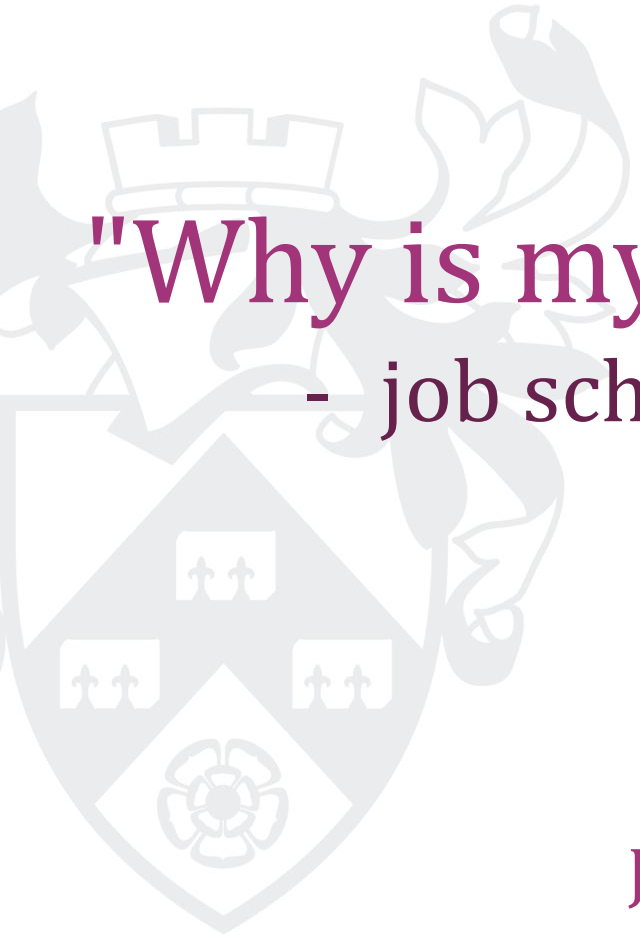


"Why is my job not running?"

- job scheduling on Viking -

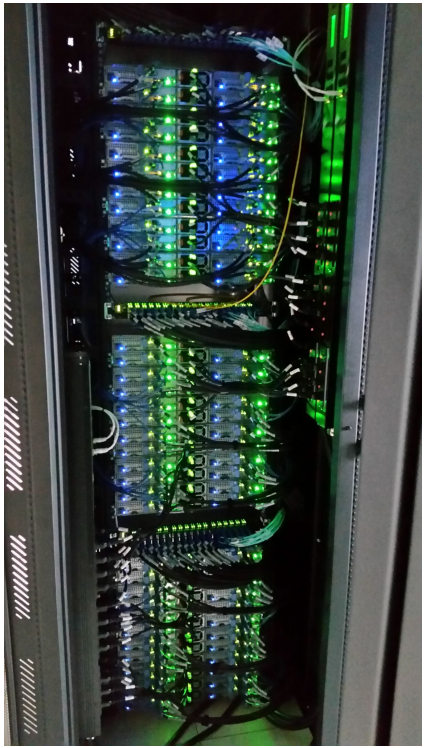
Jasper Grimm

Overview

- Viking
- Interacting with Slurm
- Job priority, fairshare and nice-ing
- Queue order
- Backfilling and preemption
- Checkpointing
- Conclusion

Viking

- 178 compute nodes
- Nodes divided into '*partitions*':
  - `nodes (default), week, month, gpu, himem, himem_week, interactive, test`
  - Full list (with time limits): `sinfo -s`
- Uses Slurm, which handles:
  - Job scheduling
  - Allocating resources
  - Startup, execution and cleanup of jobs



Interacting with Slurm

- Job submission:
  - `srun`: execute application (obtaining allocation if needed)
  - `sbatch`: submit batch script
  - `salloc`: request job allocation
- Job management:
  - `sbcast`: transfer file(s) to all of a job's allocated compute nodes
  - `scancel`: cancel job
  - `squeue`: list queued and running jobs
  - `sinfo`: information about nodes/partitions
  - `scontrol`: view or update configurations/jobs

Interacting with Slurm (cont.)

- Accounting:
  - `sacct`: record of jobs submitted
  - `sacctmgr`: view account information

Submitting a job

Two main methods:

- srun:

```
$ srun --ntasks=1 --time=0:05:00 ./example
```

- sbatch:

```
$ sbatch simple_job.sh
```

Interactive bash shell:

```
$ srun --ntasks=1 --time=0:05:00 --pty /bin/bash
```

Job State

- Current state in queue: `squeue -u $USER --start`
- For completed jobs: `sacct -u $USER`
- “Job Reason” provides additional information for pending jobs:
  - Priority: other higher-priority jobs are currently in the queue
  - Resources: job has been scheduled, and is waiting on resources to become free
  - QOS*Limits: scheduling the job would put you above the resource limits for the partition

Job priority

Job priority depends on:

- Site*
- Age
- Association*
- Fairshare
- Size
- Partition*
- QoS*
- Nice
- TRES

```
Job_priority =
        site_factor +
        (PriorityWeightAge) * (age_factor) +
        (PriorityWeightAssoc) * (assoc_factor) +
        (PriorityWeightFairshare) * (fair-share_factor) +
        (PriorityWeightJobSize) * (job_size_factor) +
        (PriorityWeightPartition) * (partition_factor) +
        (PriorityWeightQOS) * (QOS_factor) +
        SUM(TRES_weight_cpu * TRES_factor_cpu,
            TRES_weight_<type> * TRES_factor_<type>,
            ...)
        - nice_factor
```

*not used on Viking

Fairshare (Fair-Tree Algorithm)

- Prioritises users who have been under-served recently
- Accounts are allocated a Level Fairshare (LF = S/U)
  - S: normalised number of shares (constant on Viking)
  - U: effective usage ($U = U_{self} / U_{self+siblings}$)
  - Usage calculated based on TRESBillingWeights:
    - CPU=1.0, Mem=0.25G in most partitions
    - GRES/gpu=2.0 in GPU partition
  - Usage decays with a half-life of 7 days (on Viking)
- Users' LF is similarly calculated within each account
- Fairshare is normalised relative to highest LF user

Nice

- Used to modify a job's priority (nice > 0 decreases priority)
  - can reorder queue priority of your jobs
- Automatically adjusted in the GPU partition
  - only GPU resources are billed
  - jobs are nice-d to negate the priority increase from "job size" (requesting extra CPUs/mem)

Queue order

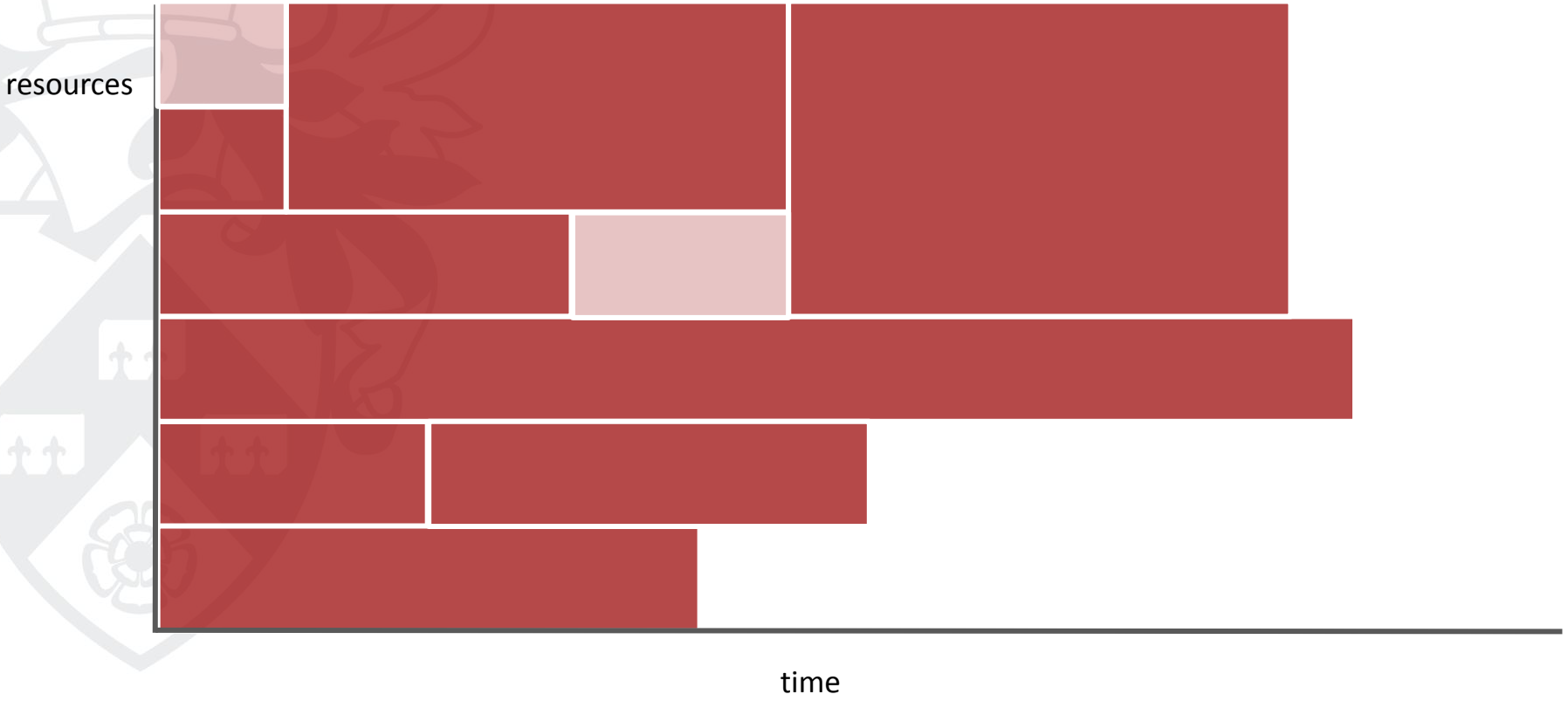
- Job priority is *not* the only factor in the order in which jobs are considered for scheduling

Preemption	Advanced reservations	Partition PriorityTier	Job priority	Job submit time/ID
Jobs that can be preempted have lower queue priority	Jobs submitted to advance reservations have higher queue priority	Higher PriorityTier partitions will be considered first	Higher priority jobs considered first	Older jobs considered before more recent jobs

- Quick evaluation when a job is submitted or completes
- Comprehensive scheduling attempts made less frequently

Backfilling

- Without backfilling, jobs would only be scheduled in priority order
- Lower-priority jobs are scheduled if this does not delay higher priority jobs
- More accurate when job requests match their runtime
- Backfilling is time-consuming, so not all jobs may be considered
- On Viking, most jobs are backfilled!


resources
time

Preemption

- Partitions limit total resources that can be allocated to one user
- In periods of low utilisation, there are free resources available
- Solution: `preempt` partition (overlaps `nodes`)
  - has no resource limits (other than physical)
  - jobs in the `preempt` partition don't count towards the limits in `nodes`
  - when resources are needed for a normally-scheduled job, preemptive jobs are terminated (requeued)
  - Slurm will try to minimise the number of jobs preempted
  - With backfilling, entire nodes will always be preempted

Checkpointing

- “Snapshot” the state of a program, so that execution can be resumed later
- Some software has native support, others can use e.g. DMTCP
- Multiple benefits:
  - fault tolerance
  - extended runtime
  - more resources available to long jobs

See: https://wiki.york.ac.uk/display/RCS/VK21%29+Checkpointing+with+DMTCP

Automatic re-submission

- Possible to write a job script that automatically re-submits itself until complete
- Reduces manual overhead
- Example job scripts available on the Viking wiki

```
# Check if job has been restarted
nrest=${SLURM_RESTART_COUNT}
if [[ "${nrest}" -eq "0" ]]; then
    # First time, use dmtcp_launch to start job
    dmtcp_launch --rm -j --no-gzip --ckptdir ${ckpt_dir} ${run_cmd} &

elif [[ "${nrest}" -gt "0" ]] && [[ -e dmtcp_restart_script.sh ]]; then
    # Restart job in background
    ./dmtcp_restart_script.sh &

else
    echo "Failed to restart job. Exiting..."
    exit
fi
```

Conclusion

- Slurm can provide estimates of when your job will start
- Request only the resources you need
  - (with some buffer, generally ~30%)
  - Especially walltime, which has a big impact on scheduler/backfill performance
- Job priority depends mainly on age, fairshare and resources
- Fairshare decays over time
- Most (smaller) jobs are backfilled

Good practice

- Ensure resource requests are reasonable
- Use job arrays for large volumes of similar jobs
  - When each iteration takes less than a few minutes, a bash loop within a single job script will reduce overhead
- Avoid submitting large volumes of individual jobs at once
  - e.g. submitting 1000 job scripts with a bash for-loop
- Do not load modules built with different toolchains

Toolchains on Viking