An Exascale Slurm Testing and Evaluation Environment Utilising Generated DAG Workloads

Laslo Hunhold, Stefan Wesner

Parallel and Distributed Systems Group
University of Cologne

16th May 2024

Motivation

Current state of Slurm simulators

- Mostly focus on scheduling performance
- Very fragile and broken due to large changes to Slurm codebase
- DAG workloads not supported, but more and more common due to workflow managers/meta-schedulers

Exascale challenges

- Some previously trivial things become very difficult (e.g. GPU testing/monitoring)
- Monitoring/data analysis tools require more testing
- No public access representative workload data

Reproducibility

- Many MODA innovations are unpublished, especially for exascale systems
- Reproduction/Documentation is often difficult due to tight integration

Goals

Extend the scope of Slurm simulators

- Job prologs, epilogs, inter-job profiling, etc.
- Observe plugin interactions, facilitate software integration/testing
- Exascale testing and evaluation environment on a single node

Provide a robust and flexible solution

- Minimal code changes
- Wide range of applications (monitoring, analytics, etc.)

Improve reproducibility

- Clearly document environment setup
- Provide environment for testing and evaluation of exascale tooling on test machines

DAG Workload Generation (1/2)

Algorithm

- No data available, need to generate DAGs
- Proposed algorithm

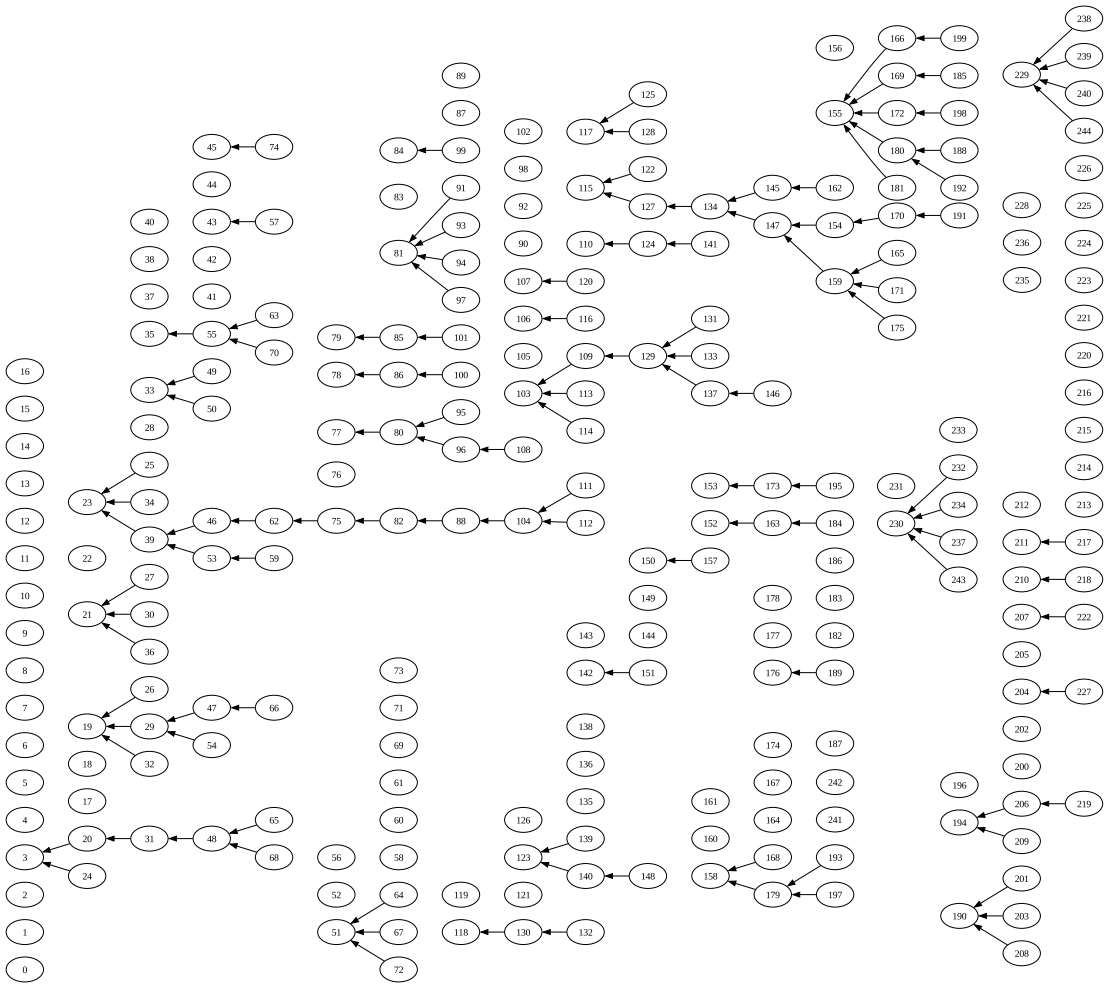
---

input : $r \in \mathbb{N}_0$: number of ranks
$\underline{n} \leq \overline{n} \in \mathbb{N}_1$: min./max. nodes per rank
$\underline{d} \leq \overline{d} \in \mathbb{N}_0$: min./max. deps. per node
output: $G := (V, E)$: directed acyclic graph
$R_1, \ldots, R_r$: sets of nodes of each rank

$G = (V, E) \leftarrow (\emptyset, \emptyset)$
for $i \leftarrow 1$ **to** r **do**
  $n_i \leftarrow \texttt{DiscreteUniformRandom}(\underline{n}, \overline{n})$
  $R_i \leftarrow \texttt{GenerateNodes}(n_i)$
  $V \leftarrow V \cup R_i$
  if $i > 1$ **then**
    for $j \leftarrow 1$ **to** n_i **do**
      $d \leftarrow \texttt{Min}(n_{i-1}, \texttt{DiscreteUniformRandom}(\underline{d}, \overline{d}))$
      $D \leftarrow \texttt{DrawDistinctRandomFrom}(R_{i-1}, d)$
      for $k \leftarrow 1$ **to** d **do**
        $E \leftarrow E \cup (R_i[j], D[k])$
      end
    end
  end
end

DAG Workload Generation (2/2)

Example and Parameter Choice



Parameter choice: High sparsity, low node degree

Job parameters: Randomly generated separately as ‘node weights’

Exascale Slurm Setup (1/3)

Reference and first steps

Reference system: Frontier, Oak Ridge National Labs (1.194 EFLOPS)

- 74 rack cabinets, 64 blades per cabinet, 2 nodes per blade
- 9,472 $\approx$ 10,000 nodes, 64 cores per node

What stop us from running Slurm with 10,000 nodes on a single machine?

Basic configuration

- Compile Slurm from source with special flag
  `./configure ... --enable-multiple-slurmd`
- Modify `slurm.conf`: Include `%n` (node name) in `SlurmdSpoolDir`, `SlurmdLogFile`, `SlurmdPidFile`
- Add node definitions

```
NodeName=atom[00000-09999] NodeHostname=HOSTNAME
Port=[10000-19999] Sockets=1 CPUs=1 CoresPerSocket=64
ThreadsPerCore=128 State=UNKNOWN PartitionName=part1
Nodes=ALL Default=YES MaxTime=INFINITE State=Up
```

Does it work? No, hits hard cgroup limits after 1,500 nodes

Exascale Slurm Setup (2/3)

Trimming overhead

- Disable cgroup process tracking (Set `ProctrackType` to `proctrack/pgid`)
- Disable task plugin

Does it work? No, nodes launch but are lobotomised

It turns out (after many hours)

- Systemd insists on being a 'cgroup broker' via dbus
- Chokes after around 2,500 nodes, no direct error
- Nodes start but can't communicate with control daemon
- Solution (found in Slurm source, but also buried in the manual): Create a file `/etc/cgroup.conf` containing `IgnoreSystemd=yes`

Exascale Slurm Setup (3/3)

System integration and helper scripts

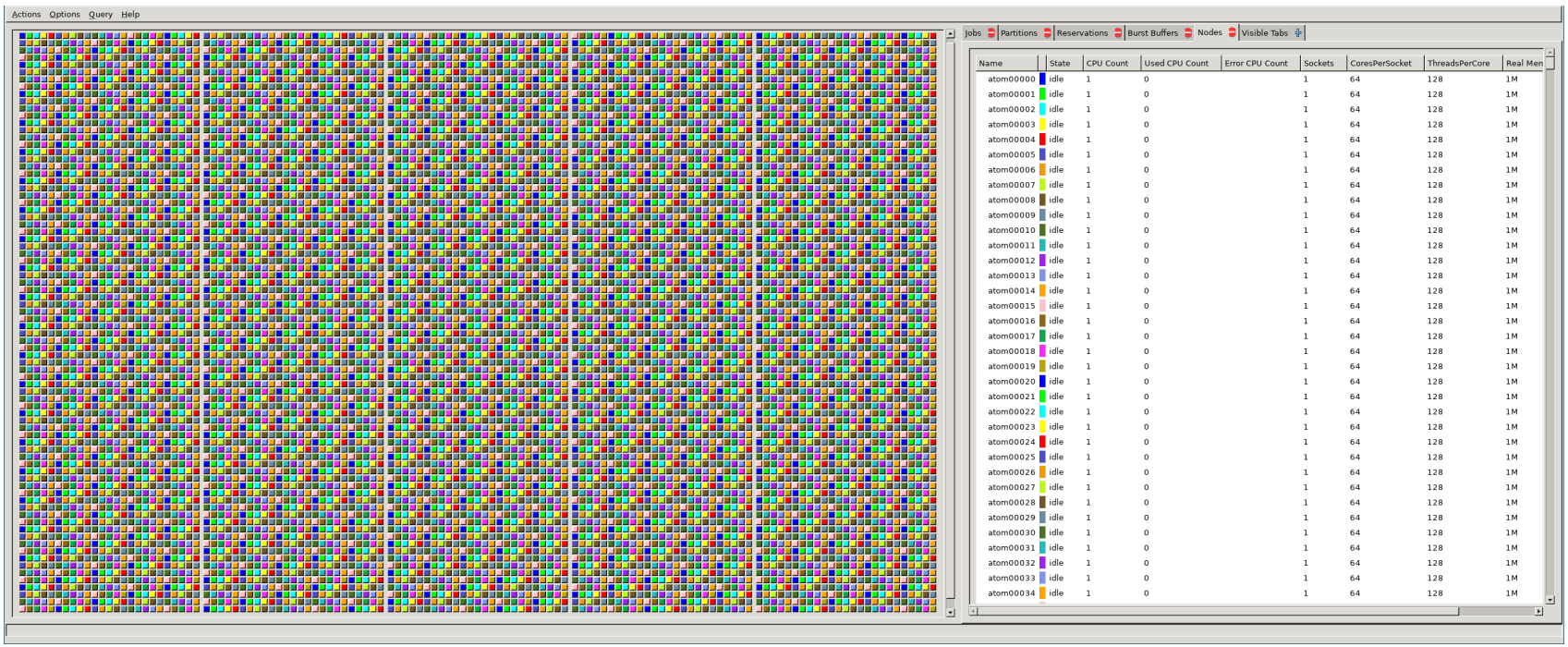
- Give Slurmctld a higher scheduling priority (add `Nice=-20` to `slurmctld.service`)
- Create node template service (Rename `slurmd.service` to `slurmd@.service` and add `-N %i` to slurmd call in `ExecStart`), each node is a service (e.g. `slurmd@atom00001`, etc.)
- Start/stop scripts (complete reset, database wipe)

```
#!/bin/sh

systemctl stop slurmctld
systemctl stop slurmdbd
rm -rf /var/spool/slurm/* /var/log/
     slurm/* /run/slurm/*

systemctl start slurmdbd
sleep 2
systemctl start slurmctld
printf "started slurmdbd, slurmctld\n"
for i in $(seq -f "%05g" 0 9999); do
  systemctl start slurmd@atom$i;
  printf "\rstarted atom $i/09999";
done
printf "\n"
```

```
#!/bin/sh

for i in $(seq -f "%05g" 0 9999); do
  systemctl kill --signal=SIGKILL
     slurmd@atom$i;
  printf "\rstopped atom $i/09999";
done
printf "\n"
systemctl stop slurmctld
printf "stopped slurmctld\n"
systemctl stop slurmdbd
printf "stopped slurmdbd\n"
rm -rf /var/spool/slurm/* /var/log/
     slurm/* /run/slurm/*
printf "DROP DATABASE slurm_acct_db;\n"
      | mysql
```

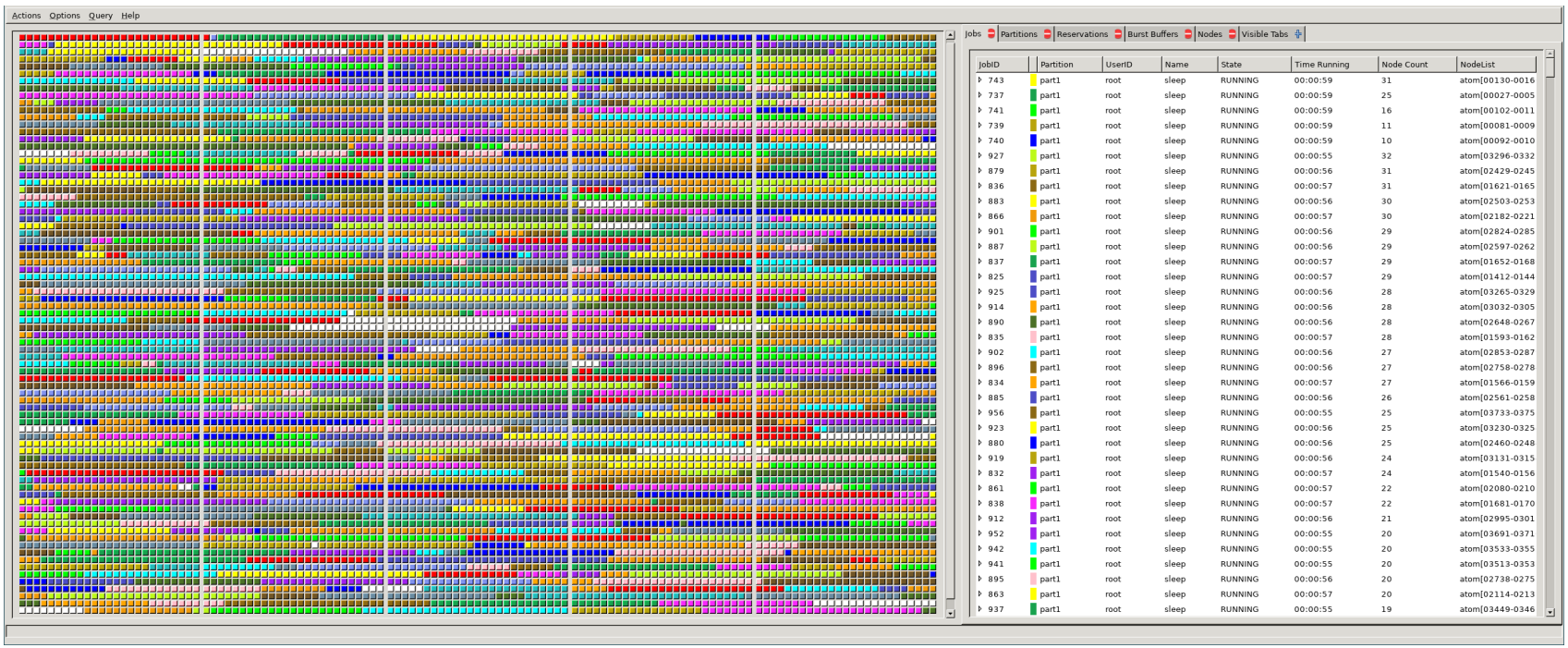
sview(1)

Node Overview



sview(1)

Job Overview



WOGE (Workload Generator and Evaluator)

Overview

Core Functions

- Randomly generate DAG workloads
- Submit workload to Slurm
- Supervise execution and collect data

Implementation Notes

- C99 using POSIX interfaces
- Separate DAG from job parameters
- Parse job completion log for data

WOGE (Workload Generator and Evaluator)

Model Job

- Jobs are sleepers
- Target runtime is randomly jittered
- Batch script

```
#!/bin/sh

#SBATCH --job-name=sleep

sleep $EXECUTION_TIME
```

- Dummy job is extensible (telemetry, prolog/epilog, etc.)
- Control via environment variables

WOGE (Workload Generator and Evaluator)

Workload Generation and Slurm Interface

Workload generation and processing

- `struct dag_parameters`: Number of ranks, range of node count per rank, range of number of dependencies per node
- `dag_generate()`: Generate DAG from parameters
- `struct experiment`: DAG parameters, job parameters
- `experiment_run()`

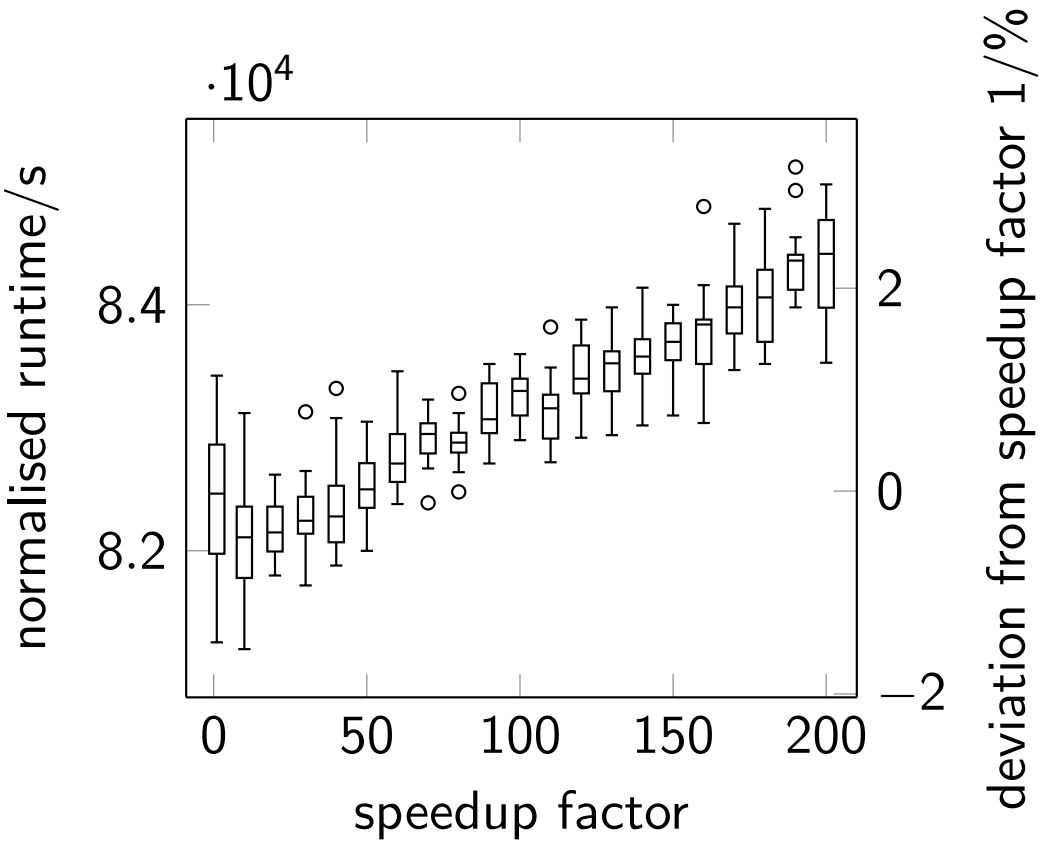
Slurm Interface

- Use Slurm CLI with `execve(2)` (high stability)
- `slurm_clear_job_queue()`
- `slurm_submit_sleeper_job()`: Job is submitted in a held state
- `slurm_release_sleeper_jobs()`: Single call to `scontrol(1)`, no loop

WOGE (Workload Generator and Evaluator)

Benchmark

- Machine: Intel Xeon E5-2637 v2, 4c8t, 192 GiB RAM
- Memory consumption: 16 MiB per virtual node, 160 GiB total
- Evaluation function: Time taken
- Workload: 1 day, speedup $\{1, 10, 20, \ldots, 200\}$, 20 repetitions each



Only around 2-3% deviation despite speedup factor 200

Conclusion

- Fully reproducible Slurm virtual exascale cluster setup, all steps laid out (e.g. student reference)
- WOGE for submitting and evaluating artifical workloads
- Outlook: More intricate random distributions, graph generators (more data needed)
- Long-term: Characterise workloads with a set of parameters and reproduce it artificially this way