.ca BIP 0/0/32 0.01 lx-amd64
---------------------------------------------------------------------------------
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
---------------------------------------------------------------------------------
l.q@speed-05.encs.concordia.ca BIP 0/0/32 0.01 lx-amd64  hc:gpu=2
---------------------------------------------------------------------------------
l.q@speed-17.encs.concordia.ca BIP 0/0/32 0.01 lx-amd64  hc:gpu=2
---------------------------------------------------------------------------------
l.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
```

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 Conclusion

3.1 Important Limitations

- As a new user, you are limited to 32 cores (a restriction that is removed for responsible users).
- Jobs are assigned a maximum runtime of one week, unless otherwise specified.
- Home directories (data directories) are served over NFS. NFS is great for acute activity, but crappy 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.

3.2 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` [11]), 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.

3.3 Use Cases

- HPC Committee's initial batch about 6 students:
  - 10000 iterations job in Fluent finished in < 26 hours vs. 46 hours in Calcul Quebec
- NAG's MAC spoofer analyzer [8, 7], 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) [10, 5, 2]
  - Web service data conversion and analysis
  - Forensic Lucid encoders (translation of large log data into Forensic Lucid [6] for forensic analysis)
  - Genomic alignment exercises

A History

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 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)
- Compute Canada / Calcul Quebec
- Computer Science and Software Engineering (CSSE) Virya GPU Cluster (2 nodes totalling 16 V100 NVIDIA GPUs), contact `gpu-help AT encs` to request access if you are CSSE member
- Dr. Maria Amer's VidPro group's nodes in Speed with additional V100 and P6 GPUs
- Dr. Hassan Rivaz's `impactlab.encs` Lambda Labs station
- Dr. Ivan Contreras' servers

References

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

[2] 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.

[3] 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`.

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

[5] 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`.

[6] 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`.

[7] 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.

[8] 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.

[9] 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`.

[10] 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.

[11] Chet Ramey. The Bourne-Again Shell. In Brown and Wilson [3]. `http://aosabook.org/en/bash.html`.

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

[13] 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.