Introduction to SciNet, Niagara & Mist

Bruno Mundim (SciNet)

April 13, 2022

Outline

- About SciNet
- Using Niagara and Mist
- Data management and I/O tips

About SciNet

SciNet is a consortium for high-performance computing of the U. of Toronto and associated hospitals.

- We run massively parallel computers to meet the needs of researchers across Canada.
- 5 similar consortia in Canada also provide academic Advanced Research Computing (ARC) resources.
- These consortia maintain and support a network of resources available to researchers across Canada, under a national allocation system.

National research computing clusters

- Four heterogeneous ("general purpose") clusters:
  - Cedar (Simon Fraser University)
  - Graham (University of Waterloo)
  - Béluga (Montréal, Québec)
  - Narval (Montréal, Québec)
- One homogeneous ("large parallel") cluster:
  - Niagara (University of Toronto)
- One homogeneous gpu cluster:
  - Mist (University of Toronto)
- Several cloud systems (Sherbrooke, Victoria, Waterloo).

What does SciNet do?

Systems

We host one of the largest supercomputers in Canada available to academics.

- Niagara

Plus some smaller ones

- Mist GPU cluster
- Rouge AMD GPU cluster
- Teach

And a longer-term storage facility

- HPSS

What else does SciNet do?

Training

- Intro to SciNet and Niagara, Linux Shell
- Scientific and Parallel Programming (C, C++, Fortran, R, Python, CUDA)
- Grad Courses on Scientific Computing , Data Analysis, and BioStatistics
- Data management, Parallel I/O, Databases, Machine learning, AI
- Ontario HPC summer school
- International HPC summer school (together with PRACE, XSEDE, RIKEN)

For full list see: https://education.scinet.utoronto.ca/

Research

https://www.scinet.utoronto.ca/research-scinet

SciNet people

Software, user support, training, etc..

- Mike Nolta
- Erik Spence
- Ramses van Zon
- Bruno Mundim
- Alexey Fedoseev
- James Willis
- Fei Mao (SOSCIP)
- Yohai Meiron (SOSCIP)

- Chief Technical Officer: Daniel Gruner

Hardware, systems, etc..

- Joseph Chen
- Ching-Hsing Yu
- Leslie Groer
- Jaime Pinto
- Marco Saldarriaga
- Vladimir Slavnic
- Ram Sharma

- Information Systems Security: Raphaelle Gauriau
- Business manager: Jackie Denholm

Reach all of us at once at **support@scinet.utoronto.ca**

Niagara

Niagara

- 80,960 x86-64 cores.
- 2,024 *Lenovo SD530* nodes
- Per node:
    - 40 Intel SkyLake/CascadeLake cores @ 2.4GHz
    - 188 GiB RAM per node (> 4 GiB per core)
- 3.6 PFlops sustained (6.25 PFlops theoretical).
  #59 on the Nov 2018 TOP500 (now #127)
- Operating system: Linux CentOS 7.
- Interconnect: InfiniBand Dragonfly+
  1:1 up to 432 nodes, 2:1 beyond that.
- Parallel shared file system for home, scratch, project
- Burst Buffer for fast I/O

Mist

Mist

- Niagara's little GPU sibling
- Also, for 70%, a SOSCIP system.
- 54 IBM Power-9 nodes with 4 GPUs.
- Per node:
  - 32 Power-9 cores @ 2.4GHz
  - 256 GB RAM per node
  - 4 NVIDIA "Volta" GPUs with 32GB
- 1 PFlops peak (1.6 PFlops theoretical).
- Operating system: Red Hat Enterprise 8.
- Interconnect: 1:1 InfiniBand Dragonfly+
- Same parallel shared file systems as Niagara

Using Niagara and Mist: Getting Access

1. Register with the Compute Canada Database (CCDB)

   https://ccdb.computecanada.ca/account_application

   If you're not a PI and your PI does not have a CC account, they have to get one first, so they can sponsor your account.

   The approval process typically takes 1-2 business days.

2. Go to

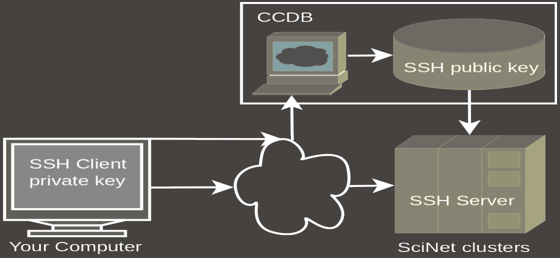
   https://ccdb.computecanada.ca/services/opt_in

   and click on the "Join" button next to Niagara and Mist.

3. After a business day or two (typically less), you get an email confirming your access to Niagara and Mist.

Using Niagara and Mist: Secure Login

- As with all SciNet and Compute Canada systems, access is via ssh only.
- Password authentication is disabled on Niagara and Mist, which means SSH keys must be used for authentication on Niagara and Mist.
- SSH keys come in a pair:
  - a **private key** which is kept on your own computer and used to connect
  - a **public key** that you upload to CCDB and which then propagates to the clusters.
- You can and should protect your private key with a passphrase.



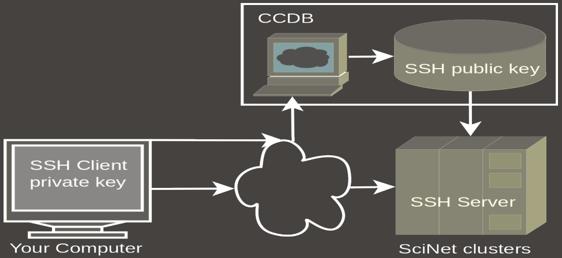
Note that you can use the same SSH keys for connecting to the other Compute Canada clusters as well.

Using Niagara and Mist: SSH key setup before first login

- To access SciNet systems for the first time, open a local terminal window on your computer (e.g. MobaXTerm).
- Then generate a ssh key pair with the following command:

```
laptop> ssh-keygen -t ed25519 -C "USERNAME@MYLAPTOP ccf" -f ~/.ssh/ccf_ed25519
```

- That will prompt you to enter a passphrase to protect your private key.
  Choose 15 characters or more. Two short sentences meaningful to you, for example.
- A private key, ccf_ed25519, and a public key, ccf_ed25519.pub are then created in the directory ".ssh" in your home directory.



- -C option allows you to insert a comment into the key.
- -f option specifies the filename of the key file.

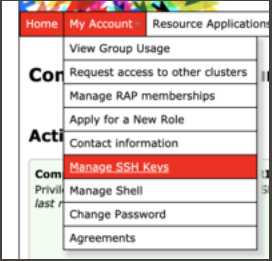
Using Niagara and Mist: Uploading Your Public Key

Once you created your ssh key pair, you need to make Niagara/Mist aware of the public part of your key.

- Step 1: Use your Compute Canada credentials to visit the following site:

  https://ccdb.computecanada.ca/ssh_authorized_keys

or via the CCDB menu:

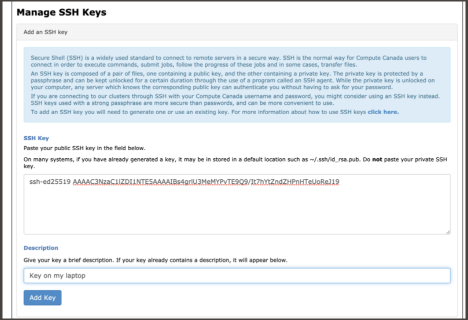


Using Niagara and Mist: Uploading Your Public Key

- Step 2: Grab your SSH public key:

```
laptop> cat ~/.ssh/ccf_ed25519.pub
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIEpDf+Wcvtru6pUcBgJQo/3+cmI4+MisfNE3U46/CDkx
USERNAME@MYLAPTOP ccf
```

- Step 3: Paste the public key into the CCDB form and click "Add Key" button:



Manage SSH Keys

Add an SSH key

Secure Shell (SSH) is a widely used standard to connect to remote servers in a secure way. SSH is the normal way for Compute Canada users to connect in order to execute commands, submit jobs, follow the progress of these jobs and in some cases, transfer files.

An SSH key is composed of a pair of files, one containing a public key, and the other containing a private key. The private key is protected by a passphrase and can be kept unlocked for a certain duration through the use of a program called an SSH agent. While the private key is unlocked on your computer, any server which knows the corresponding public key can authenticate you without having to ask for your password.

If you are connecting to our clusters through SSH with your Compute Canada username and password, you might consider using an SSH key instead. SSH keys used with a strong passphrase are more secure than passwords, and can be more convenient to use.

To add an SSH key you will need to generate one or use an existing key. For more information about how to use SSH keys **click here.**

SSH Key

Paste your public SSH key in the field below.

On many systems, if you have already generated a key, it may be in stored in a default location such as ~/.ssh/id_rsa.pub. Do **not** paste your private SSH key.

ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIBs4grlU3MeMYPvTE9Q9/It7hYtZndZHPnHTeUoReJ19

Description

Give your key a brief description. If your key already contains a description, it will appear below.

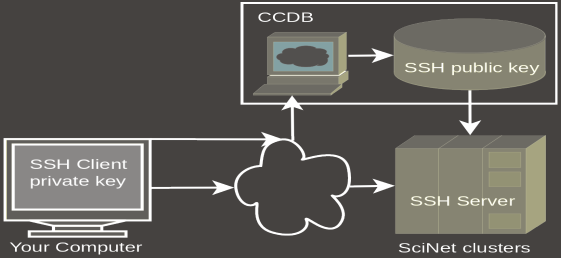
Key on my laptop

Using Niagara and Mist: Logging in

Wait a few minutes for your new uploaded public key to propagate to the systems and then ssh into the Niagara login nodes specifying the corresponding ssh private key:

```
laptop> ssh -Y -i ~/.ssh/ccf_ed25519 USERNAME@niagara.computecanada.ca
Enter passphrase for ccf_ed25519:
nia-login07:~$
```

- The optional `-Y` is needed to open windows from the Niagara command-line onto your local X server.
- `-i` option selects a file from which the identity (private key) for key authentication is read.
- For *Mist*, replace *niagara* with *mist*.



- First time? Check host key fingerprints against https://docs.computecanada.ca/wiki/SSH_host_keys

Connecting more conveniently: save ssh options and keys

Once you've logged in successfully, you can save the ssh options in `~/.ssh/config`:

```
Host niagara
   HostName niagara.computecanada.ca
   User USERNAME
   IdentityFile ~/.ssh/ccf_ed25519
   IdentitiesOnly yes
```

Now you can access Niagara by simply typing (in addition to your passphrase):

```
laptop> ssh niagara
```

This will also make data transfer commands like `scp` and `rsync` work more easily.

You can use the *ssh-agent* to hold your key for you by typing:

```
laptop> ssh-add ~/.ssh/ccf_ed25519
```

This will ask for the passphrase, and then save that key so you do not have to type the passphrase again during the session.

Ssh Key Best Practices

- Do not share your private keys!
- Always protect it with a strong passphrase!
- Never copy your private key to other systems!
- Create one key pair for each computer you use to access our systems.
- Create one key pair for each different service, role or domain, and name them accordingly.
- Do not create key pairs in shared systems like HPC clusters.

A reference to help you troubleshooting: https://docs.computecanada.ca/wiki/SSH_Keys

Niagara nodes: login, compute, and datamovers

There are three types of nodes on Niagara:

- The ***login nodes*** are where you develop, edit, compile, prepare and submit jobs.

  These login nodes are not part of the Niagara compute cluster, but have the same architecture, operating system, and software stack.

  These nodes are shared, i.e., multiple users are on the same nodes.

  These nodes have limits in terms of how long you can run and the memory your applications can use.

- To run on Niagara's ***compute nodes***, you must submit a batch job.

  In a job script, you can specify how many nodes you need and for how long.

  Once the job scheduler starts your job, it is the only thing running on its reserved nodes.

- For large data transfers, you can use the specialized ***data mover nodes***.

All these nodes see the same shared file system.

Storage Systems and Locations on Niagara and Mist

Home and scratch

You have a home and scratch directory on the shared file systems, whose locations are given by

$HOME=/home/g/groupname/username

$SCRATCH=/scratch/g/groupname/username

Use these convenient variables!

```
nia-login07:~$ pwd
/home/s/scinet/myusername

nia-login07:~$ cd $SCRATCH

nia-login07:myusername$ pwd
/scratch/s/scinet/myusername
```

Storage Systems and Locations on Niagara and Mist

Home and scratch

You have a home and scratch directory on the shared file systems, whose locations are given by

`$HOME=/home/g/groupname/username`

`$SCRATCH=/scratch/g/groupname/username`

Use these convenient variables!

```
nia-login07:~$ pwd
/home/s/scinet/myusername

nia-login07:~$ cd $SCRATCH

nia-login07:myusername$ pwd
/scratch/s/scinet/myusername
```

Project

Users from groups with a RAC allocation will also have a project directory.

`$PROJECT=/project/g/groupname/username`

Storage Systems and Locations on Niagara and Mist

Home and scratch

You have a home and scratch directory on the shared file systems, whose locations are given by

`$HOME=/home/g/groupname/username`

`$SCRATCH=/scratch/g/groupname/username`

Use these convenient variables!

```
nia-login07:~$ pwd
/home/s/scinet/myusername

nia-login07:~$ cd $SCRATCH

nia-login07:myusername$ pwd
/scratch/s/scinet/myusername
```

Project

Users from groups with a RAC allocation will also have a project directory.

`$PROJECT=/project/g/groupname/username`

Burst Buffer

Groups with heavy I/O can request access to a smaller, faster parallel file system called burst buffer.

Storage Limits on Niagara

location	quota	#files	block size	expiration	backed up	on login	compute
$HOME	100 GB	250K	1 MB		yes	yes	read-only
$SCRATCH	25 TB	6M	16 MB	2 months	no	yes	yes
$PROJECT	by group allocation	depends	16 MB		yes	yes	yes
$BBUFFER	10TB, by request		1 MB	48 hours	no	yes	yes
$ARCHIVE	by group allocation				dual-copy	no	no

- Compute nodes do not have local storage, but they have a lot of memory, which you can use as if it is local disk (`$SLURM_TMPDIR`)
- $ARCHIVE space, also called nearline storage or HPSS, is not mounted on login or compute nodes.
- Storage space on project and HPSS is allocated through the annual CC RAC allocation competition.
- Backup means a recent snapshot, not an achive of all data that ever was.

Moving data

Move amounts less than 10GB through the login nodes.

Use `scp` or `rsync` to and from niagara.computecanada.ca.

- For scp to use your ssh key, give it the '`-i ~/.ssh/YOURKEY`' option. *E.g.*

```
laptop> scp -i ~/.ssh/ccf_ed25519 this USERNAME@niagara.computecanada.ca:that
```

- These commands must be given on your computer.
- For rsync to use your ssh key, give it the '`-e "ssh -i ~/.ssh/YOURKEY"`' option.
- This will time out for amounts larger than about 10GB.

Moving data

Move amounts less than 10GB through the login nodes.

Use `scp` or `rsync` to and from niagara.computecanada.ca.

- For scp to use your ssh key, give it the '`-i ~/.ssh/YOURKEY`' option. *E.g.*

```
laptop> scp -i ~/.ssh/ccf_ed25519 this USERNAME@niagara.computecanada.ca:that
```

- These commands must be given on your computer.
- For rsync to use your ssh key, give it the '`-e "ssh -i ~/.ssh/YOURKEY"`' option.
- This will time out for amounts larger than about 10GB.

Move amounts larger than 10GB through the datamover nodes.

- Use scp or rsync with nia-datamover1.scinet.utoronto.ca or nia-datamover2.scinet.utoronto.ca .
- If you do this often, consider using Globus, a web-based tool for data transfer.

Moving data

Move amounts less than 10GB through the login nodes.

Use `scp` or `rsync` to and from niagara.computecanada.ca.

- For scp to use your ssh key, give it the ‘`-i ~/.ssh/YOURKEY`’ option. *E.g.*

```
laptop> scp -i ~/.ssh/ccf_ed25519 this USERNAME@niagara.computecanada.ca:that
```

- These commands must be given on your computer.
- For rsync to use your ssh key, give it the ‘`-e "ssh -i ~/.ssh/YOURKEY"`’ option.
- This will time out for amounts larger than about 10GB.

Move amounts larger than 10GB through the datamover nodes.

- Use scp or rsync with nia-datamover1.scinet.utoronto.ca or nia-datamover2.scinet.utoronto.ca .
- If you do this often, consider using Globus, a web-based tool for data transfer.

Moving data to HPSS/Archive/Nearline.

- HPSS is a tape-based storage solution, and is SciNet’s nearline a.k.a. archive facility.
- Store and recall using scheduled jobs or Globus.

Software and Libraries

Once you are on one of the login nodes, what software is already installed?

- Other than essentials, all installed software is made available using module commands.
- These set environment variables (`PATH`, etc.)
- Allows multiple, conflicting versions of a given package to be available.
- module spider shows the available software.

Software and Libraries

Once you are on one of the login nodes, what software is already installed?

- Other than essentials, all installed software is made available using module commands.
- These set environment variables (PATH, etc.)
- Allows multiple, conflicting versions of a given package to be available.
- module spider shows the available software.

```
nia-login07:~$ module spider
----------------------------------------
The following is a list of the modules..
----------------------------------------
  CCEnv: CCEnv
    Compute Canada software modules. Mus
    modules in 'module spider'.
  NiaEnv: NiaEnv/2018a, NiaEnv/2019b
    Software modules for Niagara. Must b
    'module spider' (loaded by default).
  antlr: antlr/2.7.7
    ANTLR, ANother Tool for Language Rec
    language tool that provides a framew
  ...
```

Software and Libraries, continued

- `module load <module-name>`

  use particular software

- `module purge`

  remove currently loaded modules

- `module spider`

  (or `module spider <module-name>`)

  list available software packages

- `module avail`

  list loadable software packages that require
  no other modules to be loaded first.

- `module list`

  list loaded modules

Software and Libraries, continued

- `module load <module-name>`

  use particular software

- `module purge`

  remove currently loaded modules

- `module spider`

  (or `module spider <module-name>`)

  list available software packages

- `module avail`

  list loadable software packages that require no other modules to be loaded first.

- `module list`

  list loaded modules

On Niagara, there are two distinct software stacks:

Software and Libraries, continued

- `module load <module-name>`

  use particular software

- `module purge`

  remove currently loaded modules

- `module spider`

  (or `module spider <module-name>`)

  list available software packages

- `module avail`

  list loadable software packages that require no other modules to be loaded first.

- `module list`

  list loaded modules

On Niagara, there are two distinct software stacks:

1. A Niagara software stack tuned and compiled for this machine. This stack is available by default, but if not, can be loaded with

```
module load NiaEnv/2019b
```

Software and Libraries, continued

- `module load <module-name>`

  use particular software
- `module purge`

  remove currently loaded modules
- `module spider`

  (or `module spider <module-name>`)

  list available software packages
- `module avail`

  list loadable software packages that require no other modules to be loaded first.
- `module list`

  list loaded modules

On Niagara, there are two distinct software stacks:

1. A Niagara software stack tuned and compiled for this machine. This stack is available by default, but if not, can be loaded with

```
module load NiaEnv/2019b
```

2. The same software stack available on Compute Canada's general purpose clusters. For the Béluga/Narval stack:

```
module load CCEnv StdEnv
```

For the Graham and Cedar stack:

```
module load CCEnv arch/avx2 StdEnv
```

Software and Libraries, continued

- `module load <module-name>`

  use particular software

- `module purge`

  remove currently loaded modules

- `module spider`

  (or `module spider <module-name>`)

  list available software packages

- `module avail`

  list loadable software packages that require no other modules to be loaded first.

- `module list`

  list loaded modules

On Niagara, there are two distinct software stacks:

1. A Niagara software stack tuned and compiled for this machine. This stack is available by default, but if not, can be loaded with

   ```
   module load NiaEnv/2019b
   ```

2. The same software stack available on Compute Canada's general purpose clusters. For the Béluga/Narval stack:

   ```
   module load CCEnv StdEnv
   ```

   For the Graham and Cedar stack:

   ```
   module load CCEnv arch/avx2 StdEnv
   ```

On Mist, there is one, system-specific stack, with modules like cuda, pgi, xl.

Module examples

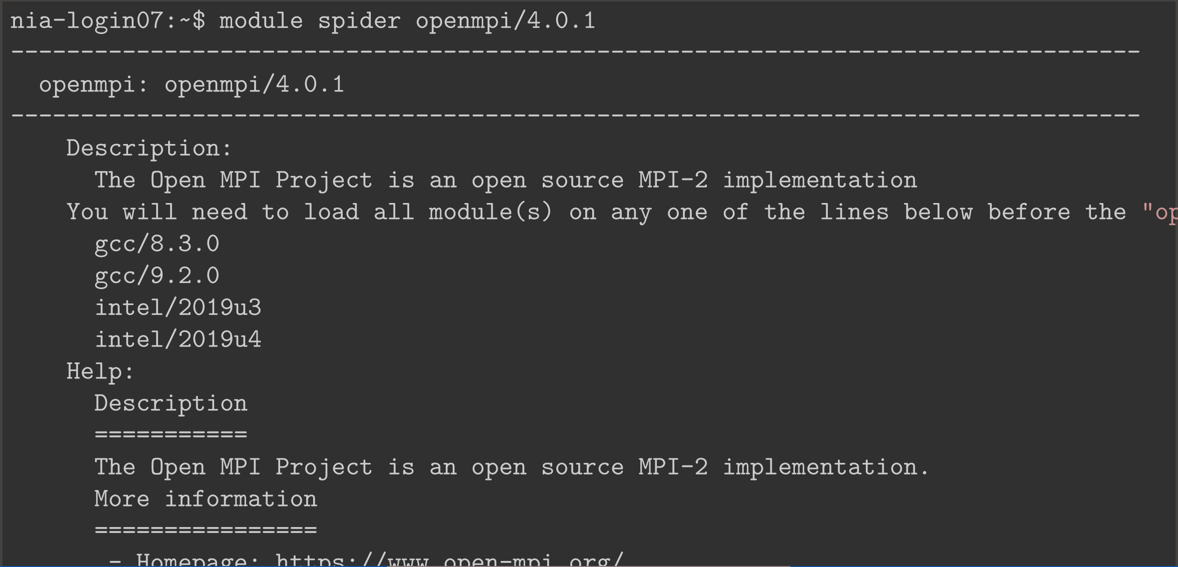
```
nia-login07:~$ module load openmpi
Lmod has detected the following error:  These module(s) or extension(s) exist but
cannot be loaded as requested: "openmpi"
   Try: "module spider openmpi" to see how to load the module(s).
```

Module examples

```
nia-login07:~$ module load openmpi
Lmod has detected the following error:  These module(s) or extension(s) exist but
cannot be loaded as requested: "openmpi"
   Try: "module spider openmpi" to see how to load the module(s).
```

```
nia-login07:~$ module spider openmpi
  openmpi:
---------------------------------------------------------------------------------
    Description:
      The Open MPI Project is an open source MPI-2 implementation
     Versions:
        openmpi/3.1.3
        openmpi/4.0.1
        openmpi/4.0.3
---------------------------------------------------------------------------------
  For detailed information about a specific "openmpi" module use the full name.
  For example:
     $ module spider openmpi/4.0.3
```

Module examples, continued



```
nia-login07:~$ module spider openmpi/4.0.1
---------------------------------------------------------------------------------
  openmpi: openmpi/4.0.1
---------------------------------------------------------------------------------
    Description:
      The Open MPI Project is an open source MPI-2 implementation
    You will need to load all module(s) on any one of the lines below before the "ope
      gcc/8.3.0
      gcc/9.2.0
      intel/2019u3
      intel/2019u4
    Help:
      Description
      ===========
      The Open MPI Project is an open source MPI-2 implementation.
      More information
      ================
       - Homepage: https://www.open-mpi.org/
```

Module examples, continued

```
nia-login07:~$ module load intel/2019u4
nia-login07:~$ module load openmpi/4.0.1
```

Module examples, continued

```
nia-login07:~$ module load intel/2019u4
nia-login07:~$ module load openmpi/4.0.1
```

```
nia-login07:~$ module list
Currently Loaded Modules:
  1) NiaEnv/2019b (S)   2) intel/2019u4   3) openmpi/4.0.1
```

Tips for loading modules

- We advise ***against*** loading modules in your .bashrc file.

  This could lead to very confusing behaviour under certain circumstances.
- Instead, load modules by hand when needed, or by sourcing a separate script.
- Load run-specific modules inside your job submission script.
- Short names give default versions; e.g. `intel` → `intel/2019u4`.

  It is usually better to be explicit about the versions, for future reproducibility.

Can I Run Commercial Software?

- Possibly, but you have to bring your own license for it.
- SciNet and Compute Canada have an extremely large and broad user base of thousands of users, so we cannot provide licenses for everyone's favorite software.
- Thus, the only commercial software installed on Niagara is software that can benefit everyone:

  Intel compilers, math libraries and parallel debuggers.
- That means no MATLAB, Gaussian, IDL, . . .
- Open source alternatives like Octave, Python, R, Julia are available.
- We are happy to help you to install commercial software for which you have a license.
- In some cases, if you have a license, you can use software in the Compute Canada stack.

Python modules

- Several python versions are available as modules.
- These comes with optimized Numpy, SciPy, . . .
- Further packages for Python and R are not installed in modules;
  These need to be installed in users' home directories.
- For installing packages for Python, use virtual environments:

Python modules

- Several python versions are available as modules.
- These comes with optimized Numpy, SciPy, . . .
- Further packages for Python and R are not installed in modules;
  These need to be installed in users' home directories.
- For installing packages for Python, use virtual environments:

```
nia-login07:~$ module load python/3.9.8
nia-login07:~$ virtualenv --system-site-packages ~/myenv
nia-login07:~$ source ~/myenv/bin/activate
(myenv) nia-login07:~$ pip install THISPACKAGE
```

If you want, use the “venv2jup” command to use your virtual environment in the JupyterHub.

If at all possible, do not use conda environments.

R modules

- Several R versions are available as modules, but you first need to load a gcc module

```
$ module load gcc
$ module -r avail ^r/
------------- /scinet/niagara/software/2019b/modules/gcc-8.3.0 -------------
     r/3.5.3    r/3.6.1    r/3.6.3 (D)    r/4.0.3    r/4.1.2
$ module load r/4.1.2
```

- To install R packages, use the R command “install.packages(. . .)”
- The first time you do this, you’ll be asked if you are okay with installing in your home directory (hint: you are).

Compiling on Niagara

- Suppose you have to compile your own C, C++ or Fortran code.
- Not a problem: Niagara has GNU compilers as well as Intel compilers installed in modules.
- Need an MPI library? Not a problem either: Niagara has openmpi and intelmpi libraries as modules.
- We recommend that you use the intel compilers with openmpi libraries.
- Use `-march=native` (gcc) or `-xhost` (intel) compilation flags to get the most out of Niagara's cpus.
- Need libraries? "Module load" them.

Compiling on Niagara

- Suppose you have to compile your own C, C++ or Fortran code.
- Not a problem: Niagara has GNU compilers as well as Intel compilers installed in modules.
- Need an MPI library? Not a problem either: Niagara has openmpi and intelmpi libraries as modules.
- We recommend that you use the intel compilers with openmpi libraries.
- Use `-march=native` (gcc) or `-xhost` (intel) compilation flags to get the most out of Niagara's cpus.
- Need libraries? "Module load" them.

Example

```
nia-login07:~$ module load intel/2019u4 gsl/2.5
nia-login07:~$ ls
main.c module.c
nia-login07:~$ icc -c -O3 -xHost -o main.o main.c
nia-login07:~$ icc -c -O3 -xHost -o module.o module.c
nia-login07:~$ icc  -o main module.o main.o -lgsl -mkl
```

Testing

- Small test jobs can be run on the login nodes.

  Rule of thumb: couple of minutes, taking at most about 1-2GB of memory, couple of cores, ≤ 1 gpu.
- You can run the the ddt debugger after `module load ddt`.
- The ddt module also gives you the map performance profiler.
- Short tests on Niagara that do not fit on a login node, or for which you need a dedicated node, request an interactive debug job with the debugjob command

  ```
  nia-login07:~$ debugjob N
  ```

  where N is the number of nodes. The duration of your interactive debug session can be at most one hour, can use at most N=4 nodes, and each user can only have one such session at a time.
- For short single-gpu tests on Mist use

  ```
  mist-login01:~$ debugjob -g 1
  ```

Submitting jobs

- Niagara and Mist use SLURM as the job scheduler.
- You submit jobs from a login node by passing a script to the sbatch command:

```
nia-login07:~$ sbatch jobscript.sh
```

- This puts the job in the queue. It will run on the compute nodes in due course.
- Jobs will run under their group's RRG allocation, or, if the group has none, under a RAS (or "default") allocation.

Submitting jobs

- Niagara and Mist use SLURM as the job scheduler.
- You submit jobs from a login node by passing a script to the sbatch command:

```
nia-login07:~$ sbatch jobscript.sh
```

- This puts the job in the queue. It will run on the compute nodes in due course.
- Jobs will run under their group's RRG allocation, or, if the group has none, under a RAS (or "default") allocation.

Keep in mind:

Submitting jobs

- Niagara and Mist use SLURM as the job scheduler.
- You submit jobs from a login node by passing a script to the sbatch command:

```
nia-login07:~$ sbatch jobscript.sh
```

- This puts the job in the queue. It will run on the compute nodes in due course.
- Jobs will run under their group's RRG allocation, or, if the group has none, under a RAS (or "default") allocation.

Keep in mind:

- Niagara scheduling is by node, so in multiples of 40-cores. *Use all cores!*
- Mist scheduling is by single gpu or by whole node (multiple of 4 gpus). *Use all GPUs!*
- Maximum walltime is 24 hours.
- Jobs must write to your scratch or project directory (home is read-only on compute nodes).
- Compute nodes have no internet access.

Hyperthreading: Logical CPUs vs. cores

- Hyperthreading is a technology that leverages more of the physical hardware by pretending there are more logical cores than real ones.
- On Niagara, each physical core becomes 2 virtual cores, so nodes seem to have 80 cores.
- On Mist, each physical core becomes 4 virtual cores, so nodes appear to have 128 cores.

Hyperthreading: Logical CPUs vs. cores

- Hyperthreading is a technology that leverages more of the physical hardware by pretending there are more logical cores than real ones.
- On Niagara, each physical core becomes 2 virtual cores, so nodes seem to have 80 cores.
- On Mist, each physical core becomes 4 virtual cores, so nodes appear to have 128 cores.

On Niagara, hyperthreading is actually fairly easy to use:

- Ask for a certain number of nodes N for your jobs.
- You know that you get 40xN cores, so you will get to use a total of 40xN MPI processes or threads. (mpirun, srun, and the OS will automaticallly spread these over the real cores)
- But you should also test if running 80xN MPI processes or threads gives you any speedup.
- Regardless, your usage will be counted as 40xNx(walltime in years).

Example submission script (OpenMP)

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --cpus-per-task=40
#SBATCH --time=1:00:00
#SBATCH --job-name omp_job
#SBATCH --output=omp_output_%j.txt
#SBATCH --mail-type=FAIL

module load NiaEnv/2019b intel/2019u4
OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
export OMP_NUM_THREADS

./omp_example # or 'srun ./omp_example'
```

```
nia-login07:scratch$ sbatch omp_job.sh
```

Example submission script (OpenMP)

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --cpus-per-task=40
#SBATCH --time=1:00:00
#SBATCH --job-name omp_job
#SBATCH --output=omp_output_%j.txt
#SBATCH --mail-type=FAIL

module load NiaEnv/2019b intel/2019u4
OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
export OMP_NUM_THREADS

./omp_example # or 'srun ./omp_example'
```

```
nia-login07:scratch$ sbatch omp_job.sh
```

- First line indicates that this is a bash script.

Example submission script (OpenMP)

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --cpus-per-task=40
#SBATCH --time=1:00:00
#SBATCH --job-name omp_job
#SBATCH --output=omp_output_%j.txt
#SBATCH --mail-type=FAIL

module load NiaEnv/2019b intel/2019u4
OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
export OMP_NUM_THREADS

./omp_example # or 'srun ./omp_example'
```

```
nia-login07:scratch$ sbatch omp_job.sh
```

- First line indicates that this is a bash script.
- Lines starting with `#SBATCH` go to SLURM.

Example submission script (OpenMP)

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --cpus-per-task=40
#SBATCH --time=1:00:00
#SBATCH --job-name omp_job
#SBATCH --output=omp_output_%j.txt
#SBATCH --mail-type=FAIL

module load NiaEnv/2019b intel/2019u4
OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
export OMP_NUM_THREADS

./omp_example # or 'srun ./omp_example'
```

```
nia-login07:scratch$ sbatch omp_job.sh
```

- First line indicates that this is a bash script.
- Lines starting with #SBATCH go to SLURM.
- sbatch reads these lines as a job request (which it gives the name omp_job).

Example submission script (OpenMP)

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --cpus-per-task=40
#SBATCH --time=1:00:00
#SBATCH --job-name omp_job
#SBATCH --output=omp_output_%j.txt
#SBATCH --mail-type=FAIL

module load NiaEnv/2019b intel/2019u4
OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
export OMP_NUM_THREADS

./omp_example # or 'srun ./omp_example'
```

```
nia-login07:scratch$ sbatch omp_job.sh
```

- First line indicates that this is a bash script.
- Lines starting with #SBATCH go to SLURM.
- sbatch reads these lines as a job request (which it gives the name omp_job).
- In this case, SLURM looks for one node with 40 cores to be run inside one task, for 1 hour.

Example submission script (OpenMP)

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --cpus-per-task=40
#SBATCH --time=1:00:00
#SBATCH --job-name omp_job
#SBATCH --output=omp_output_%j.txt
#SBATCH --mail-type=FAIL

module load NiaEnv/2019b intel/2019u4
OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
export OMP_NUM_THREADS

./omp_example # or 'srun ./omp_example'
```

```
nia-login07:scratch$ sbatch omp_job.sh
```

- First line indicates that this is a bash script.
- Lines starting with `#SBATCH` go to SLURM.
- sbatch reads these lines as a job request (which it gives the name `omp_job`).
- In this case, SLURM looks for one node with 40 cores to be run inside one task, for 1 hour.
- Submit from /scratch, as /home is read-only.

Example submission script (OpenMP)

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --cpus-per-task=40
#SBATCH --time=1:00:00
#SBATCH --job-name omp_job
#SBATCH --output=omp_output_%j.txt
#SBATCH --mail-type=FAIL

module load NiaEnv/2019b intel/2019u4
OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
export OMP_NUM_THREADS

./omp_example # or 'srun ./omp_example'
```

```
nia-login07:scratch$ sbatch omp_job.sh
```

- First line indicates that this is a bash script.
- Lines starting with `#SBATCH` go to SLURM.
- sbatch reads these lines as a job request (which it gives the name `omp_job`).
- In this case, SLURM looks for one node with 40 cores to be run inside one task, for 1 hour.
- Submit from /scratch, as /home is read-only.
- Once it found such a node, script is run:
  - Loads modules;
  - Sets an environment variable;
  - Runs the `omp_example` application.

Example submission script (Many serial jobs)

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=40
#SBATCH --time=3:00:00
#SBATCH --job-name serialjob
#SBATCH --output=serial_output_%j.txt
#SBATCH --mail-type=FAIL

module load NiaEnv/2019b
module load gnu-parallel

source ~/myenv/bin/activate
parallel python serial.py ::: {0..99}
```

```
nia-login07:scratch$ sbatch serialjob.sh
```

Example submission script (Many serial jobs)

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=40
#SBATCH --time=3:00:00
#SBATCH --job-name serialjob
#SBATCH --output=serial_output_%j.txt
#SBATCH --mail-type=FAIL

module load NiaEnv/2019b
module load gnu-parallel

source ~/myenv/bin/activate
parallel python serial.py ::: {0..99}
```

```
nia-login07:scratch$ sbatch serialjob.sh
```

- First line indicates that this is a bash script.

Example submission script (Many serial jobs)

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=40
#SBATCH --time=3:00:00
#SBATCH --job-name serialjob
#SBATCH --output=serial_output_%j.txt
#SBATCH --mail-type=FAIL

module load NiaEnv/2019b
module load gnu-parallel

source ~/myenv/bin/activate
parallel python serial.py ::: {0..99}
```

```
nia-login07:scratch$ sbatch serialjob.sh
```

- First line indicates that this is a bash script.
- Lines starting with #SBATCH go to SLURM.

Example submission script (Many serial jobs)

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=40
#SBATCH --time=3:00:00
#SBATCH --job-name serialjob
#SBATCH --output=serial_output_%j.txt
#SBATCH --mail-type=FAIL

module load NiaEnv/2019b
module load gnu-parallel

source ~/myenv/bin/activate
parallel python serial.py ::: {0..99}
```

```
nia-login07:scratch$ sbatch serialjob.sh
```

- First line indicates that this is a bash script.
- Lines starting with `#SBATCH` go to SLURM.
- sbatch reads these lines as a job request (which it gives the name `serialjob`).

Example submission script (Many serial jobs)

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=40
#SBATCH --time=3:00:00
#SBATCH --job-name serialjob
#SBATCH --output=serial_output_%j.txt
#SBATCH --mail-type=FAIL

module load NiaEnv/2019b
module load gnu-parallel

source ~/myenv/bin/activate
parallel python serial.py ::: {0..99}
```

```
nia-login07:scratch$ sbatch serialjob.sh
```

- First line indicates that this is a bash script.
- Lines starting with `#SBATCH` go to SLURM.
- sbatch reads these lines as a job request (which it gives the name `serialjob`).
- In this case, SLURM looks for one node with 40 tasks to be run for 3 hours.

Example submission script (Many serial jobs)

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=40
#SBATCH --time=3:00:00
#SBATCH --job-name serialjob
#SBATCH --output=serial_output_%j.txt
#SBATCH --mail-type=FAIL

module load NiaEnv/2019b
module load gnu-parallel

source ~/myenv/bin/activate
parallel python serial.py ::: {0..99}
```

```
nia-login07:scratch$ sbatch serialjob.sh
```

- First line indicates that this is a bash script.
- Lines starting with `#SBATCH` go to SLURM.
- sbatch reads these lines as a job request (which it gives the name `serialjob`).
- In this case, SLURM looks for one node with 40 tasks to be run for 3 hours.
- Submit from /scratch, as /home is read-only.

Example submission script (Many serial jobs)

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=40
#SBATCH --time=3:00:00
#SBATCH --job-name serialjob
#SBATCH --output=serial_output_%j.txt
#SBATCH --mail-type=FAIL

module load NiaEnv/2019b
module load gnu-parallel

source ~/myenv/bin/activate
parallel python serial.py ::: {0..99}
```

```
nia-login07:scratch$ sbatch serialjob.sh
```

- First line indicates that this is a bash script.
- Lines starting with `#SBATCH` go to SLURM.
- sbatch reads these lines as a job request (which it gives the name `serialjob`).
- In this case, SLURM looks for one node with 40 tasks to be run for 3 hours.
- Submit from /scratch, as /home is read-only.
- Once it found such a node, script is run:
  - Loads modules
  - Activates python environment
  - Uses gnu-parallel to load-balance 99 tasks over the 40 cores on the node.

https://docs.scinet.utoronto.ca/index.php/Running_Serial_Jobs_on_Niagara

Example submission script (MPI)

```
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=40
#SBATCH --time=3:00:00
#SBATCH --job-name mpi_job
#SBATCH --output=mpi_output_%j.txt
#SBATCH --mail-type=FAIL

module load NiaEnv/2019b
module load intel/2019u4
module load openmpi/4.0.1

mpirun ./mpi_app # or 'srun ./mpi_app'
```

```
nia-login07:scratch$ sbatch mpi_job.sh
```

Example submission script (MPI)

```
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=40
#SBATCH --time=3:00:00
#SBATCH --job-name mpi_job
#SBATCH --output=mpi_output_%j.txt
#SBATCH --mail-type=FAIL

module load NiaEnv/2019b
module load intel/2019u4
module load openmpi/4.0.1

mpirun ./mpi_app # or 'srun ./mpi_app'
```

```
nia-login07:scratch$ sbatch mpi_job.sh
```

- First line indicates that this is a bash script.

Example submission script (MPI)

```
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=40
#SBATCH --time=3:00:00
#SBATCH --job-name mpi_job
#SBATCH --output=mpi_output_%j.txt
#SBATCH --mail-type=FAIL

module load NiaEnv/2019b
module load intel/2019u4
module load openmpi/4.0.1

mpirun ./mpi_app # or 'srun ./mpi_app'
```

```
nia-login07:scratch$ sbatch mpi_job.sh
```

- First line indicates that this is a bash script.
- Lines starting with `#SBATCH` go to SLURM.

Example submission script (MPI)

```
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=40
#SBATCH --time=3:00:00
#SBATCH --job-name mpi_job
#SBATCH --output=mpi_output_%j.txt
#SBATCH --mail-type=FAIL

module load NiaEnv/2019b
module load intel/2019u4
module load openmpi/4.0.1

mpirun ./mpi_app # or 'srun ./mpi_app'
```

```
nia-login07:scratch$ sbatch mpi_job.sh
```

- First line indicates that this is a bash script.
- Lines starting with `#SBATCH` go to SLURM.
- sbatch reads these lines as a job request (which it gives the name `mpi_job`)

Example submission script (MPI)

```
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=40
#SBATCH --time=3:00:00
#SBATCH --job-name mpi_job
#SBATCH --output=mpi_output_%j.txt
#SBATCH --mail-type=FAIL

module load NiaEnv/2019b
module load intel/2019u4
module load openmpi/4.0.1

mpirun ./mpi_app # or 'srun ./mpi_app'
```

```
nia-login07:scratch$ sbatch mpi_job.sh
```

- First line indicates that this is a bash script.
- Lines starting with `#SBATCH` go to SLURM.
- sbatch reads these lines as a job request (which it gives the name `mpi_job`)
- In this case, SLURM looks for 2 nodes with 40 cores on which to run 80 tasks, for 3 hours.

Example submission script (MPI)

```
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=40
#SBATCH --time=3:00:00
#SBATCH --job-name mpi_job
#SBATCH --output=mpi_output_%j.txt
#SBATCH --mail-type=FAIL

module load NiaEnv/2019b
module load intel/2019u4
module load openmpi/4.0.1

mpirun ./mpi_app # or 'srun ./mpi_app'
```

```
nia-login07:scratch$ sbatch mpi_job.sh
```

- First line indicates that this is a bash script.
- Lines starting with `#SBATCH` go to SLURM.
- sbatch reads these lines as a job request (which it gives the name `mpi_job`)
- In this case, SLURM looks for 2 nodes with 40 cores on which to run 80 tasks, for 3 hours.
- Submit from /scratch, so output can be written.

Example submission script (MPI)

```
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=40
#SBATCH --time=3:00:00
#SBATCH --job-name mpi_job
#SBATCH --output=mpi_output_%j.txt
#SBATCH --mail-type=FAIL

module load NiaEnv/2019b
module load intel/2019u4
module load openmpi/4.0.1

mpirun ./mpi_app # or 'srun ./mpi_app'
```

```
nia-login07:scratch$ sbatch mpi_job.sh
```

- First line indicates that this is a bash script.
- Lines starting with `#SBATCH` go to SLURM.
- sbatch reads these lines as a job request (which it gives the name `mpi_job`)
- In this case, SLURM looks for 2 nodes with 40 cores on which to run 80 tasks, for 3 hours.
- Submit from /scratch, so output can be written.
- Once it found nodes, the script is run:
  - Loads modules;
  - Runs the `mpi_app` application.

Monitoring jobs - command line

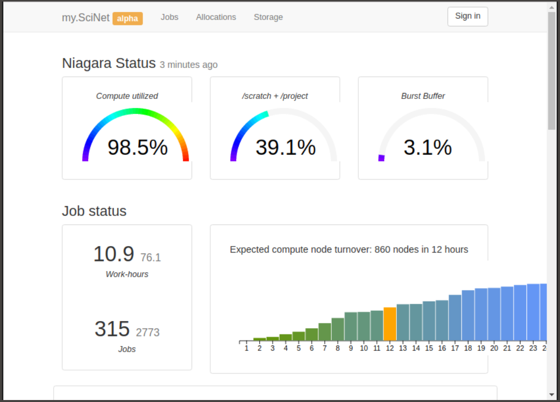
Once the job is incorporated into the queue, there are some commands you can use to monitor its progress:

- `squeue` to show the job queue (`squeue --me` for just your jobs);
- `squeue -j JOBID` to get information on a specific job

  (alternatively, `scontrol show job JOBID`, which is more verbose).
- `squeue --start -j JOBID` to get an estimate for when a job will run.
- `jobperf JOBID` to get an instantaneous view of the cpu+memory usage of a running job's nodes.
- `scancel -i JOBID` to cancel the job.
- `scancel -u USERID` to cancel all your jobs (careful!).
- `sinfo -p compute` to look at available nodes.
- `sacct` to get information on your recent jobs.
- SLURM: https://docs.scinet.utoronto.ca/index.php/Slurm

Monitoring jobs - my.scinet.utoronto.ca

Check out https://my.scinet.utoronto.ca for past and present job info.



Niagara Status 3 minutes ago

Compute utilized 98.5%

/scratch + /project 39.1%

Burst Buffer 3.1%

Job status

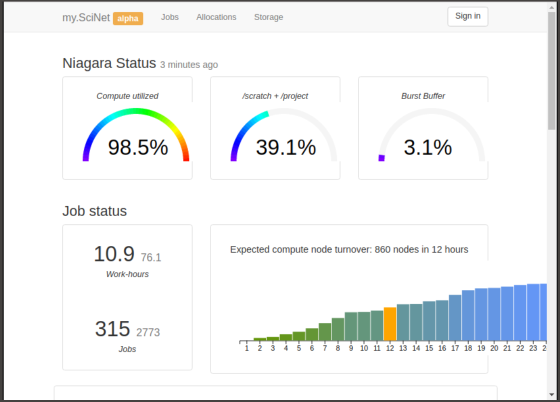
10.9 76.1 Work-hours

315 2773 Jobs

Expected compute node turnover: 860 nodes in 12 hours

Monitoring jobs - my.scinet.utoronto.ca

Check out https://my.scinet.utoronto.ca for past and present job info.



Niagara Status 3 minutes ago

Compute utilized 98.5%

/scratch + /project 39.1%

Burst Buffer 3.1%

Job status

10.9 76.1 Work-hours

315 2773 Jobs

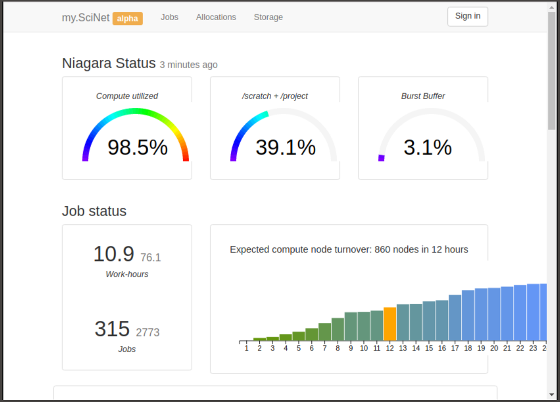
Expected compute node turnover: 860 nodes in 12 hours

Features

- Niagara cpu and storage utilization
- Status of the login nodes
- Niagara and Mist job history

Monitoring jobs - my.scinet.utoronto.ca

Check out https://my.scinet.utoronto.ca for past and present job info.



Niagara Status 3 minutes ago

Compute utilized 98.5%

/scratch + /project 39.1%

Burst Buffer 3.1%

Job status

10.9 76.1 Work-hours

315 2773 Jobs

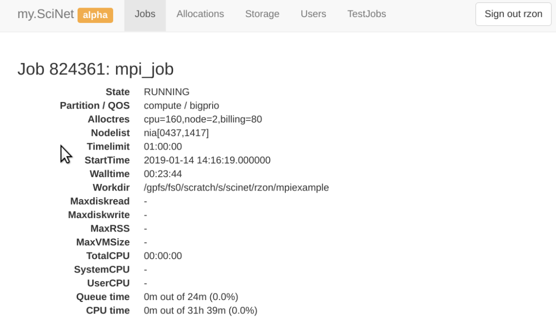
Expected compute node turnover: 860 nodes in 12 hours

Features

- Niagara cpu and storage utilization
- Status of the login nodes
- Niagara and Mist job history
- Per job:
  - jobscript
  - environment
  - wall time
  - memory usage every 10 minutes.
  - cpu usage every 10 minutes.
  - GFlops/s every 10 minutes.
  - disk I/O usage every 10 minutes.

Monitoring jobs online - my.scinet

Check out https://my.scinet.utoronto.ca for past and present job info.



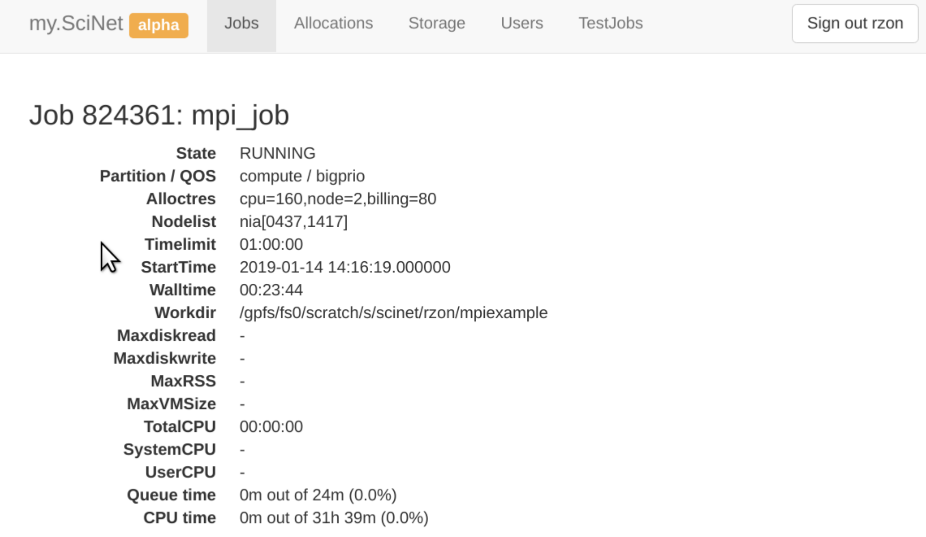
Job 824361: mpi_job

State	RUNNING
Partition / QOS	compute / bigprio
Alloctres	cpu=160,node=2,billing=80
Nodelist	nia[0437,1417]
Timelimit	01:00:00
StartTime	2019-01-14 14:16:19.000000
Walltime	00:23:44
Workdir	/gpfs/fs0/scratch/s/scinet/rzon/mpiexample
Maxdiskread	-
Maxdiskwrite	-
MaxRSS	-
MaxVMSize	-
TotalCPU	00:00:00
SystemCPU	-
UserCPU	-
Queue time	0m out of 24m (0.0%)
CPU time	0m out of 31h 39m (0.0%)

Features

- Niagara cpu and storage utilization
- Status of the login nodes
- Job history
- Per job:
  - jobscript
  - environment
  - wall time
  - memory usage every 10 minutes.
  - cpu usage every 10 minutes.
  - GFlops/s every 10 minutes.
  - disk I/O usage every 10 minutes.

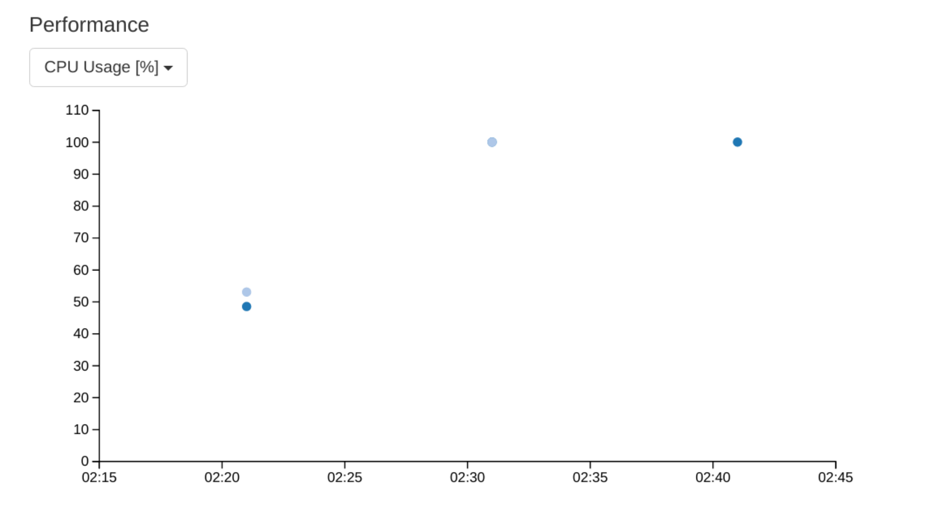
Monitoring jobs online - my.scinet



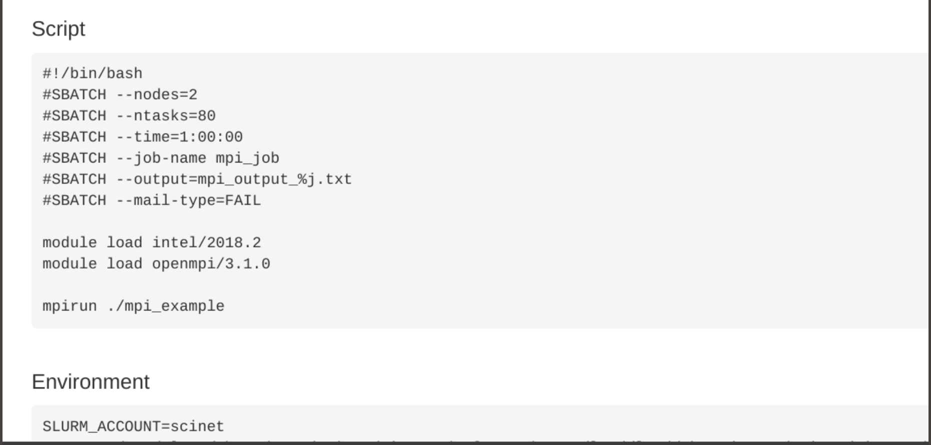
Job 824361: mpi_job

State	RUNNING
Partition / QOS	compute / bigprio
Allocres	cpu=160,node=2,billing=80
Nodelist	nia[0437,1417]
Timelimit	01:00:00
StartTime	2019-01-14 14:16:19.000000
Walltime	00:23:44
Workdir	/gpfs/fs0/scratch/s/scinet/rzon/mpiexample
Maxdiskread	-
Maxdiskwrite	-
MaxRSS	-
MaxVMSize	-
TotalCPU	00:00:00
SystemCPU	-
UserCPU	-
Queue time	0m out of 24m (0.0%)
CPU time	0m out of 31h 39m (0.0%)

Monitoring jobs online - my.scinet



Monitoring jobs online - my.scinet



Script

```
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --ntasks=80
#SBATCH --time=1:00:00
#SBATCH --job-name mpi_job
#SBATCH --output=mpi_output_%j.txt
#SBATCH --mail-type=FAIL

module load intel/2018.2
module load openmpi/3.1.0

mpirun ./mpi_example
```

Environment

```
SLURM_ACCOUNT=scinet
```

Data Management and I/O Tips

- $HOME, $SCRATCH, and $PROJECT all use the parallel file system called GPFS.
- Your files can be seen on all Niagara login and compute nodes.
- GPFS is a high-performance file system which provides rapid reads and writes to large data sets in parallel from many nodes.
- But accessing data sets which consist of many, small files leads to poor performance.
- Avoid reading and writing lots of small amounts of data to disk.
- Many small files on the system would waste space and would be slower to access, read and write.
- Write data out in binary. Faster and takes less space.
- Burst buffer is better for I/O heavy jobs and to speed up checkpoints.

  Either (1) ask `support@scinet.utoronto.ca` for persistent burst buffer space

  or (2) use the temporary `$BB_JOB_DIR`.
- Even better, when it fits is to use `$SLURM_TMPDIR`, which lives in memory.

Further information

Useful sites

- SciNet: https://www.scinet.utoronto.ca
- Niagara: https://docs.computecanada.ca/wiki/Niagara_Quickstart
- Mist: https://docs.scinet.utoronto.ca/index.php/Mist
- Other Compute Canada clusters or general topics: https://docs.computecanada.ca

- System Status: https://docs.scinet.utoronto.ca
- Training: https://education.scinet.utoronto.ca/

Support

Questions? Need help?

Don't be afraid to contact us! We are here to help.

- Email to **support@scinet.utoronto.ca** or to **niagara@computecanada.ca**