Triton Cheatsheet v2.2 / 2024-04-30

Full documentation: scicomp.aalto.fi/triton/   Quick reference: scicomp.aalto.fi/triton/ref/

About Triton

- Over 10000 CPUs, 200 GPUs available, up to 256GB or 2TB memory/node.
- Available for all Aalto staff for any research.
- Good integration with department workstations: most filesystems are cross-mounted and you can easily open and process files as if they were local.
- Rather than expect your workstation to do everything, develop to Triton and you can scale up to whatever resources you need.
- Example Triton workflows: Test code on frontend node. Submit interactive test jobs with "`srun -p debug ./your-command`" for *fast* testing. For production runs, do the same but to bigger partitions using more CPUs, or use batch submissions. Examine output on your own workstation via `/m/DEPT/scratch/`.

Getting help *docs: Getting Triton help, see also scicomp.aalto.fi/help/*

- All information on scicomp.aalto.fi/triton/. Includes quickstart tutorials.
- SciComp garage (help session): daily at 13:00: scicomp.aalto.fi/help/garage/
- Issue tracker: scicomp.aalto.fi/triton/issues (please no personal mail)
- Chat: scicomp.zulip.cs.aalto.fi - good for quick questions
- CS, NBE, and PHYS IT overlap with Triton support and can provide advice as well.
- Many courses in practical computing topics: scicomp.aalto.fi/training/
- Aalto RSE service gives advanced support: scicomp.aalto.fi/rse/

Connecting *docs: Tutorials/Connecting, see scicomp.aalto.fi/triton/tut/connecting/*

- Accounts are same as Aalto accounts, but need activation: request from link above.
- Login: ssh to `triton.aalto.fi` with Aalto user/pass or ssh keys. Use Aalto VPN.
- ondemand.triton.aalto.fi provides a web interface (including Jupyter).

Data Storage *docs: Tutorials/Data storage*

- /scratch is a Lustre filesystem: 5PB, networked and highly parallel. Also available on (CS,NBE) workstations. All calculation data goes here.
- Using local disks can be more efficient for high I/O processes.
- Other department filesystems (CS,NBE) are on login node and group servers.

B=backed up, S=shared

	B	S	
$HOME	+	+	Home dir, 10GB. Codes and configuration, not calculation files.
/m/DEPT/scratch/PROJ/		+	Shared Lustre FS. Large and fast. Per-project. **NO BACKUPS.**
/m/DEPT/work/USER/		+	Same as above, per-user. **NO BACKUPS**
/tmp/			Local disk storage. Not backed up.
version.aalto.fi			Aalto git repository.
$XDG_RUNTIME_DIR			Ramfs (in-memory filesystem): very temporary but fast space

Software availability *docs: Tutorials/Applications*

- Most software and libraries are in the "module" system. This allows you to select what you need, including exact versions. It just changes environment variables like `$PATH`, `$LD_LIBRARY_PATH`, etc. Use "env" to print these.
- Admins can install common software for you: just ask.
- The "`module`" function makes software available. Example: `module load matlab` or `module load matlab/r2023b` (better, as you know which version you get).
- Modules also contain dependencies: if you load E, it will automatically load A, B, C, D if needed. So just request what you need.
- "`which`" shows exactly what a command name will run.

`module spider PATTERN`	Search (full) for modules matching pattern.
`module spider PATTERN/VER`	Show how to load this module (may need to load others first)
`Module show NAME`	Show module details, exactly what it does.
`module load NAME`	Load a module. Specify version with `NAME/VERSION`.
`module unload NAME`	Unload a module.
`module list`	List currently loaded modules.
`module purge`	Remove all loaded modules from the current session.
`module save ALIAS`	Save/restore currently loaded modules to a collection. Loading a collection is much faster.
`module restore ALIAS`	
`module savelist`	List saved collections.

Common software

- Python: we recommend the Anaconda modules for general-purpose Python. "`module load scicomp-python-env`" for Python 3. For custom packages, see "conda" below.
- R: `module load scicomp-r-env`
- Matlab: `module load matlab`
- Mathematica: `module load mathematica`
- And so on… see user guide and/or discuss your needs with us.

Conda *docs: Apps / Python environments with Conda*

- Conda is the recommended way to install Python software
- Module `mamba` provides `mamba` and `conda` commands.
- `mamba` is a much faster drop-in replacement for `conda` .
- We recommend *not* running `conda init`, and instead use `source activate` instead of `conda activate`.
- `environment.yml` makes environments reproducible, example at right.
- Make own environment: `conda env create --file environment.yml`
- In batch scripts: `module load mamba` and `source activate NAME`
- More information can be found at scicomp.aalto.fi/triton/apps/python-conda/

```
name: example-env
channels:
    - conda-forge
dependencies:
    - numpy
```

Interactive jobs *docs: Tutorials/Interactive jobs*

- Easiest way to use triton: "**Just add srun!**" to your working command, and specify how much power you need. (details described on next page)
- Example: `srun --mem=50G --time=5:00 --cpus-per-task 6 ./your_command`
  - (50GB Memory, 5 hours max runtime, 6 CPUs)
- `sinteractive` gets you a shell which is also usable for graphical applications.
- `slurm history` shows detailed CPU/memory usage of the process.

Batch jobs *docs: Tutorials/Serial jobs*

```
#!/bin/bash -l
#SBATCH --mem=50G
#SBATCH --cpus-per-task 4
#SBATCH --gres=gpu

module purge
module load NAME

srun ./step_1 1 5
srun ./step_2 1 5
```

- Once you run interactively, you can make batch jobs which run in the background - submit and return for results later.
- Example script at right. Options can be inside the script. Output goes to files in the same directory.
- Submit job with `sbatch script-name.sh`
- Monitor with `slurm queue`.
- `slurm history` shows resource usage, including details on CPU/time/memory for *each srun step*.
- Slurm will run the batch script only once.
- Slurm will allocate as many CPUs as you request (`-c N`). It is up to you to make sure your job can use them.

Parallel jobs *docs: Tutorials/Parallel computing*

```
#!/bin/sh
#SBATCH --time=5:00
#SBATCH -n 4
#SBATCH --array=1-10

srun ./my-command  \
    input_$SLURM_ARRAY_TASK_ID  \
    -o OUTPUT_$SLURM_ARRAY_TASK_ID
```

- Easy: Array jobs. Use `--array=M-N` with sbatch and you can easily scan parameters using `$SLURM_ARRAY_TASK_ID`. The command is run once with each parameter. Good for parameter sweeps.
  - Example at right: Run with `sbatch script.sh`
- MPI, OpenMP, etc instructions in docs.
- OpenMP: Usually with `-c`. Use `export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK`
- MPI: See docs. Usually with `-n`.
- Python/R/other languages: Usually with `-c`, but depends on the code. Must be checked individually.
- Use `seff JOBID` to verify efficiency.

GPUs *docs: Tutorials/GPU computing*

- request with `--gres=gpu` . Can select type with `--constraint=NAME` . Recent names include `ampere`, `volta` , `pascal`, and `kepler`.
- Check efficiency with `sacct -j JOBID -o TRESUsageInAve -p` after job completion.
- GPUs can be hard to use efficiently (especially data loading)! Usage statistics only show the active time but do not show the real occupancy levels. Ask for help early.

Slurm details *docs: User guide/Reference, Running programs on Triton*

- *Slurm* is the system which allocates CPU, GPUs, etc. to people doing computation.
- The core is a queuing system which fairly prioritizes users. The less you run, the higher your priority.
- Work is submitted as jobs. CPUs, memory, and time must be declared for jobs. Jobs killed if these limits are exceeded too much.
- In general, just declare what you need and Slurm will do the right thing.

The following commands give history about jobs:

slurm queue (slurm qq)	Your currently queued jobs, or `slurm watch queue` for updating view
slurm history 1day|2hour|...	Your recently completed jobs, with detailed time/memory info.
slurm job JOBID	Info on a certain job.
seff JOBID	Check effectiveness of requested resources.
squeue / sacct / scontrol	Advanced info on waiting jobs / finished jobs / running jobs.

Slurm commands **Further reference:** scicomp.aalto.fi/triton/ref/#job-submission

The following commands submit jobs. All require some of the slurm options.

srun	Run a single command on nodes, I/O connected to terminal.
srun (in batch script)	Run a job step so that time/memory can be separately tracked
srun --pty [bash]	Run a command (or shell) with full terminal support.
sinteractive	Start a shell on a node, usable for graphical applications.
sbatch	Run a batch script. Submits and returns immediately.
scancel JOBID	Cancel a running job

Slurm options for `srun`, `sbatch`, or `#SBATCH` in batch scripts:

--time XXX	Total job run time (`HH:MM[:SS]` or `DD-HH`)
-c N	Number of cores (per task)
--mem nnG	Total memory per node, only for single node jobs.
--mem-per-cpu nnG	Total memory per CPU
-p PARTITION	Partition to use (usually leave off) `slurm p`
-N N	Number of nodes
-n n	Number of tasks to start (number of individual srun processes to start)
-J JOBNAME	Specify memorable job name
-o FILE, -e FILE	Job stdout/stderr is saved to this file name. Default to same dir+jobid.
--array=N-M	Array job, easy parallelization (*only* with `sbatch`). `$SLURM_ARRAY_TASK_ID`
--constraint XXX	Request hardware type (`hsw, bdw, skl, csl, milan, avx, avx2, avx512,`)
--gpus=n (request n GPUs, for gpu partition), --constraint= (GPU generations: `kepler`, `pascal`, `volta`, `ampere`). AMD GPUs: `-p gpu-amd` and no `--gres` . `--exclusive` (whole-node)	