Slurm basics

Summer Kickstart 2017

June 2017

Aalto University
School of Science

Triton layers

Triton is a powerful but complex machine. You have to consider:

- **Connecting (ssh)**
- Data storage (filesystems and Lustre)
- **Resource allocation (Slurm)**
- Code (yours)
- Other applications (modules)

This talk gets you started with the basics of connecting and running your code

Connecting to Triton

Accessing Triton

```
$ ssh AALTO_LOGIN@triton.aalto.fi
```

- Requires valid Aalto account. Contact local Triton support member and ask for granting access.
- Directly reachable from:
  - department workstations
  - wired visitor networks, wireless Aalto, Aalto Open and Eduroam at Aalto
  - CSC servers
- Outside of Aalto:
  - Must hop through Aalto shell servers; first ssh to taltta.aalto.fi or kosh.aalto.fi

Best practice: SSH key

On your workstation where from you want to login to Triton:

```
$ ssh-keygen
$ ssh-copy-id triton.aalto.fi
$ ssh triton.aalto.fi
```

for a sake of security / convenience

SSH key must have a secure passphrase!

More info: Triton wiki → Accessing triton.aalto.fi

Frontend node: intended usage

- When you first connect, you get the login node (login2)
- Just one of the computers out of others adapted for server needs

YES:

- File editing
- File transfers
- Code compilation
- Job control
- Debugging
- Checking results

NO:

- No multi-CPU loads
- No multi-GB datasets into memory
- But general Matlab, R, IPython sessions otherwise OK

Jobs must go to the queue!

Use case: transferring files

- Network share (NBE, CS)
  - `/m/triton/` or `/m/$dept/{scratch,work}` mounted on workstations
- SSHFS
  - Mount remote directories over SSH
  - Linux: Nautilus → mount folder
  - Linux (command line):

```
$ sshfs triton:/path/to/dir  dir_on_your_computer
```

- SCP/SFTP
  - Copy individual files and directories (inefficiently)

```
$ scp file.txt triton:file.txt
```

- Rsync over SSH
  - Like `scp`, but avoids copying existing files again, smart about big filess
  - `$ rsync -auv --no-group source/ triton:target/`

$WRKDIR

Right after you logged in: `cd $WRKDIR`

Your daily workplace

Exercise: logging in

- Connect to Triton
- List your home directory (`$HOME`) and work directory (`$WRKDIR`)
- Check the load the load on the frontend node: `top` / `uptime`
- What else can you learn about the node?

Slurm:
The batch system

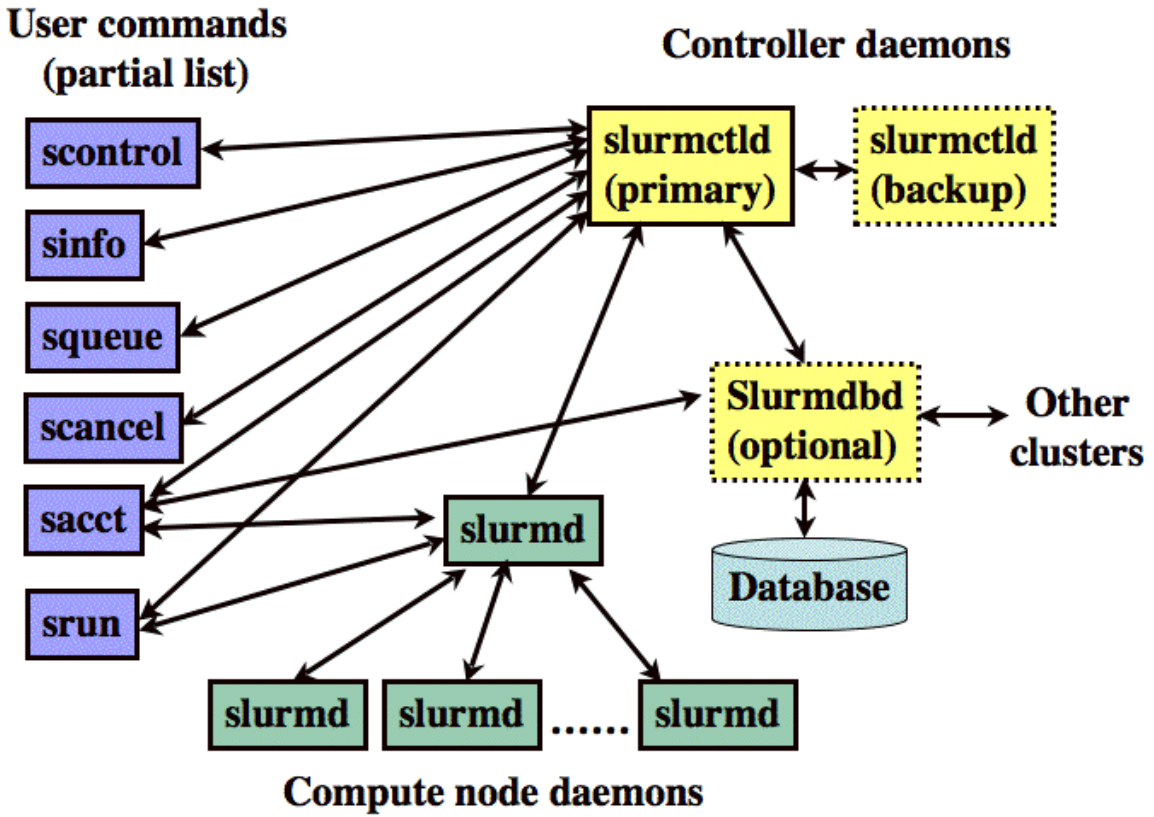
Role of queuing system in HPC

- Can't just run wherever: inefficient use of resources
- Thus, the queue system (batch system)
  - … is a manager that takes care of jobs executions on the cluster
  - .. picks up the job from the user, allocates compute node(s), launches the job, follow its execution status till it ends, reports back to user
- SLURM – http://slurm.schedmd.com/
  - Simple Linux Utility for Resource Management

The job scheduler: SLURM

- Basic units of resources:
  - Nodes / CPU cores
  - Memory
  - Time
  - GPU cards / harddrives
- Takes in jobs requests from users
- Compares user-specified requirements to resources available on compute nodes
- Starts jobs on available machine(s)
- Individual computers and the variety of hardware configurations (mostly) abstracted away
- On Triton we have a **`/etc/slurm/job_submit.lua`** script that selects right QOS and partition based on the user-defined time/mem/CPU job requirements

Slurm components



SLURM (cont.)

- Tracks historical usage (walltime etc; **`sacct`**)
  - ...to enable fair-share scheduling
- Jobs sorted in pending queue by priority (**`sprio`**)
  - Computed from user's historical usage (fair-share), job age (waiting time), and service class (QOS)
- Highest priority jobs are started on compute nodes when resources become available
- Backfill: any spare resources are filled with jobs
- Using cluster time consumes fairshare, lowering priority of future jobs for that user & department

What is a 'job'?

Job = Your program + SLURM instructions

- Consider it as a shell script with the additional instructions
  - Can be a single command or a complex piece of BASH programming
  - If there is more that one command, they are called steps
- SLURM instructions: mainly resource requirements
  - (Nodes or CPUs) x Memory x Time
- Submitted in a single script (.slrm) file from the front node

Slurm instructions (job limits)

In order to request resources efficiently, you must specify what you need:

- Partition: what nodes to run on
- Time: max time job can run
- Nodes/CPUs: how many
- Memory: how much
- Special: GPUs, hard drives, ...

Exercise: check out basic slurm info

Check the output of these commands. You haven't learned what they mean yet, but you will soon.

- `sinfo`
- `squeue`
- `srun -p debug hostname` (you will learn this later)
- `slurm history` (find the jobid)
- `sacct -j $jobid` (use the jobid from above)
- `scontrol show node wsm1`

Job options

Job instructions are special comments in batch script, with them you are saying to SLURM your needs: how long you run, how much memory and CPUs you require, what should be the name and where should go the output etc (see **`man sbatch`** for details). For instance:

When you submit a job using sbatch or srun, you declare what you need

- On the command line:

  ```
  srun --mem-per-cpu=3G --time=12:00
  ```

- Or in a batch script:
  - job.sh:

    ```
    #SBATCH --mem-per-cpu=3G
    #SBATCH --time=12:00
    ```

  - Submit with sbatch: `sbatch job.sh`

Cluster partitions (queues)

`slurm p`

On Triton, partition selection is mostly automatic.

By default most of the time one will use 'batch' queue, but there are cases when one needs to put a partition name explicitly, like 'debug', an instruction for that:

`#SBATCH --partition=`*`name`*

Partition groups compute nodes in to logical, (mostly overlapping) sets. Grouping according to some feature or requirements like run time, GPU cards, limited access:

- *debug* meant for <15 min test runs (to see if code runs)
- Default partition '*batch*' for all sort of runs
- *short* has additional nodes, for <4h jobs
  - Should use it if the task can finish in time; there are no downsides
- *comp, coin, grid* are reserved
- *gpu** is for GPU runs

`slurm p`: partition info

- **`NODES(A/I/O/T)`**: Number of nodes by state in the format "allocated/idle/other/total"

Important: job time limit

Always use estimated time option (!):

```
--time=days-hours:minutes:seconds
```

Three and half days: `--time=3-12`

One hour: `--time=1:00:00`

30 minutes: `--time=30`

One day, two hours and 27 minutes: `--time=1-2:27:00`

Otherwise: the default time limit is the partition's default time limit

By default, the longest runtime is 5 days, the longer runs are possible but user must be explicitly added to the 'long_runs_list'

Important: job memory limit

- Always specify how much memory your job needs.
- `--mem-per-cpu=<MB>` `# or 3G, etc`
- `--mem=<MB>`
- How to figure out how much memory is needed?
  - **`top`** on your workstation, look for RSS column
  - **`/proc/<pid>/status`** or similar on your workstation, see VmHWM field.
  - Just run and check:
    - **`slurm history 1day | less -S`** for a completed job and **`sstat -j $jobid`** for running. Check MaxRSS column. Fix your jobscript, and iterate.
    - Note: MaxRSS is sampled, might be inaccurate for very short jobs!
    - Note 2: For parallel jobs, only works properly if “srun” was used to launch every job step (NOT mpirun).

More slurm options

There are many, many different options (see **`man sbatch`** for details). For instance:

```
#SBATCH --job-name=my_name
#SBATCH --output=my_name.%j.out
#SBATCH --feature=$name
```
- hardware limit

And many more… see the user guide reference.

Use case: test runs

Several dedicated machines for up to 15 minutes test runs and debugging

```
$ sbatch -p debug my_job.slrm
```

or

```
$ srun -p debug my_binary
```

Practical advice

- Declare what you need. Slurm will do the right thing.
- Finding memory needs
  - Start on desktop
  - Short runs in interactive or debug partitions
- Time limits
  - Start short for debugging and get longer
  - Then start long and decrease as you can, because less resources = faster runs.

Exercises

- Repeat steps from previous exercise: do you understand anything more?
- Use `sacct` to learn partition, memory, time limit of job 20421099.
  - `sacct -j 20421099`
  - You have to look at the sacct manual page to figure out how to print more details (`man sacct`, check out `-o`/`--format`)

Quick aside:
Installed software

1000 users: how can we please everyone?

- Module system: selectively load what you need, even the exact version
- Pro: you can get what you need
- Con: you have to select what you need

Software: modules environment

```
$ python3
-bash: python3: command not found
$ module load anaconda3
$ python3 --version
Python 3.6.0 |Anaconda 4.3.0 (64-bit)
```

- **`module load`** : adds installed programs and libraries to $PATH, $LD_LIBRARY_PATH, etc.
- Can specify both programs and versions

Module environment

```
$ module avail
```

```
------------------------------------------ /cvmfs/fgi.csc.fi/modules/ ------------------------------------------
PrgEnv-amd
PrgEnv-gnu
acml/5.1.0
acml-fma4/
acml-fma4-
acml-mp/5.
amber/12
amd-app-sdk/2.7                 gcc/4.7.0                       orca/2.9.1
amd-gdebugger/6.2.438           gcc/4.7.1                       parcas/2012-10-openmpi
amd-libm/3.0.2                  gcc/4.7.2                       pgi/12.4(default)
amd-x86_open64/4.5.1            gcc/4.8.0                       pyfits/3.1.1
amd-x86_open64/4.5.2            geant4/4.9.5p1                  python/2.7.3
asciidata/1.1.1                 gpaw/0.9rev9050-ase3.6.0-intel  python/2.7.4
ase/3.6.0rev2515                gpaw/0.9rev9473-ase3.6.0-intel  python3/3.2.3
clamdblas/1.8.269               gromacs/4.5.5                   python3/3.2.4
clamdfft/1.8.269                intel/2011.11.339               python3/3.3.0
clustername/1                   intel/2011.8.273(default)       python3/3.3.1
cmake/2.8.9                     intel-ocl-sdk/2012              scalasca/1.4.2
cp2k/2.3rev12343                mkl/2013.1.117                  scipy/0.10.1
cuda/4.0                        mpip/3.3                        scipy/0.11.0
cuda/4.1                        mvapich2/1.8-gcc                siesta/3.1-pl20
cuda/5.0                        mvapich2/1.8-intel              siesta/trunk-431
dalton/dalton2011               mvapich2/1.9a                   valgrind/3.7.0
dalton/lsdalton2011             numpy/1.6.2
distribute/0.6.34               numpy/1.7.0

---------------------------------------- /share/apps/modulefiles/ ----------------------------------------
PyYAML/3.10                     gotoblas2/opteron-1.13-gcc  mne/2.7.3                   sagemath/5.8
R/2.15.2                        gotoblas2/xeon-1.13-gcc     mvapich2/1.8a2-gcc-4.4.6    scalapack/gcc-openmpi
basemap/1.0.5                   igraph/0.6                  mvapich2/1.8rc1-intel       scalapack/xeon-gcc-mvapich2
cmake/2.8.9                     lammps/22Feb13              numpy/1.7.0b2               scipy/0.11.0rc2
comsol/42a                      lapack/opteron-3.4.0-gcc    octave/3.6.3-gcc4.4         shapelib/1.2.10
comsol/43a                      lapack/xeon-3.4.0-gcc       octave/3.6.3-intel          shapelib/1.3.0
comsol/43b                      matlab/r2011b               openmpi/1.4.5-gcc-4.4.6     tutorialtools/2013
freesurfer/5.1.0                matlab/r2012a               openmpi/1.4.5-icc           vasp/5.2.12
fsl/4.1.9                       matlab/r2012b(default)      python-igraph/0.6           vasp/5.3.2
geos/3.3.5                      matlab/r2013a               python-tritontools/2.7.4    vasp/5.3.3
gotoblas2/archdep-1.13-gcc      matplotlib/1.1.1            python3-tritontools/3.3.1
```

Common application modules

- Python: module load anaconda2 or anaconda3
- Matlab: module load matlab
- R: module load R

Interactive jobs

Interactive jobs

- Triton (and similar) designed for batch running: submit and check later
- But there is an fast way to get started: interactive jobs
- PROS
  - Fast to get started
  - Easy to see job status
- CONS
  - Running more than one job is hard
  - Higher chance of job dying
  - Inefficient

You can start here, but don't end here

Interactive run

Just add srun!

- Say you have some program that works:

  $ hostname

  login2

  $ python pi.py

  3.140464

- To run on Triton, “just add srun!”:

  $ srun -p interactive hostname

  cn01

- $ srun -p interactive python pi.py

  3.140608

- This requests resources from the queue, waits for them, runs your program, then returns.
- Use slurm options: `-p PARTITION`, `--time=X`, `--mem-per-cpu=X`, `--mem=X`, etc.

Interactive shell

- The following let you get a shell in the queue, for interactive jobs
- `srun -p interactive --pty bash` … Start a shell on the node. Run what you need there. Use more srun commands to monitor individual steps.
- `sinteractive` … Similar to above, but more clever and allows graphical applications.
- You need to remember to close these when you are done, otherwise resources stay allocated to you!

Exercise: Hello ~~World~~ Triton!

```
$ ssh triton.aalto.fi
$ cd $WRKDIR
$ srun -p interactive echo 'Hello, Triton!'
```


```
srun: job 20421151 queued and waiting for resources
srun: job 20421151 has been allocated resources
Hello, Triton!
```

- Then: Try adding some more options, such as -p, --time, or –mem
- Check different history commands: slurm history, sacct, (scontrol show job)
- Try running other short jobs in the queue
-

Batch scripts

Batch scripts: asynchronous jobs

- Job gets a unique jobID, used to track outcome and history.
- Job waits in PENDING state until the resource manager finds available resources on matching node(s)
- Job script is executed on the nodes assigned to it; the requested memory and CPUs are reserved for it for a certain time
  - When job is running, user has access to the node(s) her/his job is running on, otherwise SSH access to compute nodes is restricted
- Program output saved to '.out' and '.err' files
- Check status later from output files/slurm history.

Monitoring

- Running jobs

  `slurm q` or `slurm watch q`

- Finished jobs

```
$ slurm history
```

or

```
$ slurm history 4hours
$ slurm history 3days
```

The `slurm` utility

- Triton-specific wrapper by Tapio Leipälä for Slurm commands
  - `squeue, sinfo, scontrol, sacct, sstat ...`
- Informative commands only (safe to test out)
- Shows detailed information about jobs, queues, nodes, and job history
- To get help run `slurm` with no arguments

Slurm natives commands

slurm is just a wrapper around other commands:

- **slurm p** →

```
sinfo "%10P %.11l %.15F %10f %N"
```

- **slurm q** →

```
squeue -S T,P,-S,-i -o "%18i %9P %14j
%.11M %.16S %.8T %R" -u $USER
```

- **slurm j <job_ID>** →

```
scontrol show job <job_ID>
```

Exercise: Hello Triton! with sbatch


```
$ edit Hello_Triton.slrm
#!/bin/bash
#SBATCH --time=1
#SBATCH -p debug
#SBATCH --mem-per-cpu=10M
/bin/echo 'Hello Triton!'
$ sbatch Hello_Triton.slrm
Submitted batch job 4086983
$ ls -tr
slurm-4086983.out
```

Exercise: many steps

```
#!/bin/bash
#SBATCH --time=1
#SBATCH -p debug
#SBATCH --mem-per-cpu=10M
srun hostname
srun echo 'Hello Triton!'
srun date
```

How does this appear in the slurm history?

What is the purpose of extra srun commands in your script?

Exercise: submit a job and cancel it

- Submit a job (use scripts above, but change partition to “batch”)
- Cancel it with `scancel`

Exercise: batch script in another language (advanced)

- Batch scripts can also be written in other languages. Use the proper #! line at the top.
- Create a batch script in another language, say Python or R. Can it run?

General tips and tricks

- Start with interactive if you like, but (almost) never stop there
- Slurm is declarative. Say what you need and it will generally do the right thing.
- Monitor your resource usage and adjust scripts as needed.
- It’s worth putting some time to make your run scripts organized and flexible
- Smaller resources = faster to run. Take some time to tune your resources.
- Slurm has *many* options: check the wiki. Email, job dependencies, output files, names, …
- Slurm options (--time, etc) can be both in batch script and command line.
- If you have problems, ask!

References and questions/comments?

- In the afternoon: running parallel jobs

References

- Wiki: interactive jobs https://wiki.aalto.fi/display/Triton/Interactive+jobs+tutorial
- Wiki: batch scripts: https://wiki.aalto.fi/display/Triton/Serial+jobs+tutorial
- Wiki: reference: https://wiki.aalto.fi/display/Triton/Reference