Computing Resources for UofT Statistical Sciences Students

David Veitch

University of Toronto

February 14 2024

UNIVERSITY OF TORONTO

Agenda

1. Benefits of Utilizing Computing Resources
2. Best Practices for Navigating Computing Resources
3. Filezilla & Putty
4. General Approach to Running a Script
5. Mercury Example
6. Compute Canada Cluster
7. Debugging
8. Tips

Benefits of Utilizing Computing Resources

- Access to potentially thousands of computing cores, allowing for faster iteration and troubleshooting
- Large amounts of RAM and storage
- Can run code even when laptop not open

Example

Suppose you have a file TypeISimulations.R to test the Type I Error of your newly developed statistical method. Your method is bootstrap based, hence for each experiment you run, you must calculate 1000 bootstraps. On your laptop with 4 cores each experiment runs in one minute, meaning it will take roughly 4 hours to run.

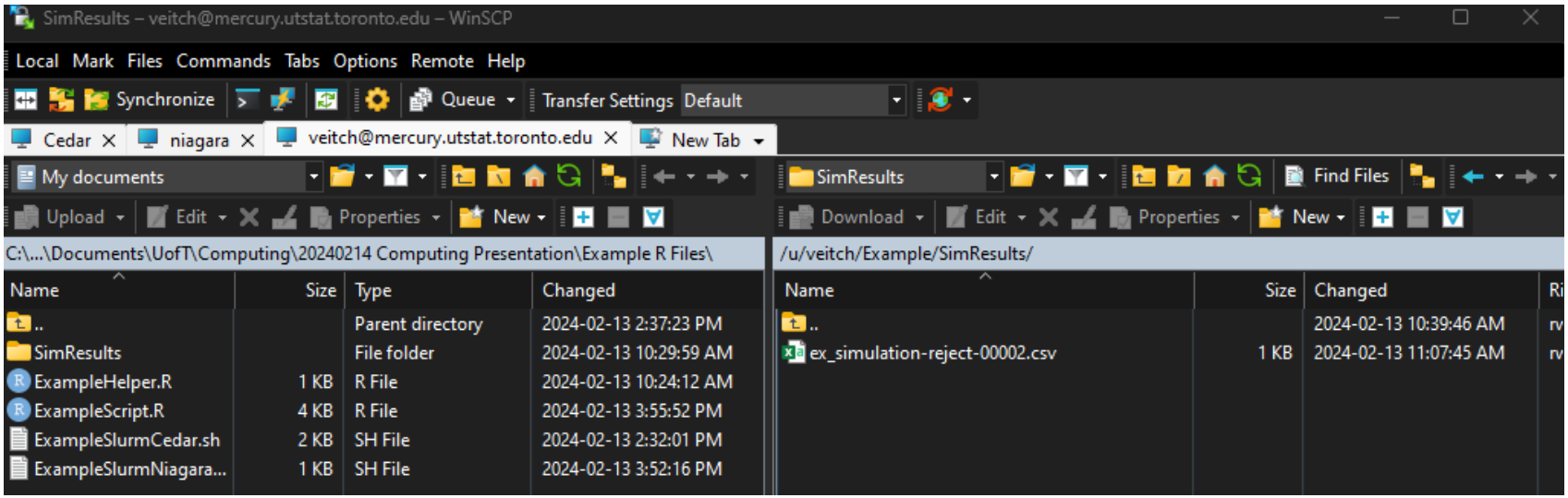
In comparison running this same file on the Mercury stats server ($\sim$ 100 cores) will take 10 minutes, and running this on the Compute Canada server should take 1 minute (assuming it has capacity).

Best Practices for Navigating Computing Resources

- Put in the effort upfront to learn how to use computing resources efficiently. This will make your life significantly easier overtime.
- There are lots of resources out there to help you navigate these computing resources. Most of your questions can be answered via these resources:
  - Mercury: `https://www.statistics.utoronto.ca/resources/departmental-computing-resources`, statstech@utoronto.ca
  - Compute Canada: `https://docs.alliancecan.ca/wiki/Technical_documentation`, `https://docs.alliancecan.ca/wiki/Niagara_Quickstart`
- Sometimes the best way to learn is from each other!

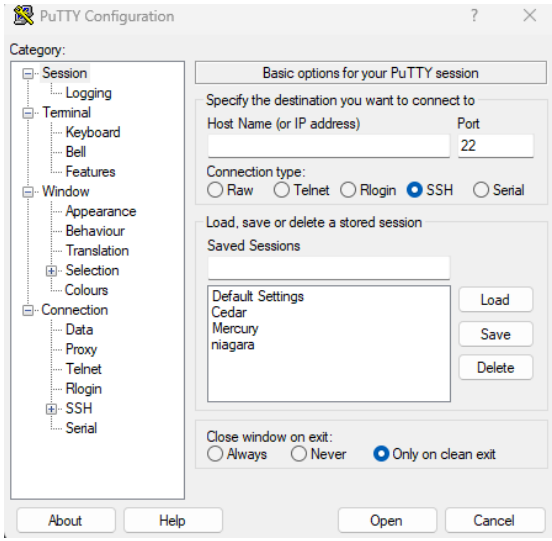
WinSCP

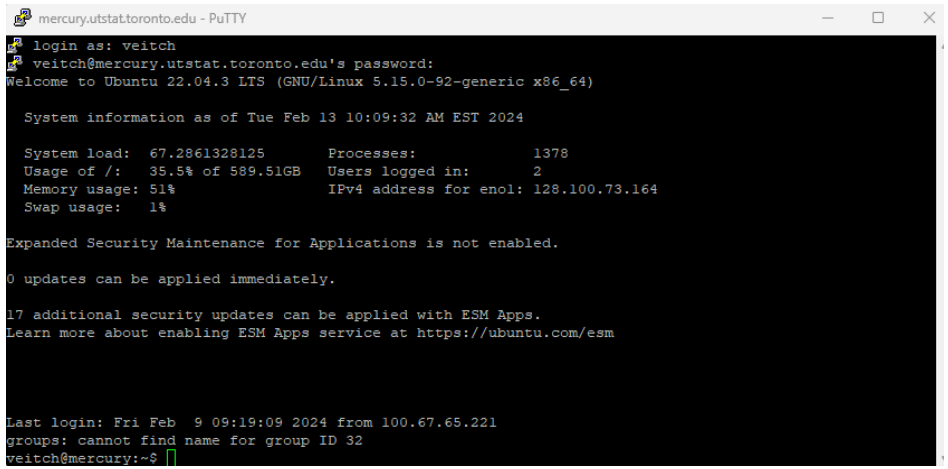
WinSCP is a FTP (file transfer protocol). Essentially you use it to view, and transfer (drag and drop), files onto the server. Useful if your R scripts produce outputs such as CSV files.



Putty

Putty is a SSH client which lets you securely connect to another computer remotely. In this case you are directly connecting to Compute Canada or Statistics computing resources.





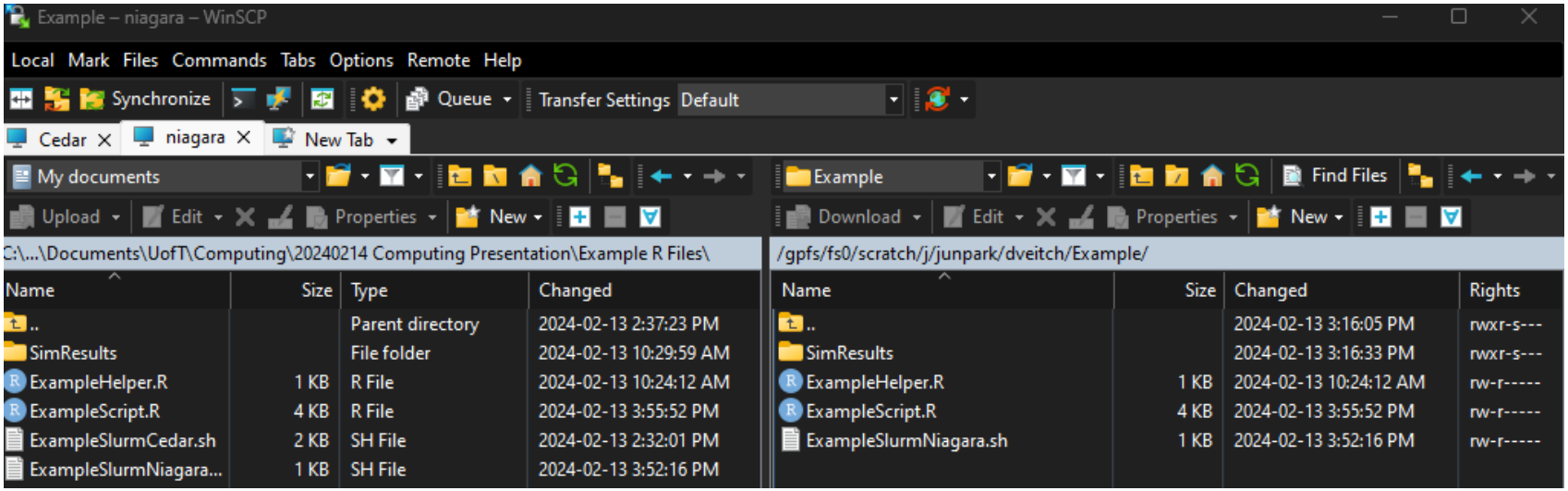
General Approach to Running a Script

Good practice to have two files

1. File 1 - Script to run simulations
   - Contains a number of functions, each corresponding to a specific simulation being run (tip: arrange by date to make it easy to find old simulations)
   - Run this script, and pass in argument `function_call` via Putty which specifies exactly which simulation to run
2. File 2 - Helper functions
   - Often there are functions shared across simulations that are used. For example if you are testing the Type I error of `function=new_method()` it makes sense to store this function in a separate file and then just import it into your simulation.

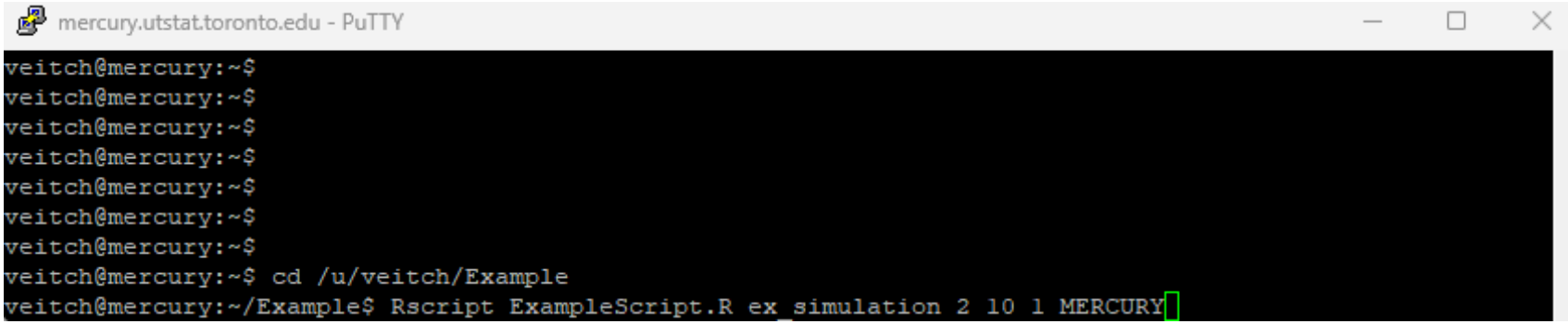
General Approach to Running a Script

Upload file to server via WinSCP



General Approach to Running a Script

Use the command line in Putty to navigate the directory with the file (using the cd command), and then run the script using the Rscript command. Note the arguments passed in after ExampleScript.R will serve as inputs into the script.



General Approach to Running a Script

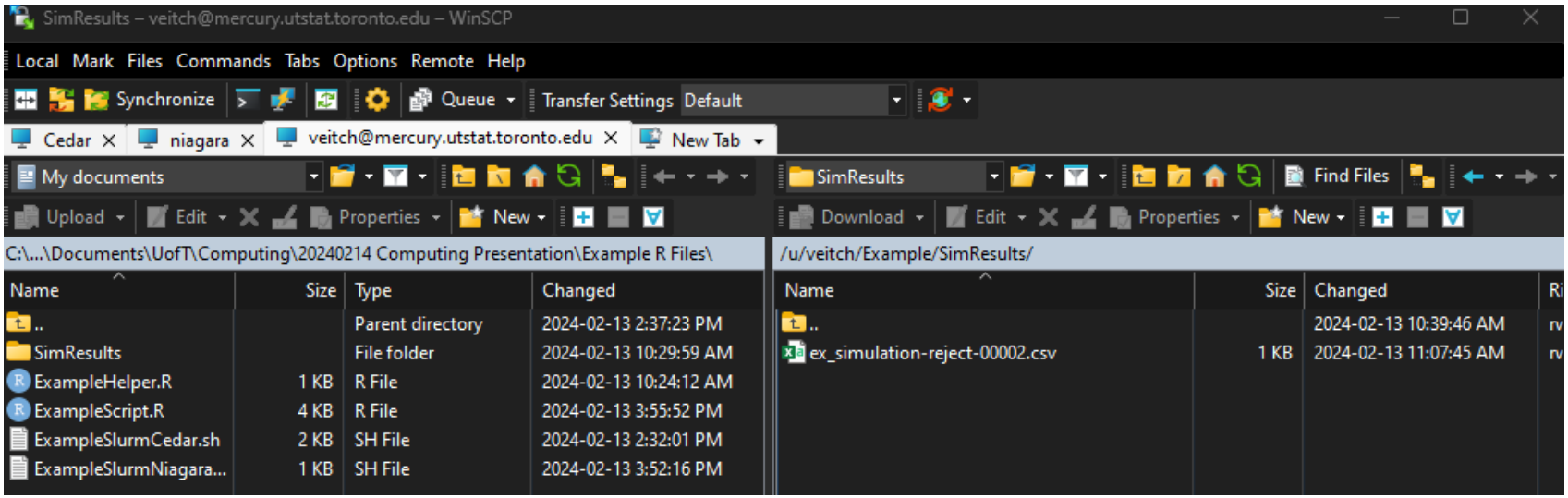
The script will run.

mercury.utstat.toronto.edu - PuTTY

```
veitch@mercury:~$
veitch@mercury:~$
veitch@mercury:~$ cd /u/veitch/Example
veitch@mercury:~/Example$ Rscript ExampleScript.R ex_simulation 2 10 1 MERCURY
Loading required package: iterators
Loading required package: parallel
[1] 5
[1] "Compute Server MERCURY"
[1] "Args Passed"
[1] "ex_simulation" "2"             "10"            "1"
[5] "MERCURY"
[1] "ex_simulation function name"
[1] "2 job number"
[1] "10 number of cores"
[1] "1 total nodes for whole job"
[1] "actual workers 10"
[1] "loaded helper functions"
starting worker pid=656504 on localhost:11116 at 11:07:43.793
starting worker pid=656499 on localhost:11116 at 11:07:43.812
starting worker pid=656501 on localhost:11116 at 11:07:43.814
starting worker pid=656496 on localhost:11116 at 11:07:43.853
starting worker pid=656503 on localhost:11116 at 11:07:43.887
starting worker pid=656497 on localhost:11116 at 11:07:43.904
starting worker pid=656500 on localhost:11116 at 11:07:43.916
starting worker pid=656505 on localhost:11116 at 11:07:43.919
starting worker pid=656498 on localhost:11116 at 11:07:43.926
starting worker pid=656502 on localhost:11116 at 11:07:43.930
veitch@mercury:~/Example$
```

General Approach to Running a Script

Here we ran job number 2, and the script produced a file `ex_simulation-reject-00002.csv` which contain its results. These could then be downloaded to a local machine to be analyzed and saved for later.



General Approach to Running a Script - Unpacking Arguments

When you run a R script from the command line it looks as follows

```
Rscript ScriptName.R arg1 arg2
```

Where ScriptName.R is the name of the R script you want to run and arg1 arg2 are optional arguments. If no optional arguments just use `Rscript ScriptName.R`). Can have as many arguments as you would like (e.g. `Rscript ScriptName.R type1 iid 1000`)

These optional arguments are very helpful as they allow us to run different parts of the Rscript without having to upload a new .R file.

General Approach to Running a Script - Unpacking Arguments

Here we are unpacking the numerous arguments we have passed into our R script via the command line. This allows us to run exactly what we want to run, without having to upload a new script.

```
# Unpack Arguments Being Passed Into R, here there are 5 arguments
# args[1] - name of function to run
# args[2] - job number, or 'ALL'; used to specify
# args[3] - number of cores to parallelize across
# args[4] - total nodes to use (only for Niagara), set to 1
# args[5] - name of compute server (e.g. 'NIAGARA', 'CEDAR', 'MERCURY')
args=(commandArgs(TRUE))
print('Args Passed')
print(args)
function_call=as.character(args[1])
if(as.character(args[2])=='ALL'){
  job_num=as.character(args[2])
}else{
  job_num=as.numeric(args[2])
}
ncores=as.numeric(args[3])
total_nodes_to_use=as.numeric(args[4])
print(paste(function_call,'function name'))
print(paste(job_num,'job number'))
print(paste(ncores,'number of cores'))
print(paste(total_nodes_to_use,'total nodes for whole job'))
registerDoParallel(cores=ncores)# Shows the number of Parallel Workers t requested.
print(paste('actual workers',as.character(getDoParWorkers()))) # you can compare with the number of actual workers
```

General Approach to Running a Script - Parallelized Code

- Here is an example of the code that is running for this example script.
- Notice the last line `docall(·)` will run the function `function_call` which was passed in via the command line.
- The `foreach(·)` function parallelizes our code across multiple cores and then combines the output in `experiment_results`. Good tutorial here: `https://privefl.github.io/blog/a-guide-to-parallelism-in-r/`

```
###################### EXAMPLE SIMULATION ##############################
ex_simulation<-function(job_num,results_dir,working_dir){

  if(job_num=='ALL'){
    start_seed=999
  }else{start_seed=job_num}

  cl <- parallel::makeCluster(ncores,outfile="")
  doParallel::registerDoParallel(cl)

  experiment_results=foreach(i=1:10, .packages = c('Matrix'),.combine='rbind')%dopar%{
                              set.seed(i+start_seed)
                              setwd(working_dir)
                              source('ExampleHelper.R')
                              x=sim_normal_rv(1,5)

                              x
                            }
  parallel::stopCluster(cl)

  setwd(results_dir)
  write.csv(experiment_results, file = paste(function_call,'-reject-',sprintf("%05d",job_num),'.csv',sep=''))
}

###################### RUN CODE ##############################
do.call(function_call,list(job_num,results_dir,working_dir))
```

Mercury Example

Live coding example

Mercury Example

- Mercury has 128 cores.
- Shared across stats department. Can see how many cores are being used via `top` command in command line. Often not heavily used; although if others using it try to limit number of cores being used. Here we see 9 tasks running, 8 of which would be fully using a core.

```
top - 11:22:47 up 18 days,  2:11, 10 users,  load average: 54.21, 55.91, 63.27
Tasks: 1382 total,   9 running, 1357 sleeping,  16 stopped,   0 zombie
%Cpu(s): 68.9 us,  0.0 sy,  0.0 ni, 31.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem : 257304.5 total,  30015.8 free, 105988.1 used, 121300.6 buff/cache
MiB Swap:   8192.0 total,   8007.3 free,    184.7 used. 149051.3 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
3806034 galleg24  20   0   10.6g 729536  13700 R  3992   0.3 150034:34 R
3805511 galleg24  20   0 9646852 693640  13716 R  3992   0.3 150766:45 R
3805547 galleg24  20   0 9572716 627736  14068 R 299.3   0.2 163608:42 R
 658878 esqfuen+  20   0 8857876 391404   9424 R 100.0   0.1   5:47.45 R
 658879 esqfuen+  20   0 8858444 391916   9424 R 100.0   0.1   5:47.45 R
3216570 marija    20   0   47280  28300   4020 R 100.0   0.0  19107:48 mla_c_rp2_h4_LO
3216572 marija    20   0   47952  28884   3784 R 100.0   0.0  19111:36 mla_c_rp2_h4_LO
3806237 galleg24  20   0 9663176 707188  13776 R 100.0   0.3 149958:10 R
 646534 ruizsuar  20   0  750624 134256  34120 S  18.7   0.1   5:17.69 node
 659964 veitch    20   0   11948   5432   3260 R   1.6   0.0   0:00.12 top
3216435 marija    20   0  332844 206548  16648 S   1.0   0.1 292:31.13 R
 646149 ruizsuar  20   0 1168176 340000  41860 S   0.7   0.1   4:23.85 node
 646205 ruizsuar  20   0 9038476 102552  15768 S   0.7   0.0   0:31.24 R
```

Mercury Example

If you run a script on Mercury, and then want to do something else, can use the `screen` command

- **Create a new screen** screen -S <screenname>
- **Run your script**
- **Detach from screen** Ctrl+A Ctrl+D
- **View screen sessions** screen -ls
- **Reconnect to screen** screen -r <screenname>
- **Kill Screen While Detached From It** screen -X -S <screenname> quit

Live coding example

Compute Canada Cluster

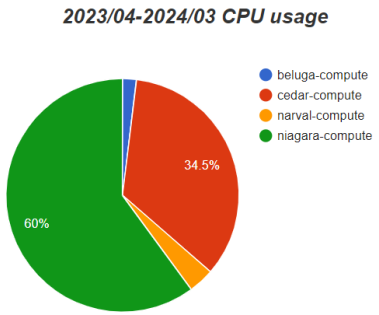
- The Compute Canada cluster is a resource shared across all Canadian universities which has far more cores (Niagara alone has +60,000 cores).
- Submitting jobs more complicated, may be in queue for awhile. However, potential to access so many cores can be helpful.

Compute Canada Cluster

- First must register for an account. List professor you are working under and they will approve account.
- Once approved you will be able to submit jobs to clusters across Canada (Beluga, Cedar, Graham, Narval, Niagara)
- Cedar probably best to submit to for most things (have to wait long time for Niagara sometimes)

Compute Canada Cluster

- Usage is tracked on Compute Canada website. From my understanding default allocation is 40 core years for a professor.
- However this is based on a rolling window of usage, not total usage. From my understanding if you use a lot of compute, for two weeks your jobs may be deprioritized, but after that they will have regular priority.
- If server not being fully utilized your script should run right away (early in morning, particularly on Monday).
- Compute Canada website has information on usage, try to not use to much as resources shared across group.



Resource	Total CPU Usage (in core years)	Projected CPU Usage (in core years)	Extra Info
niagara-compute	60.32	69.27	Show monthly usage Show submitter usage
cedar-compute	34.61	39.75	Show monthly usage Show submitter usage -->
narval-compute	3.59	4.12	Show monthly usage Show submitter usage
beluga-compute	1.93	2.21	Show monthly usage Show submitter usage
Total	**100.45**	**115.36**	

CPU usage on cedar-compute by submi

Person	Total CPU Usage (in core years)	Projected CPU Usage (in core years)
[redacted]	3.37	3.87
	4.16	4.78
	10.52	12.08
	1.53	1.75
	4.62	5.31
DAVID VEITCH	10.31	11.84

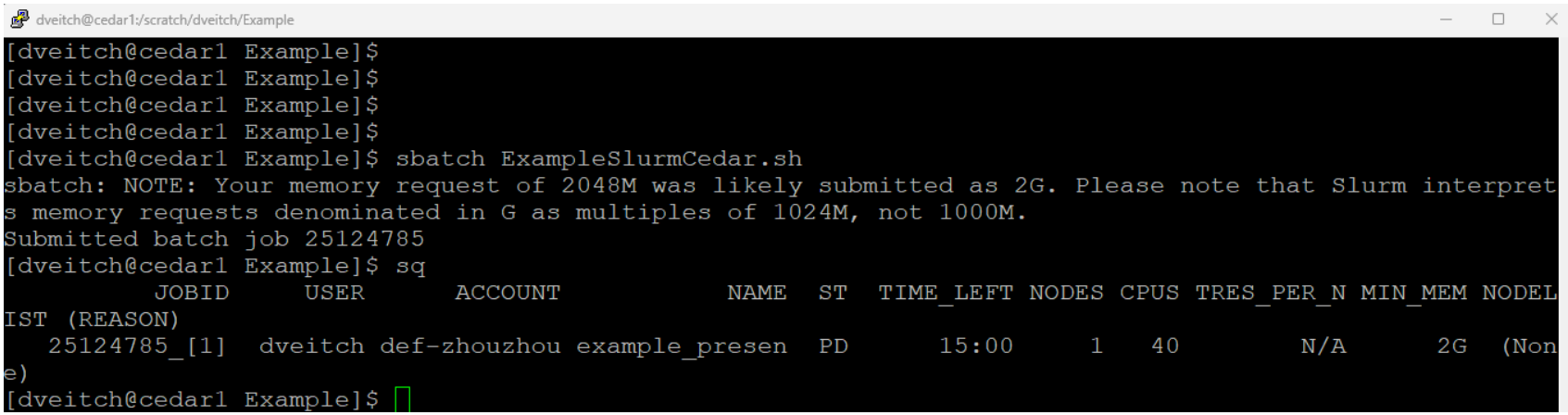
Compute Canada Cluster

- Two types of ways to run job on supercomputers, login nodes or SLURM
  - **Login Nodes**: use a small number of cores (e.g. 4) to test code
  - **SLURM**: use this to submit big jobs, gives instructions to supercomputer which will then schedule your job
- One wrinkle with Compute Canada is you will run your job and write your results in the `SCRATCH` directory. This directory automatically deletes files that haven't been used after a few months. Can move files to `PROJECT` directory or download to computer to get around this. Can also use GLOBUS file transfer system if need to move between SCRATCH and PROJECT.

Compute Canada Cluster - Submit Job

t

To submit job, you must create a .sh file (show example). Once this file is on the server, navigate to the directory it is in, and then use `sbatch` to run it. You can then use `sq` to see its status.



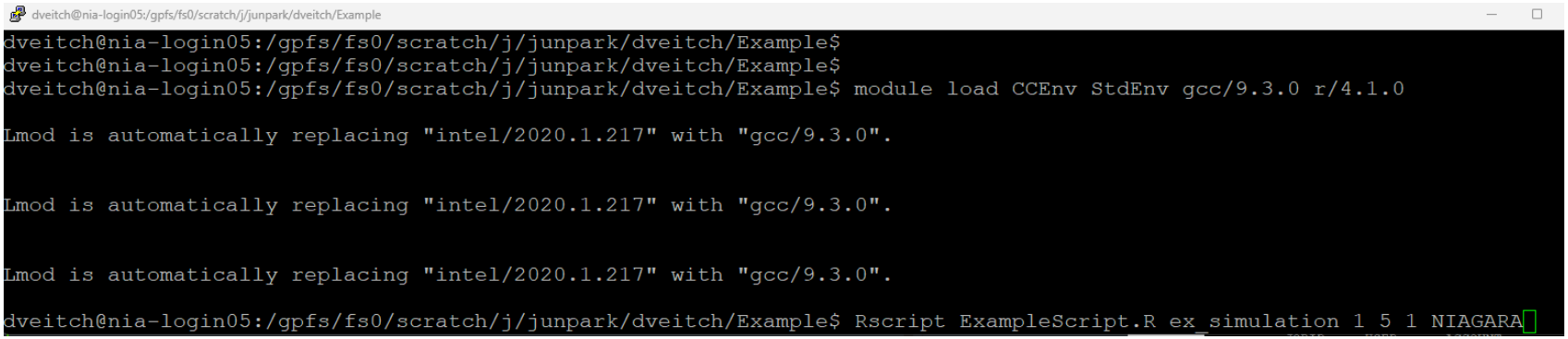
If we no longer wanted this job to run we could cancel it using `scancel` followed by its job number (e.g. `scancel 25124785`).

Compute Canada Cluster - Login Node

To run a job on the login node (for testing) you need to load certain environments and modules before you run the script. These are done on the command line, the commands are (can adjust version of R as needed)

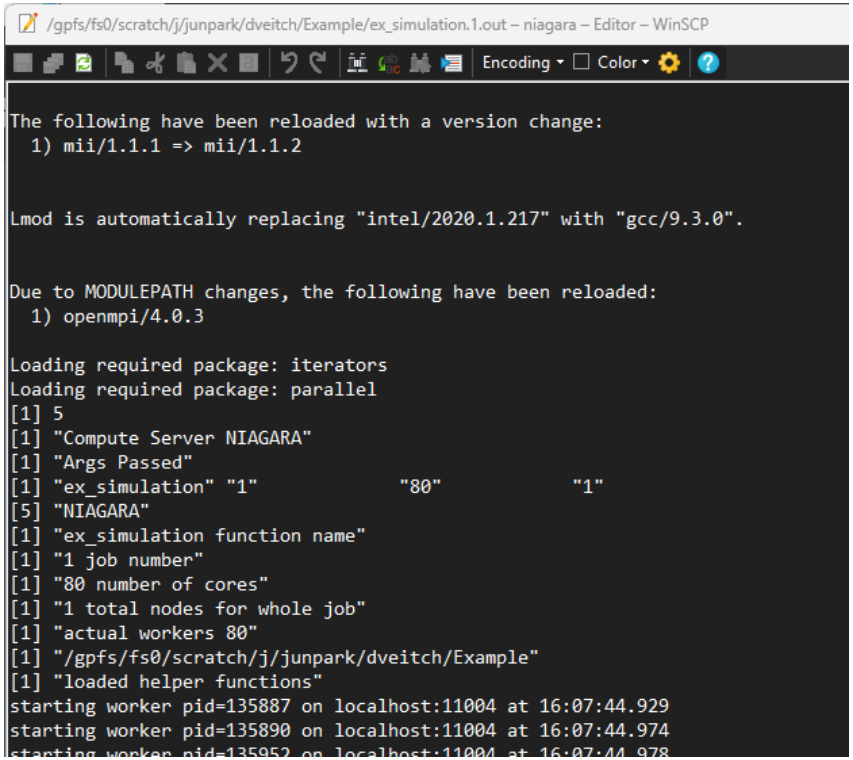
- **Cedar** module load StdEnv gcc/9.3.0 r/4.1.0
- **Niagara** module load CCEnv StdEnv gcc/9.3.0 r/4.1.0

Once this is done you can run Rscript ....



Debugging

If you run a job via sbatch the main way to debug is via the .out files which are created. This shows what the output would be if you ran something directly in the command line (including what R produces)



Sometimes helpful to have R print out text after each experiment finishes (helps with debugging which experiment is problematic)

Debugging

Another good way to debug (on Niagara only) is via debug nodes. More information here:
`https://docs.scinet.utoronto.ca/index.php/Slurm`,

In command line type

```
debugjob --clean 1
```

you can now run a Rscript using a full node (80 cores) in the command line (as opposed to only a few cores in login nodes).

Debugging

- Often you will run simulations across a variety of sample sizes, dimension, data generating processes.
- Bugs sometimes only show up in your code in a subset of the experiments you run.
- Strongly recommend debugging in advance. For example if running 100 different experiments with 1000 iterations each and 1000 bootstraps per iteration, try running all experiments with 10 iterations and 25 bootstraps to ensure no bugs.

Tips

- Short jobs can sometimes get around the queue, something called backfilling. Try under 3 hours jobs (believe Niagara has to be either 1 hour or 3 hours).
- Make sure to install R packages on the server for the correct version of R you are using.
- SciNet has many great resources (`https://www.youtube.com/@scinethpcattheuniversityof8962/videos`,`https://www.scinethpc.ca/events-3/`) and also does a good job responding to e-mails support@scinet.utoronto.ca
- Compute Canada just introduced two factor authentication which complicates some parts of logging into server. May have to use WinSCP instead of Filezilla.