

# CICA Cluster Updates

5th June 2020

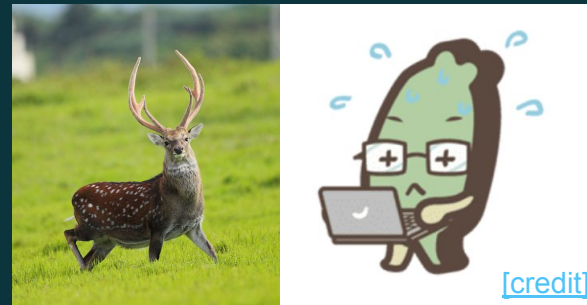


For details and a copy of these slides see our new wiki:

<https://github.com/nthu-ioa/cluster/wiki>

# Outline

- Slurm will replace PBS/Torque
- News / policy updates
- Q&A



# Acknowledging CICA

CICA is funded by multiple grants to the IoA. It's **very** important that we acknowledge this support in all papers that use the cluster **in any way**.

*“This work used high-performance computing facilities at the Center for Informatics and Computation in Astronomy (CICA), operated by the NTHU Institute of Astronomy. This equipment was funded by the **Taiwan Ministry of Education**, the **Taiwan Ministry of Science and Technology** and **National Tsing Hua University**.”*

**Slurm** replaces PBS/Torque

# Changing to Slurm

- Slurm is widely used (about 60% of the Top 500);
- Slurm is actively maintained (unlike Torque/PBS);
- Slurm makes tracking usage a bit easier (helpful for reports) and offers more advanced options for the future.

**The actions you need to take now are:**

- **Learn the Slurm equivalents of PBS commands;**
- **Convert your PBS batch scripts to Slurm.**

# Getting Help: look at the wiki

We have a page about Slurm on the **new** CICA cluster wiki:

<https://github.com/nthu-ioa/cluster/wiki/SLURM>

## SLURM

Andrew Cooper edited this page on Apr 11 · 23 revisions

[Edit](#)[New Page](#)

The batch queue system on our cluster is called Slurm. This manages the resources allocated to jobs running on the compute nodes. Slurm replaces the previous system we used (called Torque/PBS).

### Other examples / tutorials for Slurm

There is lots of useful information about Slurm on the web.

- [https://helpdesk.strw.leidenuniv.nl/wiki/doku.php?id=slurm\\_tutorial](https://helpdesk.strw.leidenuniv.nl/wiki/doku.php?id=slurm_tutorial)

### Basic commands

Inspect the current state of the queues: `squeue`

▼ Pages **22**

[Home](#)[Acknowledgements and Code of Conduct](#)[Cluster Login](#)[Cluster Software](#)[Cluster Specification](#)

# Getting Help: look at the wiki

And a page about migrating from Torque to Slurm:

<https://github.com/nthu-ioa/cluster/wiki/Torque-to-Slurm>

## Torque to Slurm

Andrew Cooper edited this page now · 5 revisions

[Edit](#)[New Page](#)

### Important differences between Slurm and Torque

- What Torque calls "queues" ( `-q` ), Slurm calls "partitions" ( `-p` ); the option `-q` to `srun` does something else.
- `sbatch` does not source shell config files like `~/bashrc` or `~/profile`, but `srun` does.
- By default, the commands in `sbatch` see an environment inherited from the submitting shell. If you want a clean environment, use the option `#SBATCH --export=NONE` in your job script. If you just want a clean set of modules, remember to call `module purge` in your script.
- With Slurm, the working directory when the job starts is the directory of the job script (not the

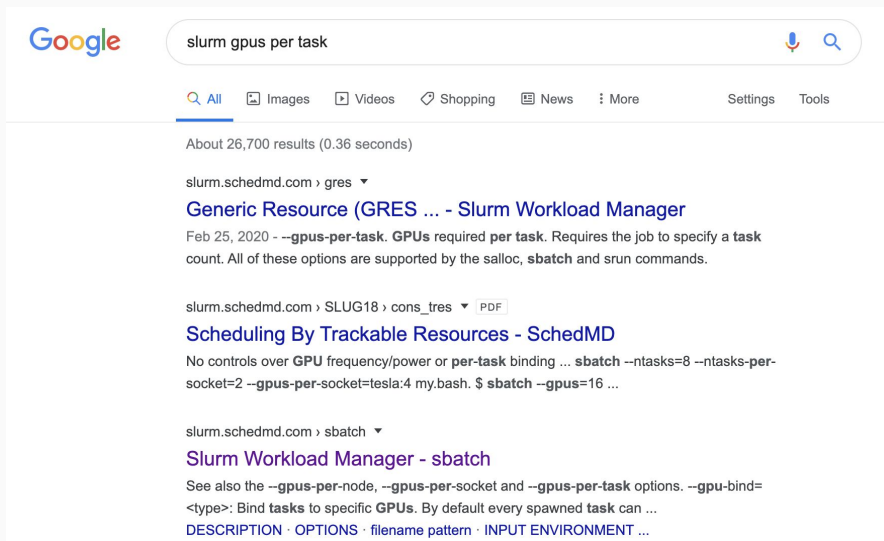
▼ Pages **(22)**

[Home](#)[Acknowledgements and  
Code of Conduct](#)[Cluster Login](#)[Cluster Software](#)[Cluster Specification](#)



# Getting Help: ask the internet

There are **lots** of Slurm tutorials and FAQs on the web. Best resources are usually big HPC centers.



Google search results for "slurm gpus per task". The search bar shows the query and the Google logo. Below the search bar, there are tabs for All, Images, Videos, Shopping, News, and More. The search results show about 26,700 results in 0.36 seconds. The first result is from slurm.schedmd.com, titled "Generic Resource (GRES ... - Slurm Workload Manager". The snippet mentions "Feb 25, 2020 - --gpus-per-task. GPUs required per task. Requires the job to specify a task count. All of these options are supported by the salloc, sbatch and srun commands." The second result is also from slurm.schedmd.com, titled "Scheduling By Trackable Resources - SchedMD". The snippet mentions "No controls over GPU frequency/power or per-task binding ... sbatch --ntasks=8 --ntasks-per-socket=2 --gpus-per-socket=tesla:4 my.bash. \$ sbatch --gpus=16 ...". The third result is from slurm.schedmd.com, titled "Slurm Workload Manager - sbatch". The snippet mentions "See also the --gpus-per-node, --gpus-per-socket and --gpus-per-task options. --gpu-bind=<type>; Bind tasks to specific GPUs. By default every spawned task can ...".

Google search results for "slurm gpus per task".

About 26,700 results (0.36 seconds)

slurm.schedmd.com › gres ▾

**Generic Resource (GRES ... - Slurm Workload Manager**

Feb 25, 2020 - `--gpus-per-task`. GPUs required per task. Requires the job to specify a task count. All of these options are supported by the `salloc`, `sbatch` and `srun` commands.

slurm.schedmd.com › SLUG18 › cons\_tres ▾ PDF

**Scheduling By Trackable Resources - SchedMD**

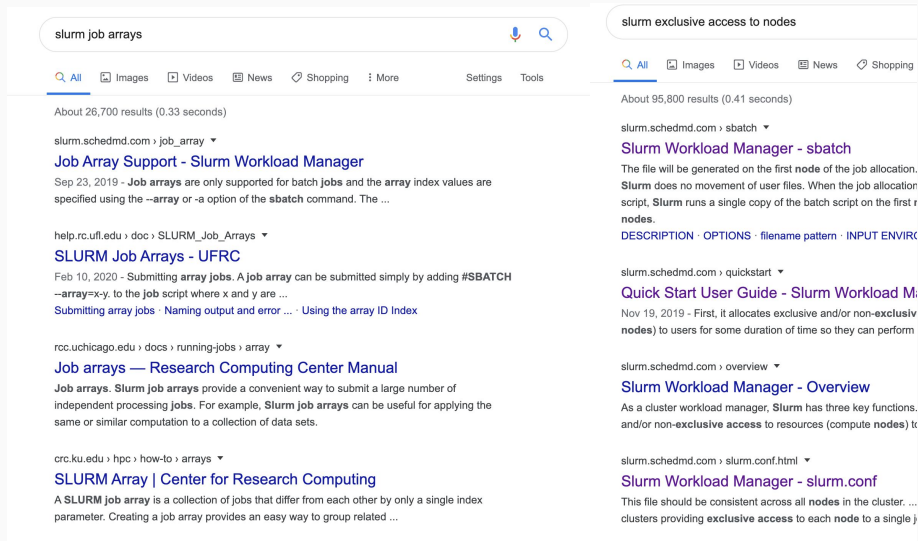
No controls over GPU frequency/power or per-task binding ... `sbatch --ntasks=8 --ntasks-per-socket=2 --gpus-per-socket=tesla:4 my.bash. $ sbatch --gpus=16 ...`

slurm.schedmd.com › sbatch ▾

**Slurm Workload Manager - sbatch**

See also the `--gpus-per-node`, `--gpus-per-socket` and `--gpus-per-task` options. `--gpu-bind=<type>;` Bind tasks to specific GPUs. By default every spawned task can ...

DESCRIPTION · OPTIONS · filename pattern · INPUT ENVIRONMENT ...



Two side-by-side search results. The left panel shows search results for "slurm job arrays". The search bar shows the query and the Google logo. Below the search bar, there are tabs for All, Images, Videos, News, Shopping, and More. The search results show about 26,700 results in 0.33 seconds. The first result is from slurm.schedmd.com, titled "Job Array Support - Slurm Workload Manager". The snippet mentions "Sep 23, 2019 - Job arrays are only supported for batch jobs and the array index values are specified using the --array or -a option of the sbatch command. The ...". The second result is from help.rc.uit.edu, titled "SLURM Job Arrays - UFRC". The snippet mentions "Feb 10, 2020 - Submitting array jobs. A job array can be submitted simply by adding #SBATCH --array=x-y, to the job script where x and y are ...". The third result is from rc.uchicago.edu, titled "Job arrays — Research Computing Center Manual". The snippet mentions "Job arrays. Slurm job arrays provide a convenient way to submit a large number of independent processing jobs. For example, Slurm job arrays can be useful for applying the same or similar computation to a collection of data sets." The right panel shows search results for "slurm exclusive access to nodes". The search bar shows the query and the Google logo. Below the search bar, there are tabs for All, Images, Videos, News, and Shopping. The search results show about 95,800 results in 0.41 seconds. The first result is from slurm.schedmd.com, titled "Quick Start User Guide - Slurm Workload Manager". The snippet mentions "Nov 19, 2019 - First, it allocates exclusive and/or non-exclusive nodes to users for some duration of time so they can perform ...". The second result is from slurm.schedmd.com, titled "Slurm Workload Manager - Overview". The snippet mentions "As a cluster workload manager, Slurm has three key functions, and/or non-exclusive access to resources (compute nodes) to ...". The third result is from slurm.schedmd.com, titled "Slurm Workload Manager - slurm.conf". The snippet mentions "This file should be consistent across all nodes in the cluster. ... clusters providing exclusive access to each node to a single j ...".

slurm job arrays

About 26,700 results (0.33 seconds)

slurm.schedmd.com › job\_array ▾

**Job Array Support - Slurm Workload Manager**

Sep 23, 2019 - Job arrays are only supported for batch jobs and the array index values are specified using the `--array` or `-a` option of the `sbatch` command. The ...

help.rc.uit.edu › doc › SLURM\_Job\_Arrays ▾

**SLURM Job Arrays - UFRC**

Feb 10, 2020 - Submitting array jobs. A job array can be submitted simply by adding `#SBATCH --array=x-y`, to the job script where `x` and `y` are ...

Submitting array jobs · Naming output and error ... · Using the array ID Index

rc.uchicago.edu › docs › running-jobs › array ▾

**Job arrays — Research Computing Center Manual**

Job arrays. Slurm job arrays provide a convenient way to submit a large number of independent processing jobs. For example, Slurm job arrays can be useful for applying the same or similar computation to a collection of data sets.

crc.ku.edu › hpc › how-to › arrays ▾

**SLURM Array | Center for Research Computing**

A SLURM job array is a collection of jobs that differ from each other by only a single index parameter. Creating a job array provides an easy way to group related ...

slurm exclusive access to nodes

About 95,800 results (0.41 seconds)

slurm.schedmd.com › sbatch ▾

**Slurm Workload Manager - sbatch**

The file will be generated on the first node of the job allocation. Slurm does no movement of user files. When the job allocation script, Slurm runs a single copy of the batch script on the first r nodes.

DESCRIPTION · OPTIONS · filename pattern · INPUT ENVIRONMENT ...

slurm.schedmd.com › quickstart ▾

**Quick Start User Guide - Slurm Workload Manager**

Nov 19, 2019 - First, it allocates exclusive and/or non-exclusive nodes to users for some duration of time so they can perform ...

slurm.schedmd.com › overview ▾

**Slurm Workload Manager - Overview**

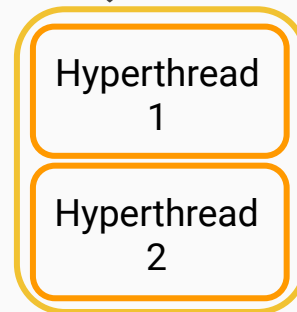
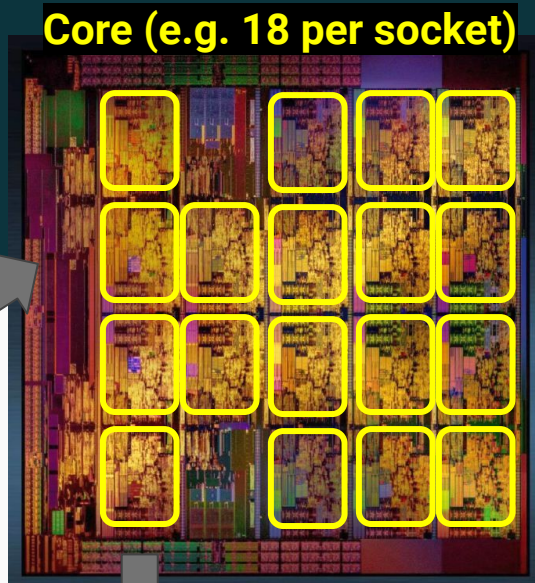
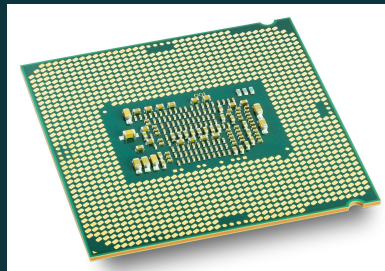
As a cluster workload manager, Slurm has three key functions, and/or non-exclusive access to resources (compute nodes) to ...

slurm.schedmd.com › slurm.conf.html ▾

**Slurm Workload Manager - slurm.conf**

This file should be consistent across all nodes in the cluster. ... clusters providing exclusive access to each node to a single j ...

# Slurm jargon



**2 Slurm  
CPUs  
per core**

**Node (e.g. c01)**

**Socket**

**Socket**

**Partition**

**cpu  
gpu  
mem**



# Slurm jargon

**Node:** A machine on the cluster network, e.g. c01.

**Socket:** A slot on a motherboard of a node that holds a package of cores (i.e. "a processor chip").

**Core:** A **physical** region on a processor chip that executes instructions in your program. Modern processors have many cores. A processor with 18 cores can execute 18 processes in parallel. In the simplest setup, one core is equivalent to one Slurm "CPU".

**CPU:** The basic unit that **Slurm** sees as able to execute a program. A CPU can only do one thing at a time. We use '**hyperthreading**' on the cluster, which means that **1 core looks like 2 CPUs to Slurm**.

# Slurm jargon

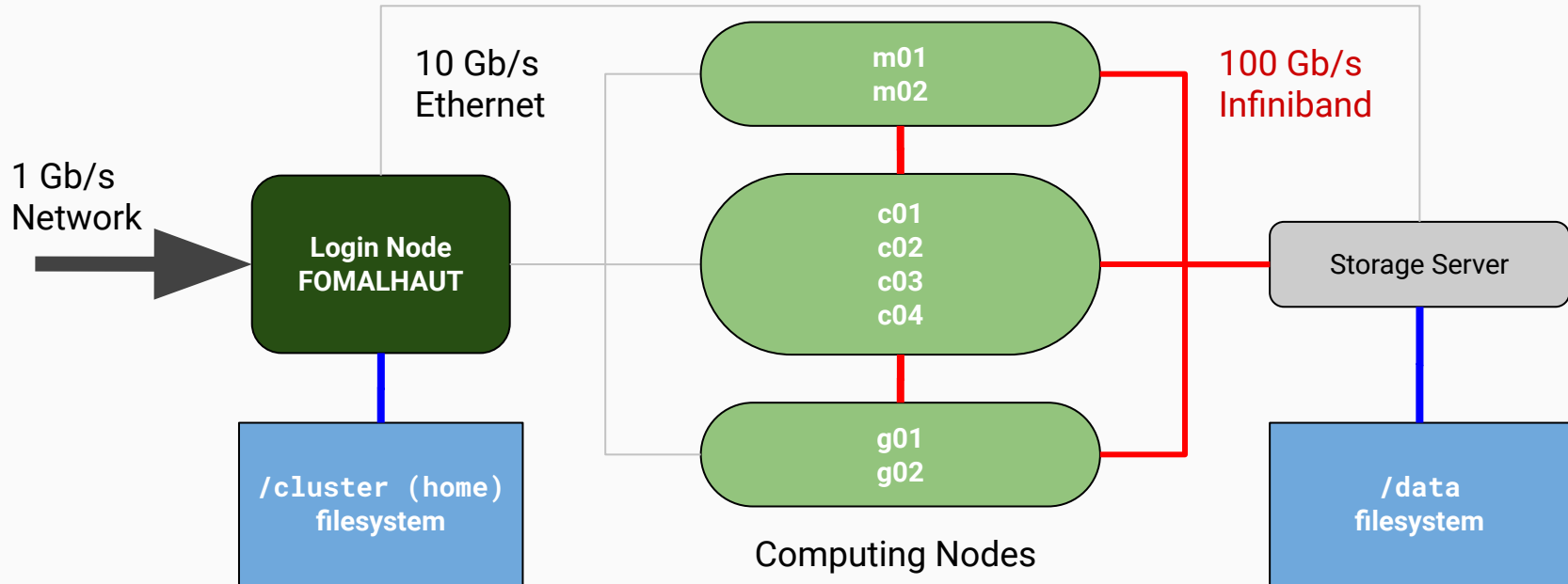
**A Slurm partition:** A logical group of nodes with common resources, connected by the network. Other systems call this a *queue*. We define 3 partitions in our cluster: **cpu**, **gpu** and **mem**.

All our nodes have 2 sockets. The nodes in the cpu and gpu partitions have 18 cores per socket and 2 CPUs (a.k.a. threads) per core, hence can support  $2 \times 18 \times 2 = 72$  processes running at the same time (mem nodes have 20 cores per socket).

**On a single node**, using *shared memory*, processes can easily communicate with each other (regardless of which core/CPU they are executing on).

**Communication *between nodes*** requires *Message Passing* over the network.

# CICA Cluster Overview (Phase 1)



# Basic Slurm Commands

*What are the partitions and what state are they in?*

**sinfo**

*What jobs are in the queue now?*

**squeue**

*Submit a job script*

**sbatch**

*Stop a running job*

**scancel**

*Check on the state of a running job*

**sstat** (and **sacct**)

*Check on a completed job*

**seff** (and **sacct**)

**Tip:** read the man pages and online Slurm documentation for these commands. They all have many useful options.

# sinfo: what resources are available?

```
fomalhaut ~ sinfo
```

PARTITION	AVAIL	TIMELIMIT	NODES	STATE	NODELIST
cpu	up	infinite	4	idle	c[01-04]
mem	up	infinite	2	idle	m[01-02]
gpu	up	infinite	3	idle	g[01-03]
all*	up	infinite	9	idle	c[01-04],g[01-03],m[01-02]

# sinfo: what resources are available?

```
fomalhaut ~ sinfo -lN
```

```
Tue Mar 24 14:08:51 2020
```

NODELIST	NODES	PARTITION	STATE	CPUS	S:C:T	MEMORY	TMP_DISK	WEIGHT	AVAIL_FE	REASON
c01	1	cpu	idle	72	2:18:2	386600	0	1	(null)	none
c01	1	all*	idle	72	2:18:2	386600	0	1	(null)	none
c02	1	cpu	idle	72	2:18:2	386600	0	1	(null)	none
c02	1	all*	idle	72	2:18:2	386600	0	1	(null)	none
c03	1	cpu	idle	72	2:18:2	386600	0	1	(null)	none
c03	1	all*	idle	72	2:18:2	386600	0	1	(null)	none



# queue: detailed look at jobs in queue

```
fomalhaut ~ queue
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
7868	mem	jupyter	apcooper	R	2-03:21:55	1	m01
7915	cpu	runbinar	ivywang2	R	1:16:44	1	c01
7916	cpu	runbinar	ivywang2	R	1:15:47	1	c01
7917	cpu	runbinar	ivywang2	R	1:14:48	1	c01
7918	cpu	runbinar	ivywang2	R	1:13:15	1	c01
7919	cpu	runbinar	ivywang2	R	1:12:24	1	c01
7920	cpu	runbinar	ivywang2	R	33:39	1	c01
7923	mem	jupyter	lwliao	R	28:26	1	m01

```
fomalhaut ~
```

# sbatch: submit a job script

```
> sbatch my_job_script.sh  
Submitted batch job 7819  
  
> squeue  
  
> sbatch --help  
> man sbatch
```

# scancel: stop running jobs

```
> scancel 7819  
> queue
```

# Job scripts

```
#!/bin/bash
#SBATCH --partition cpu
#SBATCH --cpus-per-task=8
#SBATCH --time=01:00:00
#SBATCH --mem=32G

module load python
source activate mpyy

srun python do_something.py
```

*This is a shell script.*

**Resource requests** and other selections go here, as lines that start with #SBATCH.

Define the environment, using regular shell commands. **This will be exported to the nodes that execute your job.**

The command line for your job. The "srun" helps Slurm to track your job. Use **mpirun** here instead for MPI jobs.

# Translating your scripts

See our wiki for tips on migrating your scripts from PBS Torque to Slurm:

<https://github.com/nthu-ioa/cluster/wiki/Torque-to-Slurm>

Some common issues:

- Environment variables, modules and shell configuration files;
- Confusing ntasks-per-node with cpus-per-task;
- Log files ending up in the wrong place or jobs not starting because logs can't be written.

# Check your job

Log on to the node where your job is running and use **top** or **htop**. Use **free -h** to check memory usage.

Use **sstat** and **sacct** to check the job while it's running. Read the manpages and learn about their formatting options. MaxRSS is the number to look at for memory usage.

Use **seff** once the job has finished, to check how efficiently you used the resources you asked for. How does the CPU time compare to the wall time? You can also use **sacct --start 1/1/2020** to get a list of your completed/failed jobs.

For the above, it helps to get your jobid from **squeue** first. For now, these monitoring commands only work on fomalhaut.

# Interactive Jobs

```
> srun --time=1:00:00 -p mem --mem=20G --pty bash  
[apcooper@m01 ~]$
```

```
> srun --nodes=1 --ntasks-per-node=1 --cpus-per-task=8  
--time=1:00:00 -p mem --mem=20G --pty bash
```

# Common use cases

- **Single-process jobs**, run as one-offs or in sequence;
- **Embarrassingly parallel arrays of single-process jobs**, run at the same time and not talking to each other;
- **Shared Memory** jobs that run a **single process executing multiple parallel threads** on different cores on a **single machine** (hence talking to each other through **shared memory**);
- **MPI**: Jobs that run a **single process executing multiple parallel threads** on different cores on **different machines**, talking to each other through **message-passing over the network**.



# sbatch/srun options: nodes/tasks/cpus

*--nodes, -N : number of nodes (only really important for MPI jobs)*

*--ntasks, -n : number of tasks (instances of your program)*

*--ntasks-per-node : (maximum) number of tasks to start on each node*

*--cpus-per-task, -c : number of processors to allocate for each task*

Note that **srun --ntasks=8** will start **8 instances** of whatever command follows, not 1 instance that has access to 8 CPUs. Unless you are running an MPI program, **you usually want 1 instance with access to multiple cpus**. **That is what --cpus-per-task is for.**

# Common use cases

- **Single-process serial jobs:** `--nodes=1` (the default; everything else is implied).
- **Single-process multi-threaded serial jobs:** `--nodes=1 --cpus-per-task=8`
- **Shared Memory** message-passing parallel jobs: `--nodes=1 --ntasks-per-node=3`
- **Shared Memory** parallel jobs with threads: `--nodes=1 --ntasks-per-node=3 --cpus_per_task=3`
- **MPI:** `--nodes=2 --ntasks-per-node=3`

Note that most users will have `--nodes=1` most of the time. Note that the use of nodes is not exclusive unless you add `--exclusive`, which you should not do without good reason).

# Slurm Details and Tips

# MPI Jobs

For the time being, **ignore advice elsewhere to use `srun` rather than `mpirun`** for the command line that starts your code. This might change in future.

In your script, use `mpirun` as you would do with Torque.

If you are using MPI, try to make a request that fills nodes (or less than one node, although that does not make a lot of sense for an MPI job). For maximum efficiency, use `--exclusive`, with caution.

Our Slurm+MPI+Infiniband setup still needs more testing. Please let us know if you have any suggestions or difficulties, or find something unusual.

# Jupyter Lab

```
#!/bin/bash
#SBATCH --partition mem
#SBATCH --odelist m01
#SBATCH --nodes 1
#SBATCH --ntasks-per-node 1
#SBATCH --cpus-per-task 1
#SBATCH --mem 32G
#SBATCH --job-name jupyter
#SBATCH --output /data/apcooper/.jupyterlog/jupyter-%J.log
#SBATCH --time 3-0
```

```
port_id = 8890
echo "Running Jupyter on port ${port_id}"
```

```
module load python
source activate mypy
```

```
cd /
srun jupyter lab --no-browser --port ${port_id}
```

On your client machine, connect to the server with an SSH tunnel, using the appropriate port.

```
#!/bin/bash
echo Starting SSH pass-through to jupyter on port $1
ssh fomalhaut -L $1:localhost:$1 ssh m01 -L $1:localhost:$1
```

See:

<https://github.com/nthu-ioa/cluster/wiki/Jupyter>

# Job Arrays

**Example:** I have a simple code that takes 3 parameters (A,B,C) and outputs a single-line result.

I want to run this for all combinations of 10 values of A, 5 values of B and 3 values of C, i.e.  $10 \times 5 \times 3 = 150$  runs of the same code, with different parameters each time..

Depending on the parameters, one run takes between 10 minutes and 2 hours. Each run is independent of every other run.

This is an **embarrassingly parallel** task (but there is nothing embarrassing about it!).

The **wrong way** to do this on the cluster is with a loop over parameters inside a wrapper script spawning many threads/processes. Slurm can't help to schedule that efficiently.

# Job Arrays

The simplest efficient solution is usually an **array job** with a **slot limit**. See wiki for details:

<https://github.com/nthu-ioa/cluster/wiki/SLURM#array-jobs>

```
> sbatch --array 0-150%10 my_slurm_script.sh
```

Slurm will schedule all 150 jobs separately in the queue, but grouped together in a way that makes them easier to manage

**"%10" is the slot limit.** This means that only 10 jobs in the array will be allowed to run at once. This lets you limit the overall number of cores that you use. When an element is finished, Slurm will start a new one from the array.

You can do more fancy things with the range (e.g. lists, step sizes).

# Array jobs

```
(mypy) fomalhaut /data/apcooper/coco/data/cdm/lacey15/trees_rvmax sq
```

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
3301_[10-64%16]	mem	makeinpu	apcooper	PD	0:00	1	(Resources)
3301_0	mem	makeinpu	apcooper	R	0:07	1	m01
3301_1	mem	makeinpu	apcooper	R	0:07	1	m01
3301_2	mem	makeinpu	apcooper	R	0:07	1	m01
3301_3	mem	makeinpu	apcooper	R	0:07	1	m01
3301_4	mem	makeinpu	apcooper	R	0:07	1	m01
3301_5	mem	makeinpu	apcooper	R	0:07	1	m02
3301_6	mem	makeinpu	apcooper	R	0:07	1	m02
3301_7	mem	makeinpu	apcooper	R	0:07	1	m02
3301_8	mem	makeinpu	apcooper	R	0:07	1	m02
3301_9	mem	makeinpu	apcooper	R	0:07	1	m02



# Elements have unique **SLURM\_ARRAY\_ID**

Inside your script, the `#SBATCH` requests should be for a **single job element** (e.g. if the elements are independent, single-threaded jobs, you should ask for `--cpus_per_task=1`). This includes **memory** and **runtime**.

Slurm sets an environment variable **SLURM\_ARRAY\_ID** to a unique array index for each element. **You can pass this to your own code** to control e.g. which set of parameters it uses. For example:

```
srun my_simple_code.py --index=${SLURM_ARRAY_ID}
```

If you find this limiting, or want to run 100s of simultaneous jobs some other way, please talk to us first.

# Guidelines

**Everyone is sharing the same limited resources as you: time, CPUs and memory.**

If you want to take lots of resources (memory, CPUs, GPUs) for a long time, please make the effort to understand **how your program will use those resources** and think about the **best parallel strategy**.

The more resources you want to use, the bigger the benefit from thinking before submitting.

**We are happy to help.** Please try to avoid overclaiming resources "to be on the safe side", because we can't support that for everyone.

# Things to think about

How many threads does my job run in parallel? Can it really use all the cores I'm requesting? Does it actually use those cores? What is the maximum memory my job needs?

**Do smaller test runs to make these judgements.** Think about the most efficient strategy to distribute your jobs. Profile a single instance of your code. Have your code write logs of its progress (when will it finish?).

If the machine is busy, writing **sensible limits** in your sbatch script will get your job into the queue faster. Short runtime limits are helpful.

# Things to avoid

- Job arrays with 1000s of entries, each of which executes for only a few seconds. Consider combining these into larger batches inside each element. Investigate *gnu parallel*.
- Jobs which use many cores or large amounts of memory for only a tiny fraction of the runtime, and spend the rest of the time on a single core. Sometimes hard to avoid, but try.
- Jobs that spawn new processes outside the control of Slurm. Do not do this,. You almost certainly don't need to. Use threads or job steps.
- Jobs which submit other jobs, unless you really know what you are doing (read up on Slurm job steps and dependencies).

# Restart files

If a job is going to run for a very long time, it's worth making it robust to unexpected failures of the machine.

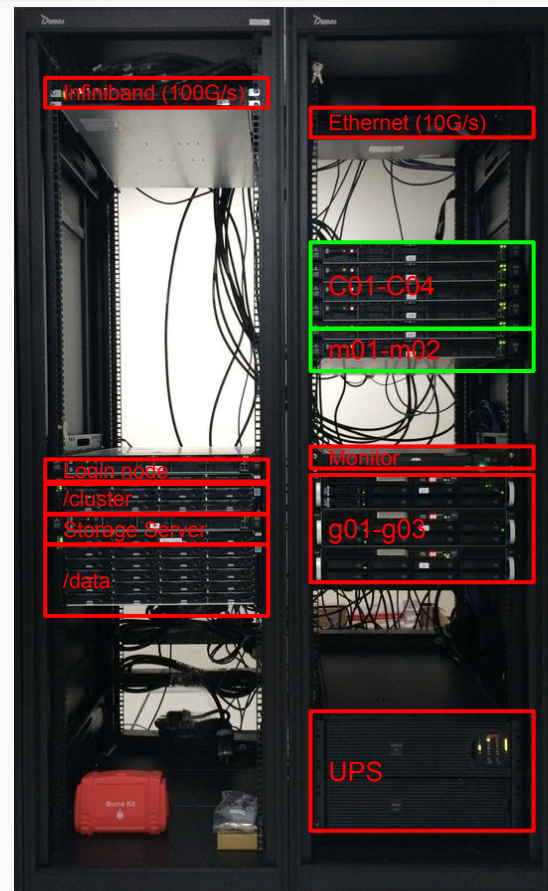
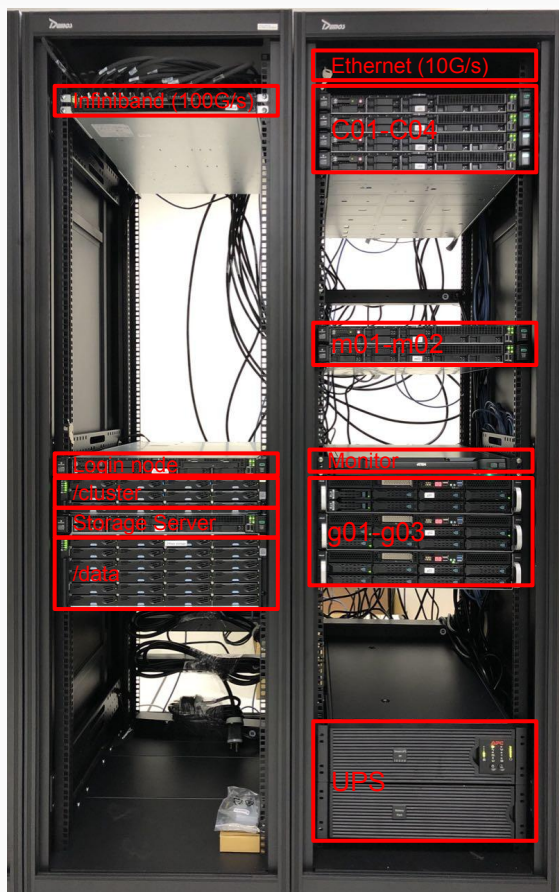
Robust jobs should be able to restart from part-way through their calculation, by occasionally saving their current state into 'restart' files. Delete these when your job has finished.

Advanced: SIGTERM should be sent before jobs are killed, but don't count on that.

Also: cache expensive calculations to disk. Make sure processes don't fight for the same cache files.

News

# News



# HDF5 module is the same as other modules

**`module load gcc hdf5`**

is enough for serial HDF5, no need for `module load flavor/hdf5/serial` or similar.

**The default is `v1.10.5`. Other options: `v1.8` or `v1.10.5` with `v1.8` API**

*You may also notice that having python HDF5 packages installed via conda gives you access to the HDF5 command line tools. This is handy, but compile code against the cluster module libraries.*



# Code of conduct

<https://github.com/nthu-ioa/cluster/wiki/Acknowledgements-and-Code-of-Conduct>

This is a 'contract' for using the cluster. Most of it should be obvious, but better to have it written down.

**Summary: be nice, only use the cluster for work, do not share/expose any details of your account.**

This will evolve. Suggestions are welcome. If you see problems, please let us know as soon as possible, in confidence.

# Job awareness

Jobs become '**noticeable**' when we see them in the queue for several days, when we notice high network or I/O traffic, when we notice very low CPU usage on long-running jobs, when other users send us emails saying they can't get their work done because of your job, etc. etc.

On rare occasions, we (the administrators) might ask you about your jobs.

We expect that, if we ask, you can tell us what your job is doing, how it works, why you need the resources you have requested, and when you expect your job to finish. If you can't explain/justify these things, we might have to remove your job from the queue, especially if other jobs are waiting.

You have a right to privacy on the cluster, especially from other users, but system administrators reserve the right to inspect e.g. source code, I/O files, network traffic and submission scripts of problematic jobs.

# Using the cluster for academic work

With the agreement of their supervisor, **IoA students** are welcome to do their **NTHU** academic work on the cluster, so long as the load is not 'noticable' (and the same rules apply as for everyone else).

**Research jobs and system maintenance always have priority.**

"The CICA cluster went down before my job finished" or "the administrators killed my job" are not a valid excuse for late homework / failing your course / not graduating (unless your instructor specifically told you to use the cluster and/or you agreed heavy work with us beforehand).

Talk to your supervisor. It is better to seek permission than ask forgiveness. **Always acknowledge the use of the cluster.**

# I/O From Batch Jobs

**Please avoid any high-throughput I/O from compute jobs to the `/home` space.**

Reason: `/home` is directly attached to the login node. Heavy I/O on this node eats up limited network bandwidth and CPU time.

We have a dedicated storage node and fast `/data` space specifically for job I/O. Please try to use it!

Please try to avoid writing 1000s of tiny files. Please try to avoid reading and writing through long chains of symlinks. Avoid intensive work over symlinks between `/home` and `/data`.

# Compiling on Fomalhaut vs. nodes

All nodes have the same skylake-avx512 architecture. Compiling on fomalhaut is generally fine.

**Power users** should watch out for differences in the GCC "-march=native" defaults between fomalhaut and the compute nodes. **Always module load gcc before you compile!**

**Fomalhaut:** l2-cache-size=11264, -mno-avx512vnni`

**c/g nodes:** l2-cache-size=25344, -mno-avx512vnni

**m nodes:** l2-cache-size=28160, -mavx512vnni

Thanks for your attention!

Q&A