LCI workshop
on high performance clustered computing

LCI Advanced Workshop 2025: **Slurm Scheduler Tuning: Balancing HTC & Large Jobs**

Josh Burks
HPC Systems Analyst
Arizona State University Research Computing
josh.burks@asu.edu

This document is a result of work by volunteer LCI instructors and is licensed under CC BY-NC 4.0 (https://creativecommons.org/licenses/by-nc/4.0/

ASU Research Computing

Learning Goals

- Understand the differences between HTC and large-job workloads and their impact on scheduling
- Learn how to identify and mitigate large-job starvation in mixed environments
- Explore scheduler parameter tuning for different cluster sizes
- Compare the roles of the main scheduler vs backfill scheduler
- Explore best practices for partition design and public/private node management
- Gain strategies for iterative tuning and monitoring performance impact

Resource Allocation Models in Slurm

Whole Node Allocation (SelectType=select/linear)

- Job gets entire node regardless of requested resources
- Simple, predictable, no fragmentation
- Wasteful for small jobs

Resource Allocation Models in Slurm

Consumable Resources (SelectType=select/cons_tres)

- Allocates CPUs, GPUs, memory individually
- SelectTypeParameters: CR_CPU, CR_Core, CR_Socket, CR_Memory
- Maximum resource utilization but increased complexity

Scheduling Considerations:

- Job sizing and resource requests
- Node fragmentation
- Memory enforcement (MemEnforceLimit)
- GPU/accelerator placement
- NUMA awareness

Scheduler Types in Slurm

LCI workshop
on high performance clustered computing

SchedulerType=sched/buildin

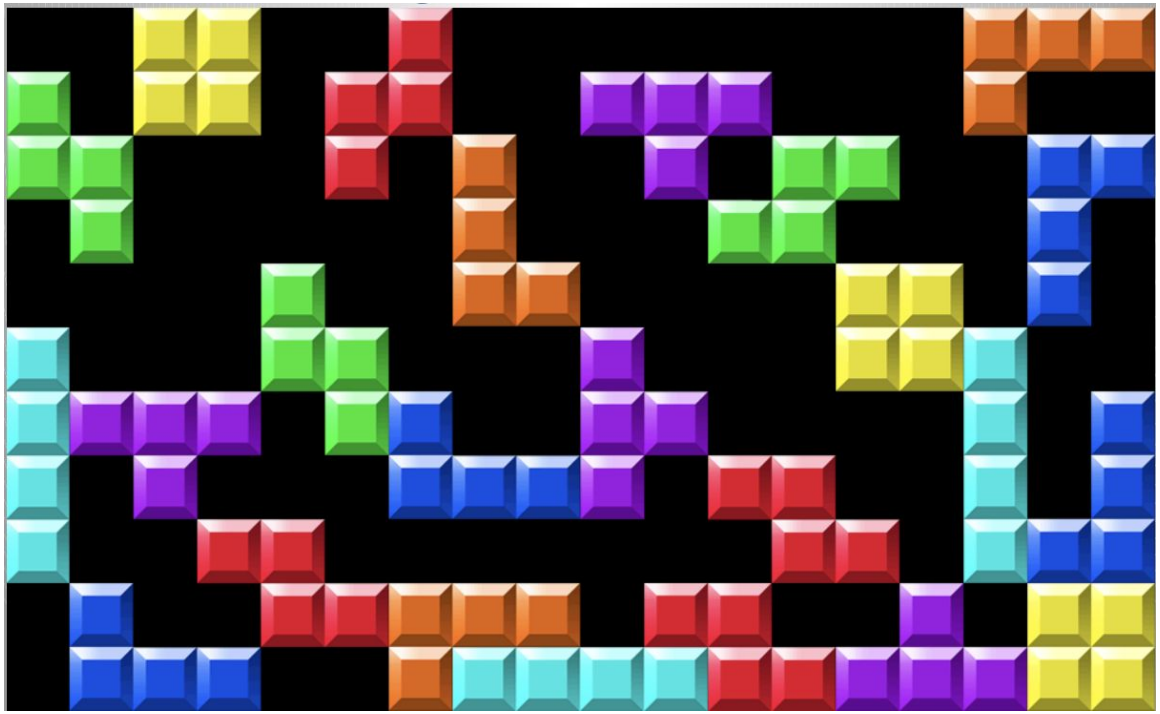
- FIFO scheduler initiates jobs in priority order.
- If any job in the partition can not be scheduled, no lower priority job in that partition will be scheduled.
- exception is made for jobs that can not run due to partition constraints (e.g. the time limit) or down/drained nodes.

SchedulerType=sched/backfill

- Backfill scheduling module to augment the default FIFO scheduling
- Effectiveness dependent upon users specifying job time limits

Main vs Backfill Scheduler Roles

- Main scheduler: determines job start order based on priority
- Backfill scheduler: fills idle resources without delaying higher-priority jobs



Main Scheduler Loop

- Wake up every SchedulerTimeSlice seconds (default: 5)
- Load job queue sorted by priority
- For each job in priority order:
  - Check partition limits (MaxNodes, MaxCPUsPerUser, etc.)
  - Check association/QOS limits
  - Check job dependencies
  - Attempt resource allocation
  - If allocation succeeds: start job or make reservation
  - If allocation fails: continue to next job
- Stop when max_sched_time exceeded OR bf_max_job_test jobs evaluated
- Sleep until next cycle

Backfill Scheduler Loop

- Triggered every bf_interval seconds (default: 30)
- Build "reservation table" from main scheduler decisions
- Calculate earliest start times for all pending jobs
- For each job in queue (not priority order):
  - If job can start now without violating reservations: start it
  - Track jobs started (bf_max_job_start limit)
  - Track jobs tested (bf_max_job_test limit)
- Stop when limits hit or queue exhausted

The Value of Backfill Scheduling

LCI workshop
on high performance clustered computing

Without Backfill

- Strict priority queue processing
- Resources idle while waiting for large jobs
- Poor cluster utilization
- Users frustrated by idle nodes they can't access

With Backfill

- - Smaller jobs fill gaps between large jobs
- - Better user experience (shorter wait times for small jobs)
- - Maintains fairshare and priority goals

Balancing Partitions to Optimize Scheduling

LCI workshop
on high performance clustered computing

Overlapping Resources

- Multiple partitions sharing same nodes
- Scheduler evaluates each partition separately
- Duplicate effort checking same resources
- Can cause scheduler performance issues

Minimize partition overlap where possible

- Use AllowAccounts/AllowGroups instead of separate partitions
- Partition by job characteristics (size, runtime, priority)
- Monitor sdiag for excessive partition evaluation times

Using sdiag to Diagnose Scheduler Issues

```
# sdiag
Main schedule statistics
(microseconds):
        Last cycle:   2302
        Max cycle:    62634
        Total cycles: 26014
        Mean cycle:   3323
        Mean depth cycle:  69
        Cycles per minute: 57
        Last queue length: 68
```

```
Main scheduler exit:
        End of job queue:26012
        Hit default_queue_depth: 2
        Hit sched_max_job_start: 0
        Blocked on licenses: 0
        Hit max_rpc_cnt: 0
        Timeout (max_sched_time): 0
```

Using sdiag to Diagnose Scheduler Issues

```
# sdiag
Backfilling stats
        Total backfilled jobs: 243060
        Total cycles: 454
        Last cycle: 609383
        Max cycle:  853384
        Mean cycle: 294402
        Last depth cycle: 130
        Last depth cycle (try sched): 130
        Depth Mean: 189
        Depth Mean (try depth): 188
Backfill exit
        End of job queue:454
        Hit bf_max_job_start: 0
        Hit bf_max_job_test: 0
        System state changed: 0
        Hit table size limit: 0
        Timeout (bf_max_time): 0
```

Tuning Main Scheduler Cycle Timing

- **max_sched_time** - Max time per main cycle (default: 2s), prevents scheduler from blocking too long

- **sched_interval** - How often the main scheduling loop will run, considering all jobs while still honoring the partition_job_depth limit

- **default_queue_depth -** Number of jobs to consider for scheduling on each event that may result in a job being scheduled. (Default 100)

Tuning Backfill Scheduler Cycle Timing

bf_interval

- How often backfill runs (default: 30s)
- Independent of main scheduler cycle
- Reduce for fast-moving HTC workloads
- Increase to reduce overhead on large systems

Tuning Backfill Scheduler Cycle Timing

bf_window

- Jobs needing resources beyond window ignored
- Increase for long jobs
- A value at least as long as the highest allowed time limit is generally advisable to prevent job starvation

bf_window_linear

- Setting this will create higher resolution of time precision in the backfill scheidler uses linear backoff instead of exponential back off
- Not recommended with more than a few hundred simultaneously executing jobs.

Tuning Backfill Scheduler Cycle Timing

bf_max_job_test

- Max jobs evaluated per backfill cycle (default: 300)
- Increase for deeper backfill
- Watch cycle times - can make cycles slow

bf_max_job_start

- Max jobs started per backfill cycle (default: 0 = unlimited)
- Limit to prevent backfill from being too aggressive
- Useful for protecting large job opportunities

bf_max_job_assoc

- How many jobs per user associona backfill can check
- Large impact when users submit many jobs

Tuning for All HTC Workloads

LCI workshop
on high performance clustered computing

defer - Do not attempt to schedule jobs individually at submit time. Can be useful for high-throughput computing.

defer_batch - Like **defer**, but only will defer scheduling for batch jobs.

pack_serial_at_end - may reduce resource fragmentation by put serial jobs at the end of the available nodes

SelectTypeParameters=CR_LLN - Schedule resources to jobs on the least loaded nodes

Workload Mix & Scheduling Pressure

HTC Workloads Want:

- Fast turnaround (seconds to minutes)
- High throughput (jobs/second)
- Frequent backfill opportunities
- Minimal wait times

Large Jobs Want:

- Resource guarantees
- Protected start times
- Priority enforcement
- Minimal fragmentation

Tuning for All HTC

LCI workshop
on high performance clustered computing

```
# Fast, frequent cycles
SchedulerTimeSlice=2
SchedulerParameters=max_sched_time=1

# Aggressive backfill
SchedulerParameters=bf_interval=15
SchedulerParameters=bf_window=120  # 2 hours
SchedulerParameters=bf_max_job_test=1000
SchedulerParameters=bf_max_job_start=0 # unlimited
SchedulerParameters=bf_continue
PriorityFavorSmall=Yes
```

Tuning for All Large Jobs

LCI workshop
on high performance clustered computing

```
# Standard cycles, deeper evaluation
SchedulerTimeSlice=5
SchedulerParameters=max_sched_time=4

# Conservative backfill
SchedulerParameters=bf_interval=60
SchedulerParameters=bf_window=10080  # 7 days
SchedulerParameters=bf_max_job_test=300
SchedulerParameters=bf_max_job_start=5
SchedulerParameters=bf_reserve_nodes=1
```

Workload Mix & Scheduling Pressure

LCI workshop
on high performance clustered computing

The Conflict:

- HTC jobs fill all available resources
- Large jobs can't find contiguous allocations
- Priority system gets subverted by constant backfill
- Users of both types dissatisfied

Solution Requires careful balance of main and backfill tuning, partition design, and policy enforcement

Tuning for Hybrid Workloads

LCI workshop
on high performance clustered computing

```
# Moderate cycle timing
SchedulerTimeSlice=5
SchedulerParameters=max_sched_time=2

# Tuned backfill
SchedulerParameters=bf_interval=30
SchedulerParameters=bf_window=1440  # 24 hours
SchedulerParameters=bf_max_job_test=500
SchedulerParameters=bf_max_job_assoc=20 # CRITICAL
```

Scheduler Tuning at ASU

```
SchedulerParameters     =
bf_window=43200
bf_resolution=300
bf_max_time=120
bf_max_job_assoc=25
bf_max_job_test=150000
bf_interval=30
max_rpc_cnt=1000
enable_user_top
bf_continue
nohold_on_prolog_fail
```

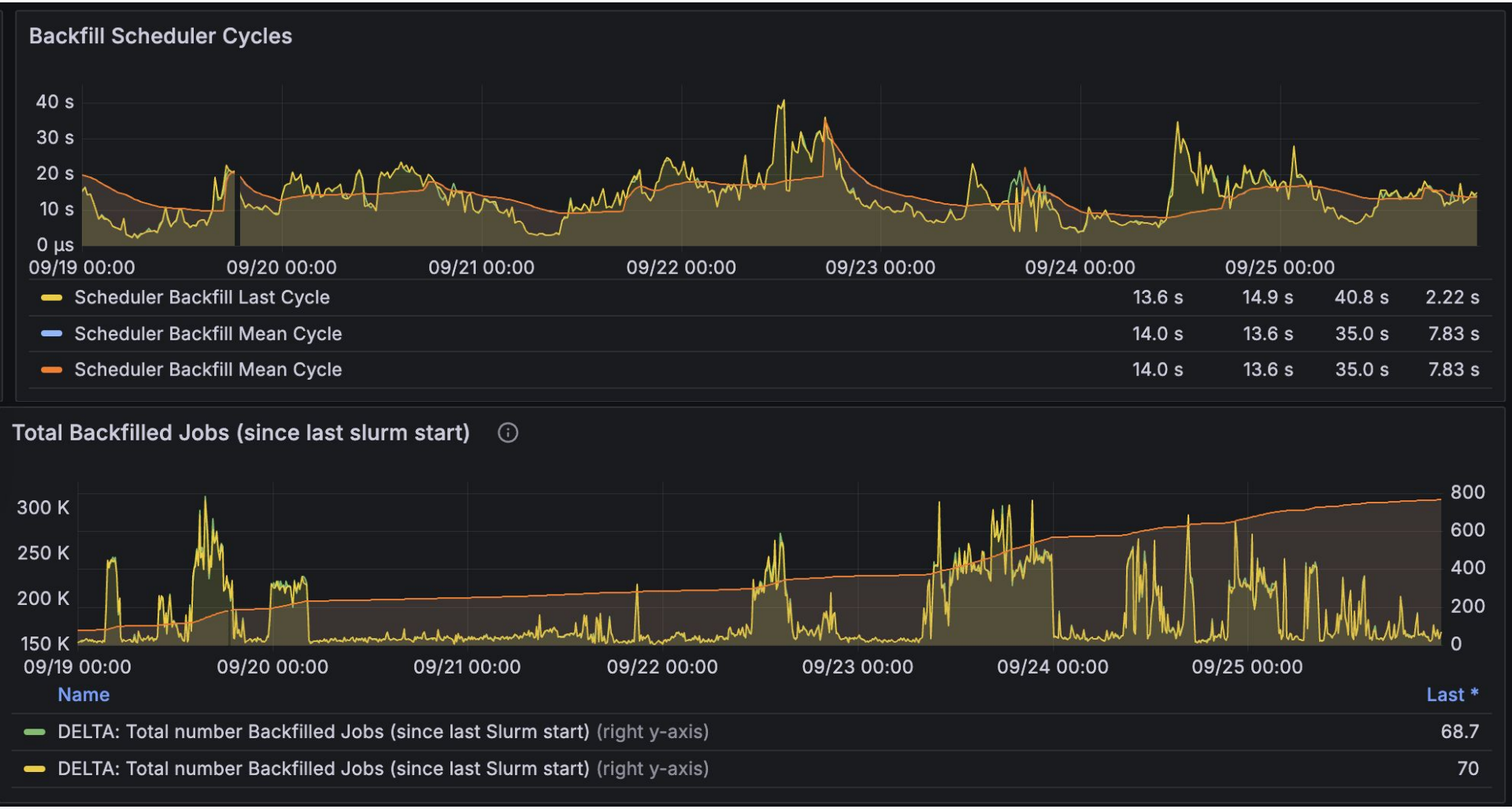
Monitoring & Iteration

- Baseline before changes
- Metrics: wait times, utilization, fairshare
- Gradual adjustments
- User feedback loop
- Slurm Exporter:
    - github.com/vpenso/prometheus-slurm-exporter
    - github.com/ubccr/slurm-exporter (Uses Slurm API)
    - gitlab.com/SchedMD/slinky/slurm-exporter (Official)

Monitoring & Iteration

LCI workshop
on high performance clustered computing

grafana.com/dashboards/4323



ASU Research Computing

Runaway Jobs

LCI workshop
on high performance clustered computing

```
# sacctmgr show runawayjobs
```

NOTE: Runaway jobs are jobs that don't exist in the controller but have a start time and no end time in the database

```
[root@sol-slurm01:~]# sacctmgr show runawayjobs
NOTE: Runaway jobs are jobs that don't exist in the controller but have a start time and no end time in the database
ID                Name  Partition    Cluster      State          TimeSubmit           TimeStart             TimeEnd
----------- ---------- ---------- ---------- ---------- ------------------- ------------------- -------------------
28108882     /home/jeb+  lightwork        sol    PENDING 2025-07-02T00:05:17             Unknown             Unknown
35866256     ood-vscod+    general        sol    PENDING 2025-10-14T09:49:36             Unknown             Unknown
35866258     ood-virtu+    general        sol    PENDING 2025-10-14T09:49:46             Unknown             Unknown
35866283     interacti+        htc        sol    PENDING 2025-10-14T10:39:40             Unknown             Unknown

Would you like to fix these runaway jobs?
(This sets the end time for each job to the latest of the job's start, eligible, and submit times, and sets the state
 to completed.
Once corrected, this triggers the SlurmDBD to recalculate the usage from before the earliest submit time of all the r
unaway jobs. Warning: This could take a long time and sreport may not return data until the recalculation is complete
d.)

 (You have 30 seconds to decide)
```

ASU Research Computing

Key Takeaways

- Match tuning to workload type and site goals
- Use partitions and limits strategically
- Balance main scheduler and backfill tuning
- Iterate with data and feedback

Q & A

LCI workshop
on high performance clustered computing