Intro to SciNet and Niagara

Mike Nolta (SciNet)

2019 May 10

Outline

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

About SciNet

SciNet is a consortium for High-Performance Computing (HPC) consisting of researchers at U. of T. and its associated hospitals.

- We run massively parallel computers to meet the needs of researchers across Canada.
- There are 5 similar consortia in Canada that provide Advanced Research Computing (ARC) and High Performance Computing resources to Canadian academic researchers and their collaborators.
- These consortia maintain and support a network of Advanced Research Computing resources made available to researchers across Canada, under the allocation system run by Compute Canada (CC):
  - Three heterogenous ("general purpose") clusters
    - ⋆ Cedar (Simon Fraser University)
    - ⋆ Graham (University of Waterloo)
    - ⋆ Béluga (Montréal, Québec)
  - One homogeneous ("large parallel") cluster:
    - ⋆ Niagara (University of Toronto)
  - Several ****cloud** ** systems:

    (Sherbrooke, Victoria, Waterloo)

What does SciNet do?

Supercomputers

We host the largest supercomputers in Canada available to academics.

What does SciNet do?

Supercomputers

We host the largest supercomputers in Canada available to academics.

Niagara

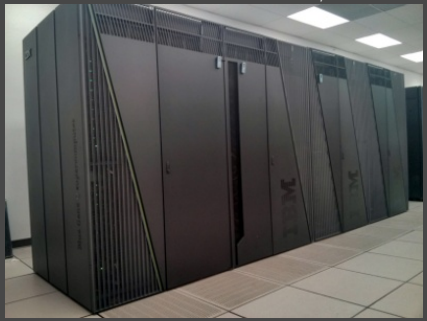
What does SciNet do?

Supercomputers

We host the largest supercomputers in Canada available to academics.

Niagara

BlueGene/Q



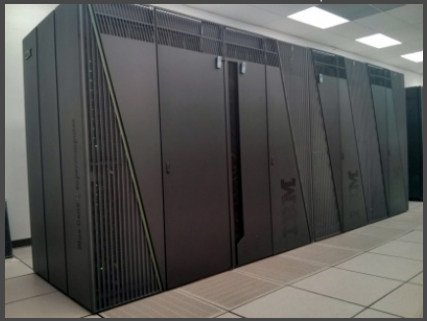
What does SciNet do?

Supercomputers

We host the largest supercomputers in Canada available to academics.

Niagara

BlueGene/Q



Plus a bunch of smaller ones (P7, P8, KNL, SOSCIP GPU).

What does SciNet do?

Supercomputers

We host the largest supercomputers in Canada available to academics.

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://courses.scinet.utoronto.ca

Research

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

SciNet staff is here to help you!

Questions? Need help?

- Read the wiki: https://docs.scinet.utoronto.ca/index.php/Niagara_Quickstart
- Still need help? Email to **support@scinet.utoronto.ca** or **niagara@computecanada.ca**.
- Still need help? Come in for a one-to-one consultation.

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

SciNet people

Software, user support, training, etc.

- Mike Nolta
- Marcelo Ponce
- Erik Spence
- Ramses van Zon
- Bruno Mundim
- Alexey Fedoseev
- Fatih Ertinaz (SOSCIP/IBM)
- Fei Mao (SOSCIP/IBM)

Hardware, systems, etc.

- Scott Northrup
- Joseph Chen
- Ching-Hsing Yu
- Leslie Groer
- Jaime Pinto
- Marco Saldarriaga
- Vladimir Slavnic

- Chief Technical Officer: Daniel Gruner
- Scientific director: Prof. Richard Peltier

- Business manager: Teresa Henriques
- Project coordinator: Alex Dos Santos

Niagara

- 60,000 x86-64 cores.
- 1,500 *Lenovo SD530* nodes
- Per node:
  - 40 Intel SkyLake cores @ 2.4GHz
  - 188 GiB RAM per node ($>$ 4 GiB per core)
  - 0 gpus, 0 harddisks
- 3.02 PFlops delivered / 4.6 PFlops theoretical.

  #59 on the Nov 2018 TOP500
- Operating system: Linux CentOS 7.
- Interconnect: InfiniBand Dragonfly+

  1:1 up to 432 nodes, 2:1 beyond that.
- Parallel share file system for home, scratch, project
- Burst Buffer for fast I/O

Using Niagara: Getting an account

- 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 can take up to 2 business days.

- Go to

  https://ccdb.computecanada.ca/me/facilities

  and click on the "Apply" button next to SciNet.

- After a business day or two (typically), you'll get an email confirming your access to Niagara.

Using Niagara: Logging in

As with all SciNet and Compute Canada systems, access to Niagara is via ssh (secure shell) only.

To access SciNet systems, first open a terminal window (e.g. MobaXTerm on Windows).

Then ssh into the Niagara login nodes with your CC credentials:

```
$ ssh -Y USERNAME@niagara.scinet.utoronto.ca
```

or

```
$ ssh -Y USERNAME@niagara.computecanada.ca
```

- The Niagara 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.
- The optional `-Y` is needed to open windows from the Niagara command-line onto your local X server.
- To run on Niagara's compute nodes, you must submit a batch job.

Storage Systems and Locations on Niagara

Home and scratch

You have a home and scratch directory on the system, whose locations will be given by

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

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

Use these convenient variables!

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

nia-login07:~$ cd $SCRATCH

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

Storage Systems and Locations on Niagara

Home and scratch

You have a home and scratch directory on the system, whose locations will be given by

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

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

Use these convenient variables!

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

nia-login07:~$ cd $SCRATCH

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

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

Home and scratch

You have a home and scratch directory on the system, whose locations will be given by

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

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

Use these convenient variables!

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

nia-login07:~$ cd $SCRATCH

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

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.

`$BBUFFER=/bb/g/groupname/username`

Storage Systems and Locations on Niagara: Purpose

Home individual user files, common software or small datasets used by others in the same group. /home is read-only on the compute nodes, and has backups.

Scratch temporary or transient files, results of your computations and simulations, and any material that can be easily recreated or reacquired. Use scratch for intermediate step in your workflow, provided this does not induce too much I/O or too many small files. /scratch is purged on a regular basis and has no backups.

Project common group software, large static datasets, or any data very costly to be reacquired or re-generated. Material on /project should be relatively immutable over time. Project is backed up. On Niagara, /project is only available to groups with RAC allocation.

BBuffer an fast alternative to /scratch made of solid-state drives. You may request this resource if your jobs perform a lot of disk operations per second, but try Ramdisk first.

Archive a nearline storage pool, to offload semi-active material from any of the above file systems. Material can remain on HPSS for a few months to a few years. Substantial space on /archive requires a RAC allocation.

Ramdisk much faster than real disk and the Burst Buffer. Up to 70 percent of the RAM on the node may be used as a temporary local file system with predictable performance. After a compute job ends, anything stored on ramdisk is erased.

Storage Limits on Niagara

location	quota	block size	expiration time	backed up	on login	on compute
$HOME	100 GB	1 MB		yes	yes	read-only
$SCRATCH	25 TB	16 MB	2 months	no	yes	yes
$PROJECT	by group allocation	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.
- 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.

- Compute node not visible from outside SciNet.
- Use scp or rsync to niagara.scinet.utoronto.ca or niagara.computecanada.ca (no difference).
- This will time out for amounts larger than about 10GB.

Moving data

Move amounts less than 10GB through the login nodes.

- Compute node not visible from outside SciNet.
- Use scp or rsync to niagara.scinet.utoronto.ca or niagara.computecanada.ca (no difference).
- This will time out for amounts larger than about 10GB.

Move amounts larger than 10GB through the datamover nodes.

- Use `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.

- Compute node not visible from outside SciNet.
- Use scp or rsync to niagara.scinet.utoronto.ca or niagara.computecanada.ca (no difference).
- This will time out for amounts larger than about 10GB.

Move amounts larger than 10GB through the datamover nodes.

- Use `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.
- Storage space on HPSS is allocated through the annual CC RAC allocation.
- 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 currently av
---------------------------------------------------
  CCEnv: CCEnv
  NiaEnv: NiaEnv/2018a
  anaconda2: anaconda2/5.1.0
  anaconda3: anaconda3/5.1.0
  autotools: autotools/2017
    autoconf, automake, and libtool
  boost: boost/1.66.0
  cfitsio: cfitsio/3.430
  cmake: cmake/3.10.2 cmake/3.10.3

  ...
```

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
- `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
- `module list`
  list loaded modules

On Niagara, there are really two 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
- `module list`
  list loaded modules

On Niagara, there are really two software stacks:

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

```
module load NiaEnv
```

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
- `module list`
  list loaded modules

On Niagara, there are really two software stacks:

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

   ```
   module load NiaEnv
   ```

2. The same software stack available on Compute Canada's heterogeneous clusters Graham, Cedar and Beluga:

   ```
   module load CCEnv arch/avx512 StdEnv
   ```

   The `StdEnv` module loads the same default modules as available on Graham/Cedar/Béluga.

Tips for loading software

- 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/2018.2`.

  It is usually better to be explicit about the versions, for future reproducibility.
- Modules sometimes require other modules to be loaded first.
  Solve these dependencies by using `module spider`.

Module spider

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

Module spider

```
nia-login07:~$ module load openmpi
Lmod has detected the error:  These module(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:
-----------------------------------------------------------------------------------------------------------
     Versions:
        openmpi/1.10.7
        openmpi/2.1.4
        openmpi/3.0.1
        openmpi/3.0.2
        openmpi/3.1.0
        openmpi/3.1.1

-----------------------------------------------------------------------------------------------------------
  For detailed information about a specific "openmpi" module (including how to load the modules) use
  the module s full name.
  For example:
     $ module spider openmpi/3.0.2
-----------------------------------------------------------------------------------------------------------
```

Module spider, continued

```
nia-login07:~$ module spider openmpi/3.1.0
----------------------------------------------------------------------------------------------------
  openmpi: openmpi/3.1.0
----------------------------------------------------------------------------------------------------
    You will need to load all module(s) on any one of the lines below before the "openmpi/3.1.0"
    module is available to load.

      NiaEnv/2018a  gcc/7.3.0
      NiaEnv/2018a  intel/2018.2
```

Module spider, continued

```
nia-login07:~$ module spider openmpi/3.1.0
----------------------------------------------------------------------------------------------------
  openmpi: openmpi/3.1.0
----------------------------------------------------------------------------------------------------
    You will need to load all module(s) on any one of the lines below before the "openmpi/3.1.0"
    module is available to load.

      NiaEnv/2018a  gcc/7.3.0
      NiaEnv/2018a  intel/2018.2
```

```
nia-login07:~$ module load intel/2018.2
nia-login07:~$ module load openmpi/3.1.0
```

Module spider, continued

```
nia-login07:~$ module spider openmpi/3.1.0
----------------------------------------------------------------------------------------------------
  openmpi: openmpi/3.1.0
----------------------------------------------------------------------------------------------------
    You will need to load all module(s) on any one of the lines below before the "openmpi/3.1.0"
    module is available to load.

      NiaEnv/2018a  gcc/7.3.0
      NiaEnv/2018a  intel/2018.2
```

```
nia-login07:~$ module load intel/2018.2
nia-login07:~$ module load openmpi/3.1.0
```

```
nia-login07:~$ module list
Currently Loaded Modules:
  1) NiaEnv/2018a (S)   2) intel/2018.2   3) openmpi/3.1.0

  Where:
   S:  Module is Sticky, requires --force to unload or purge
```

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 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.

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.
- MPI? Not a problem either: Niagara has openmpi and intelmpi libraries as modules.
- We recommend that you use the intel compilers with openmpi libraries.

Use the `-march=native` or `-xhost` compilation flags to get the most out of Niagara's cpus. . . .

Example

```
nia-login07:~$ module load intel/2018.2 gsl/2.4
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
nia-login07:~$ ./main
```

Testing

You really should test your code before you submit it to the cluster to know if your code is correct and what kind of resources you need.

- 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.

- You can run the the ddt debugger on the login nodes after `module load ddt`.
- The ddt module also gives you the map performance profiler.
- Short tests 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.

Submitting jobs

- Niagara uses SLURM as its 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 uses SLURM as its 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 uses SLURM as its 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:

- Scheduling is by node, so in multiples of 40-cores. *Use all cores!*
- 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.

  → Download data you need beforehand on a login node.

SLURM nomenclature

SLURM has a somewhat different way of referring to things like MPI processes and threads tasks.

Term	Meaning	SLURM	Scheduler option(s)
· **job**	Scheduled piece of work for which specific resources were requested	**job**	`sbatch,salloc,debugjo`
· **node**	Basic computing component with several cores (40) that share memory	**node**	`--nodes -N`
· **MPI process**	One of a group of programs using MPI for parallel computing	**task**	`--ntasks -n` `--ntasks-per-node`
· **physical core**	A fully-functional independent execution unit	**billing**	
· **logical cpu**	An execution unit that the operating system can assign work to	**cpu**	`--ncpus-per-task`
· **thread**	One of multiple simultaneous execution paths in a program that share memory		`--ncpus-per-task` and `OMP_NUM_THREADS`
· **hyperthreading**	Hardware-assisted overloading of cores		

Hyperthreading: Logical CPUs vs. cores

- Hyperthreading, a technology that leverages more of the physical hardware by pretending there are twice as many logical cores than real once, is enabled on Niagara.
- So the OS and scheduler see 80 logical cores.
- 80 logical cores vs. 40 real cores typically gives about a 5-10% speedup (YMMV).

Hyperthreading: Logical CPUs vs. cores

- Hyperthreading, a technology that leverages more of the physical hardware by pretending there are twice as many logical cores than real once, is enabled on Niagara.
- So the OS and scheduler see 80 logical cores.
- 80 logical cores vs. 40 real cores typically gives about a 5-10% speedup (YMMV).

Because Niagara is scheduled by node, 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 use (at least) 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 openmp_job
#SBATCH --output=openmp_output_%j.txt
#SBATCH --mail-type=FAIL

module load intel/2018.2
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

./openmp_example  # or 'srun ./openmp_example'
```

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

Example submission script (OpenMP)

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

module load intel/2018.2
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

./openmp_example  # or 'srun ./openmp_example'
```

```
nia-login07:scratch$ sbatch openmp_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 openmp_job
#SBATCH --output=openmp_output_%j.txt
#SBATCH --mail-type=FAIL

module load intel/2018.2
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

./openmp_example  # or 'srun ./openmp_example'
```

```
nia-login07:scratch$ sbatch openmp_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 openmp_job
#SBATCH --output=openmp_output_%j.txt
#SBATCH --mail-type=FAIL

module load intel/2018.2
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

./openmp_example  # or 'srun ./openmp_example'
```

```
nia-login07:scratch$ sbatch openmp_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 `openmp_job`).

Example submission script (OpenMP)

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

module load intel/2018.2
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

./openmp_example  # or 'srun ./openmp_example'
```

```
nia-login07:scratch$ sbatch openmp_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 `openmp_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 openmp_job
#SBATCH --output=openmp_output_%j.txt
#SBATCH --mail-type=FAIL

module load intel/2018.2
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

./openmp_example  # or 'srun ./openmp_example'
```

```
nia-login07:scratch$ sbatch openmp_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 `openmp_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 openmp_job
#SBATCH --output=openmp_output_%j.txt
#SBATCH --mail-type=FAIL

module load intel/2018.2
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

./openmp_example  # or 'srun ./openmp_example'
```

```
nia-login07:scratch$ sbatch openmp_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 `openmp_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 `openmp_example` application.

Example submission script (MPI)

```
#!/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   # or 'srun ./mpi_example'
```

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

Example submission script (MPI)

```
#!/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   # or 'srun ./mpi_example'
```

```
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=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   # or 'srun ./mpi_example'
```

```
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=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   # or 'srun ./mpi_example'
```

```
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=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   # or 'srun ./mpi_example'
```

```
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 1 hour.

Example submission script (MPI)

```
#!/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   # or 'srun ./mpi_example'
```

```
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 1 hour.
- Submit from /scratch, so output can be written.

Example submission script (MPI)

```
#!/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   # or 'srun ./mpi_example'
```

```
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 1 hour.
- Submit from /scratch, so output can be written.
- Once it found nodes, the script is run:
  - Loads modules;
  - Runs the `mpi_example` application.

Monitoring jobs - command line

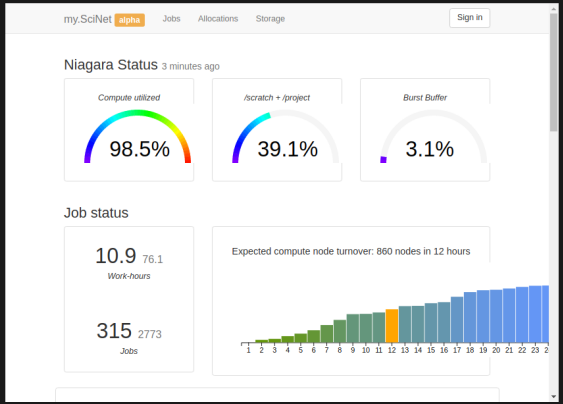
Once the job is incorporated into the queue, there are some command you can use to monitor its progress.

- `squeue` to show the job queue (`squeue -u $USER` 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.

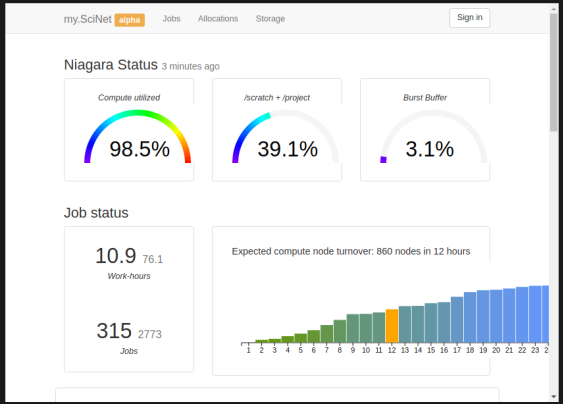
Monitoring jobs - my.scinet.utoronto.ca

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



Monitoring jobs - my.scinet.utoronto.ca

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

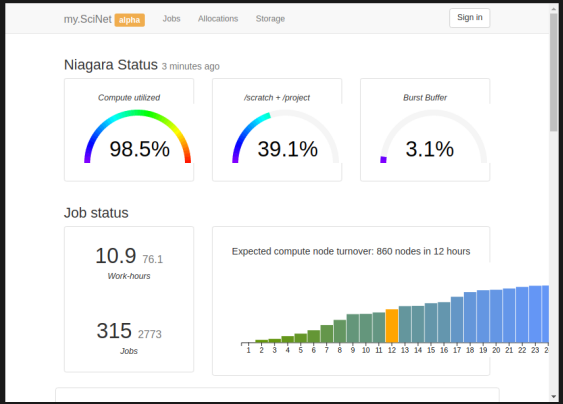


Features

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

Monitoring jobs - my.scinet.utoronto.ca

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

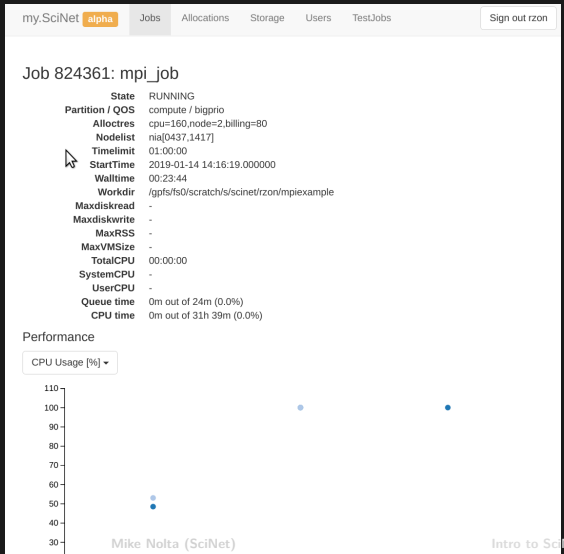


Features

- 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

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



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.

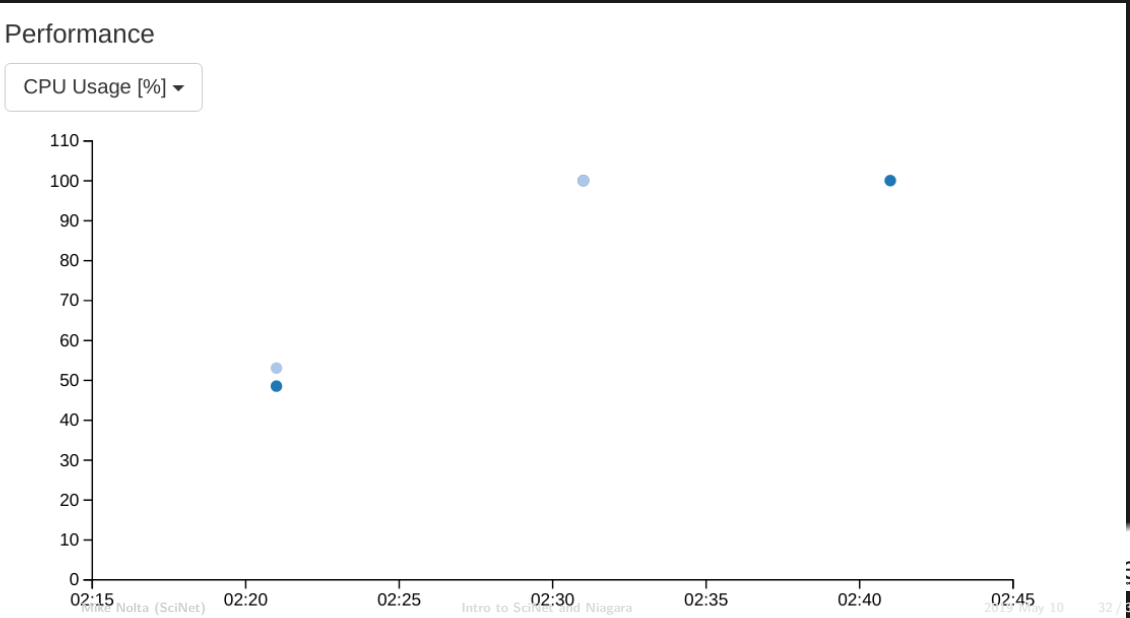
Monitoring jobs online - my.scinet

my.SciNet alpha | Jobs | Allocations | Storage | Users | TestJobs | Sign out rzon

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%)

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.scinet.utoronto.ca/index.php/Niagara_Quickstart
- Other Compute Canada clusters or general topics: https://docs.computecanada.ca
- System Status: https://docs.scinet.utoronto.ca
- Training: https://courses.scinet.utoronto.ca