Aalto University Site Report

Janne Blomqvist, Ivan Degtyarenko, Mikko Hakala

2014-09-24

Aalto University

- Fusion of 3 independent Universities in 2010
  - Helsinki University of Technology
  - Helsinki School of Economics
  - University of Art and Design
- Organized in 6 “schools” (=faculties)
- 20000 students
- 5000 faculty and staff
  - 370 professors

Aalto School of Science (Aalto SCI)

- “Basic Science” faculty in Aalto University.
- Math, Physics, Biomedical Engineering, Computer Science, etc.
- Science-IT project (that’s us!) takes care of scientific computing infrastructure.
    - Procurement
    - Running of cluster(s)
    - User education

Computational Science @Aalto-SCI

- Density Functional Theory
- Molecular Dynamics
- Statistical Physics (Phase transitions etc.)
- Quantum Many-Body Theory
- Nuclear reactor physics (fission & fusion)
- Brain imaging analysis
- Machine learning
- Genomic analysis
- Speech and language processing
- Image analysis

Science-IT project

- Organized per a “stakeholder” model where major users provide money for personnel and procurements.
  - Currently 3 departments are the major stakeholders.
- New users “opt-in”, usage is free as long as it’s reasonably small.

Triton cluster

- Our current "workhorse" cluster.
- Standard x86(-64) servers of varying ages.
  - 6-core Opteron CPU's.
  - 6-core Xeon Westmere CPU's.
  - 10-core Xeon Ivy Bridge CPU's.
- Most servers are 2-socket ones. A couple "hugemem" nodes with 1 TB RAM and 4x 8-core Xeon X7542 for jobs with large memory requirements.
- A few GPU nodes.
- Total ~550 nodes, ~7000 cores.
- Part of Finnish Grid Initiative (FGI). Possible for external users to run jobs via grid environment (20% share).

Slurm@Triton

- Used Slurm since beginning on Triton (SGE on previous cluster).
- Generally very happy with it.
- The community is nice.
- Have implemented a few features and contributed some bugfixes over the years.
- Two main compute partitions, “batch” and “short”, mostly overlapping.
  - Constraints to force specific hw (e.g. –constraint=xeonib).
  - QoS to boost priority for short jobs.
  - job submit plugin selects QoS and partition automatically depending on job timelimit if not explicitly specified.

Ticket-Based Fairshare algorithm 1/4

- Due to the stakeholder model, it's important for us that the fairshare algorithm drives towards equilbrium for different parent accounts
  - This wasn't the case with the original fairshare algorithm due to different usage from different departments
- So we created the Ticket-Based Fairshare algorithm.
  - PriorityFlags=Ticket_Based , originally PriorityType=priority/multifactor2 in Slurm 2.5.

Ticket-Based Fairshare algorithm 2/4

- Start with a number of *tickets* at the root of the account tree.
- Tickets are distributed to active child nodes in the tree proportional to the fair share weights (active = account or subaccount has pending jobs).
- In the end, the user with the most tickets get the fair-share priority 1.0, the rest of the active users proportional to how many tickets they have compared to the user with the most tickets.

Ticket-Based Fairshare algorithm 3/4

- The good
  - Much better at balancing department usage compared to the original fairshare algorithm.
  - Easy to balance the fairshare weight vs. other priority_multifactor weights since the highest priority job always has fair-share priority 1.0.
- The bad
  - Still happens that department usage can get unbalanced. Consider e.g. department X with 1 (very) active user vs. department Y with N active users. Since the dept. X tickets are all given to 1 user vs. dept. Y tickets are distributed over N users, dept. X user gets higher priority even though dept. Y may have much higher fair share factor.
  - Prioritites fluctuate depending on the queue situation, unintuitive for users.

Ticket-based fair-share algorithm 4/4

- Subsequently several other fairshare algorithms have been proposed.
- Depth-oblivious (CEA), see SUG 2013 slides.
- ~~Level-based~~ Fair Tree (BYU), SUG 2014 presentation earlier today.
- Haven't yet had time to try them out.

Triton usage profile

- We have *a lot* of users who submit lots of serial jobs (yes, array jobs are awesome for this).
- We also have users who submit small and medium-sized parallel jobs (a few hundred cpu's at most). Mostly physics + a bit of hadoop etc.
- We don't have really large parallel jobs - users requiring this tend to use the national level resources for such jobs.
- Turns out our workload is challenging for slurm
  - Due to lots of small jobs, we want fast scheduling. Additionaly, sometimes these small jobs are also very short.
  - Due to the parallel and/or long-running jobs, we want sophisticated scheduling, with deep backfill lookahead etc.
  - Finding a suitable combination of scheduler parameters is a whack-a-mole game!

slurm utility

- Beginner users where often confused about the variety of slurm commands
- Created a wrapper for various slurm querying functions
  - Under the hood, uses squeue, scontrol, sshare, sacct, sstat, sprio, etc. as appropriate
  - Only read-only commands, so always safe to use (doesn't accidentally kill all your jobs!)

https://github.com/jabl/slurm_tool

Slurm gripes & wishlist 1/2

- Behavior under load & scalability.
  - srun job steps failing due to “send/recv timeout” makes users very angry. Possible to increase the srun timeout when running inside an allocation?
  - Longer term?
    - More efficient connection handling, e.g. non-blocking sockets with epoll() instead of thread-per-connection?
    - More fine-grained locking, or something different such as RCU?
- Resiliency
  - In backup mode, every connection waits a while trying the master before trying the backup.
  - No master/slave for slurmdbd.
  - Requirement for a shared state save directory enlarges the problem to also require a high availability NFS server.
  - Push state save problem to some lightweight replicated key-value DB (Redis/etcd?). StateSavePlugin=?

Slurm gripes & wishlist 2/2

- Account memory/gres/etc in addition to cpu-secs. Handle cpu's with different performance. Bug #858 has some interesting work in this direction by Ryan Cox, BYU.
- Containers
  - Lots of work in this area thanks to cloud computing.
  - Checkpoint - migrate (reschedule!) - restart would be nice.
    - E.g. pack serial jobs to allow a parallel –exclusive job to start.
    - Maintenance without waiting for long-running jobs to finish
  - libcontainer (docker/google/redhat/parallels/ubuntu)
    - Job as a container?
  - Container job? "My software requires Ubuntu 10.10!"
- A pony.

That's all, folks

Thank you for listening. Questions?