


**Barcelona
Supercomputing
Center**
Centro Nacional de Supercomputación

A central graphic showing a classical building with a pediment and columns, overlaid with the text "SLURM SIMULATOR" in a bold, italicized, black sans-serif font.

***SLURM
SIMULATOR***

Phoenix, September
2011

Alejandro Lucero
Alejandro.lucero@bsc.es

SLURM SIMULATOR



1. Introduction: Why Slurm Simulator?
2. Design & Implementation
3. Results
4. Use Examples
5. Future Work

SLURM SIMULATOR



1. **Introduction: Why Slurm Simulator?**
2. Design & Implementation
3. Results
4. Use Examples
5. Future Work

SLURM SIMULATOR: INTRODUCTION



- I work as a performance engineer at Barcelona SuperComputing Center (BSC), Spain
- We have 5 HPC cluster with Slurm, the big (and old) one with 2500 nodes
- Expecting a Petaflop machine before year's end
- BSC leader of RES (Supercomputers Network of Spain) with 7 HPC clusters (technology centers)

SLURM SIMULATOR: INTRODUCTION



SLURM SIMULATOR: WHY SLURM SIMULATOR?

- We use Moab with Slurm for batch scheduling in most of the clusters
- Scheduling configuration/tuning depends on parameters like: fair sharing tree, qos/user/group limits, backfilling interval/chunk, reservations
- Moab supports some sort of simulation mode (Moab manual says that). Using real job submission traces under simulation can tell us which configuration would be better

SLURM SIMULATOR: WHY SLURM SIMULATOR?

- I could not make this simulation mode working under Moab (Adaptive Computing “this is not supported any more...”)
- Uhhmm ... Why not to have such a mode with Slurm?
- Last Slurm meeting in Paris we presented a first proposal and no one said we were mad so ...

SLURM SIMULATOR



1. Introduction: Why Slurm Simulator?
2. **Design & Implementation**
3. Results
4. Use Examples
5. Future Work



Some things about Slurm:

- Slurm is a multithread and distributed software
- Two main components: slurmctl and slurmd
- Code optimized for starting/signaling jobs through hundred or thousands of node
- Agent threads for communications



- Goal: Minimum changes to slurm core
- A new program, *sim_mgr* will take the control of Slurm execution and maintain simulation time domain
- LD_PRELOAD will be used catching slurmctl/slurmd time-related functions (time, sleep, gettimeofday) inside a library *sim_lib*
- Shared memory will be used for global simulation time
- Just one slurmd will be needed but no jobs executed



- Capturing time-related calls with LD_PRELOAD is easy and simple with a single thread program but...
- With Multithread & Distributed Slurm:
 - Thread-related calls need to be captured as well inside sim_lib
 - A per-thread structure in the shared memory keeping thread ID (simulation domain), sleep seconds field, semaphores
 - Semaphores used for simulation control: just one thread executed concurrently*



- Slurm threads from simulator point of view:
 - Those being executed through the full simulation / execution with a periodic cycle using sleep call (created when slurm is initialized during first simulation cycles)
 - Those created and finished during a simulation cycle: no sleep calls made (created with new events: job submission, job dispatched, job finished)
 - Some special ones like rpc threads which need to execute at any time without sim_mgr control for avoiding deadlocks. Those live thorough the whole simulation but no sleep call made.



- A thread from `slurmctld` or `slurmd` will go through:
 - (1) `pthread_create` is captured and a thread is registered inside `sim_mgr` thread array getting an unique ID
 - (2) Threads code executes
 - (3) If this is a periodic thread, `sleep` call is captured by `sim_lib` and thread waits on a semaphore
 - (4) Threads call `pthread_exit` which releases slot in `sim_mgr` array



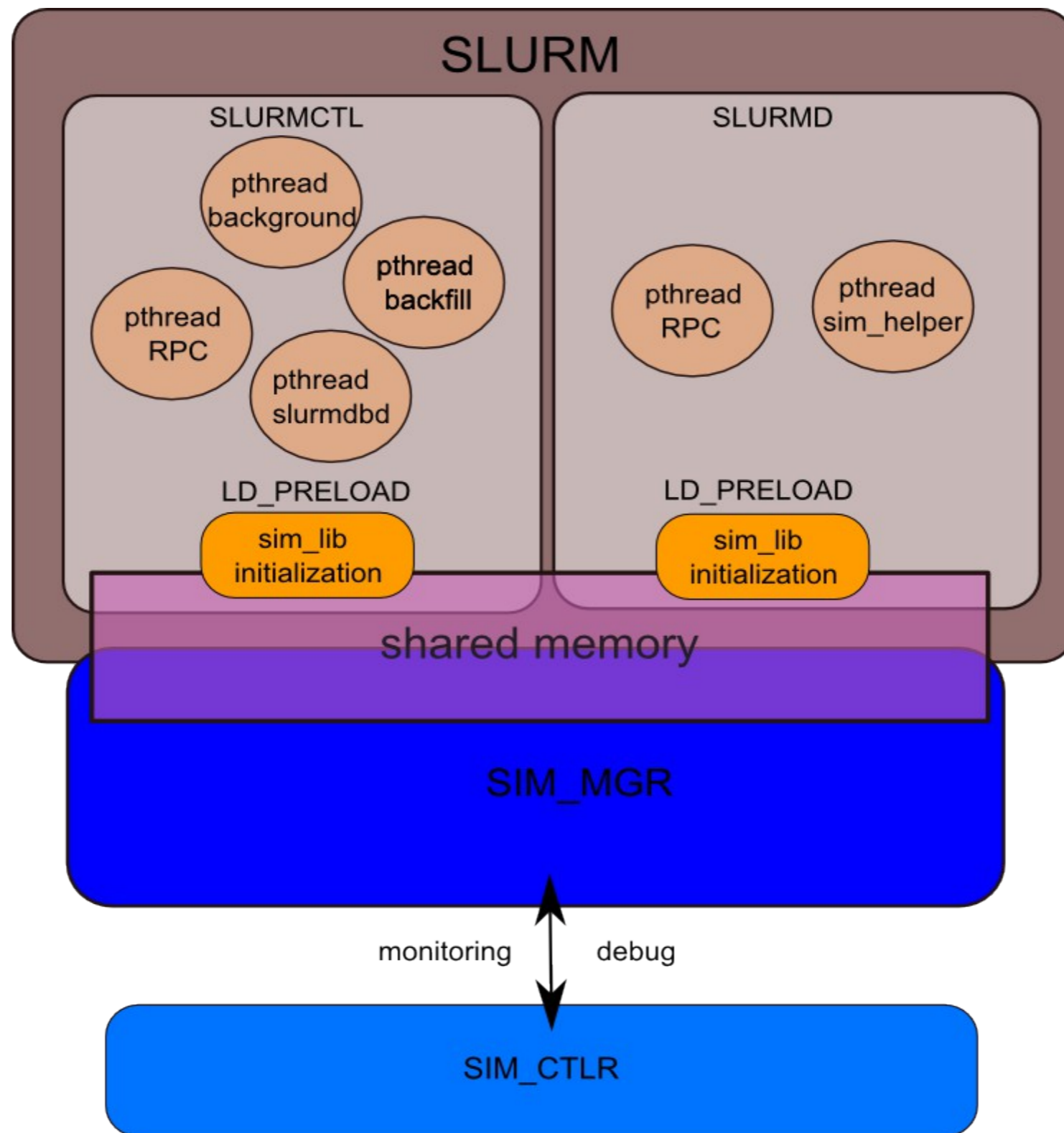
sim_mgr

- A simulation cycle represent a real second
- Each cycle:
 - (1) Goes through all threads registered and leaves them to execute if no sleeping, just one at a time (determinism)
 - (2) Looks for new events from trace file: new job, new reservation
 - (3) Checks for new threads created during this cycle (this is done in several places) and waits till all exit.
 - (4) Increments simulation time



- Slurm simulator should be transparent for slurm core developers
- Two pieces of software external to slurm core: sim_mgr and sim_lib
- Semaphores and shared memory created and initialized by sim_mgr

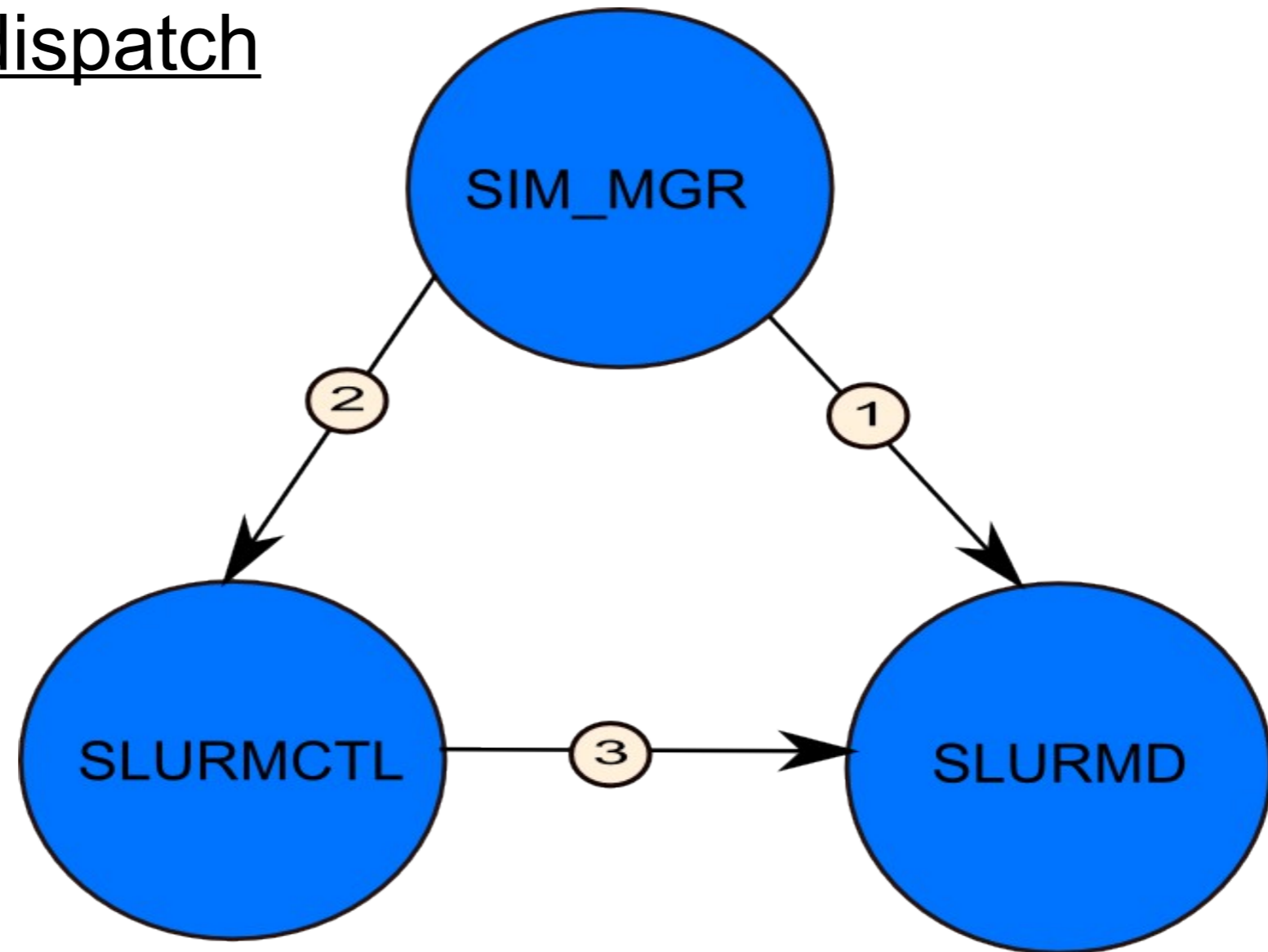
SLURM SIMULATOR: IMPLEMENTATION



Slurm code changes

- Threads need to call `pthread_exit` explicitly
- Jobs and nodes Monitoring is not activated at `slurmctld`
- Agents are avoided: job start message is part of thread doing the schedule (deadlock, determinism)
- `Slurmd` accepts messages from `sim_mgr` for getting information about job duration. A new thread controls when a job finishes for sending message to `slurmctld`

Job submission/dispatch



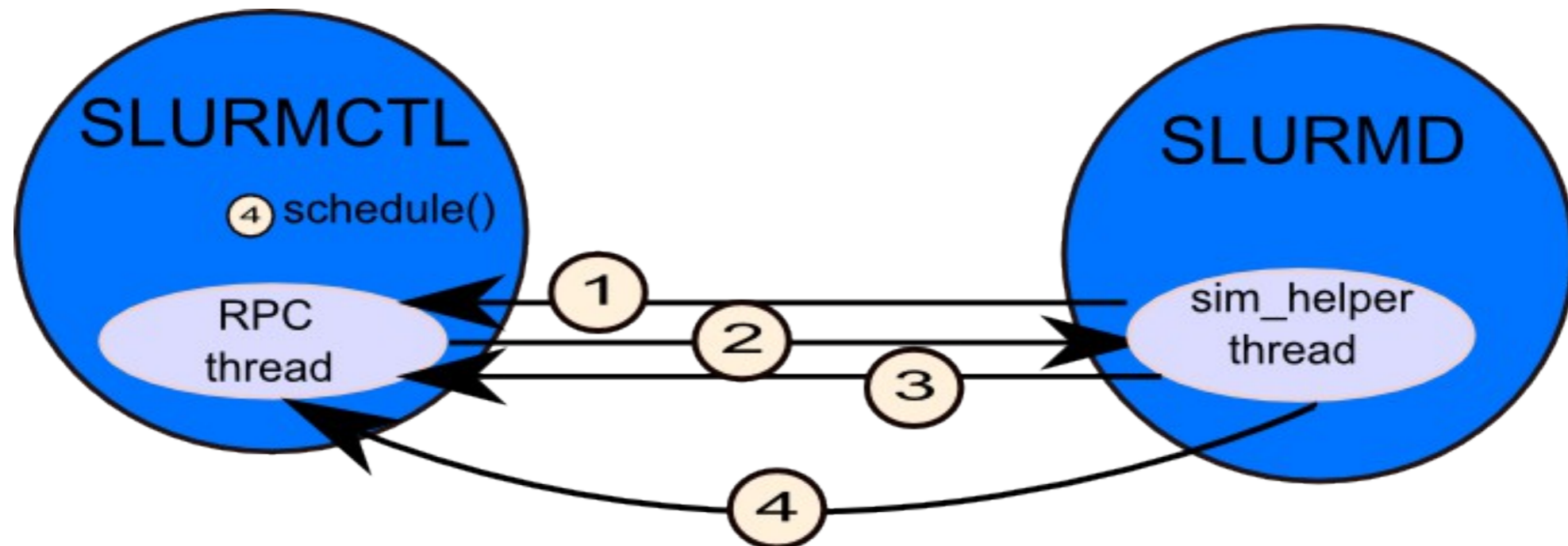
- ① REQUEST_SIM_MGR (jobid, duration)
- ② REQUEST_SUBMIT_BATCH_JOB
- ③ REQUEST_BATCH_JOB_LAUNCH



Job completion

- Sim_helper thread (slurmd) checks for jobs finishing in each simulation cycle
- Usual COMPLETE_BATCH_SCRIPT and EPILOG_COMPLETE sent to slurmctld
- *EPILOG_COMPLETE* message does NOT give rise to schedule in slurmctld. A single call to schedule instead when all epilog messages are sent using a special message from slurmd to slurmctld (determinism)

Job completion



- ① JOB_COMPLETE_BATCH_SCRIPT
- ② TERMINATE_JOB
- ③ EPILOG_COMPLETE
- ④ SIM_HELPER_CYCLE_MSG

Slurm code changes

- Backfilling thread execution time by scheduling cycle is dependent on number of jobs waiting. It can take long even tuning bf depth
- Backfilling algorithm checks if execution time exceeds a configured limit (`sched_timeout = 5 seconds` by default). If so it goes to sleep for `backfilling_interval` seconds.
- It keeps going from same point inside backfilling algorithm except if some update to any job, node or partition.
- Under simulation it is not possible to check “real” execution time easily. A loop counter for processed jobs



Simulator Monitor

- Stopping simulation on a specific point in time
- It can be periodic for getting data
- When simulation stopped some slurm commands can still be executed like:

```
scontrol setdebug 9
```

SLURM SIMULATOR



1. Introduction: Why Slurm Simulator?
2. Design & Implementation
3. **Results**
4. Use Examples
5. Future Work



- Simulator initial work on slurm-2.1.9
- First porting to slurm-2.2.6 really fast and easy
- Lines changed:
 - 563 added
 - 17 removed
 - Plus ~2000 lines `sim_mgr.c` and `sim_lib.c`

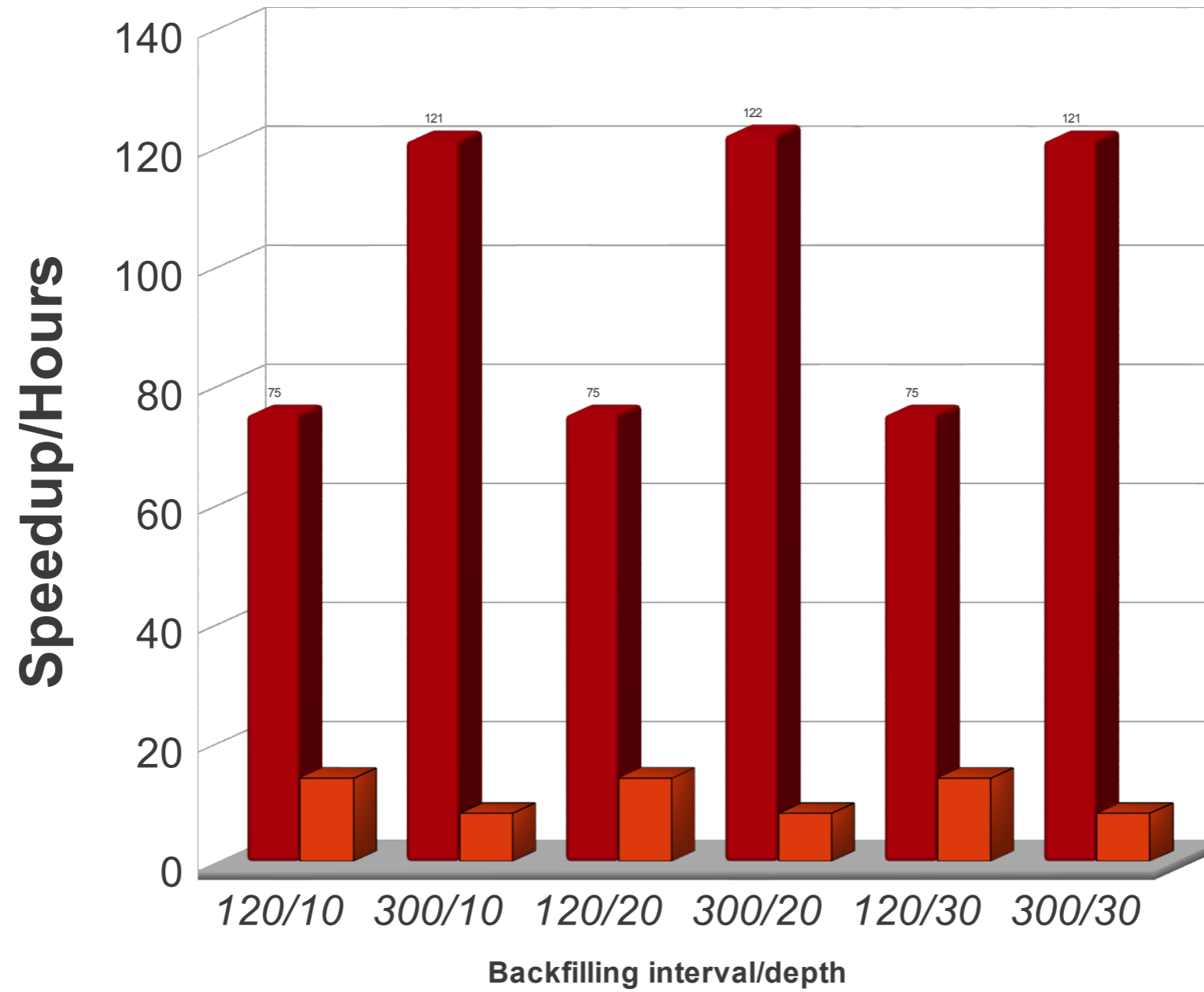


Performance

- Intel Xeon 2.5Ghz, 8 cores, 12GBytes of memory
- Using a real two month trace from Marenostrum (~50000 jobs, 489 users, 19 qos, 15 accounts)
- Marenostrum: 2500 nodes, 4 cores by node
- Using slurmdbd with fair sharing (no limits) NOT under simulation control
- Using backfilling limiting loops to 20 (hardcoded)

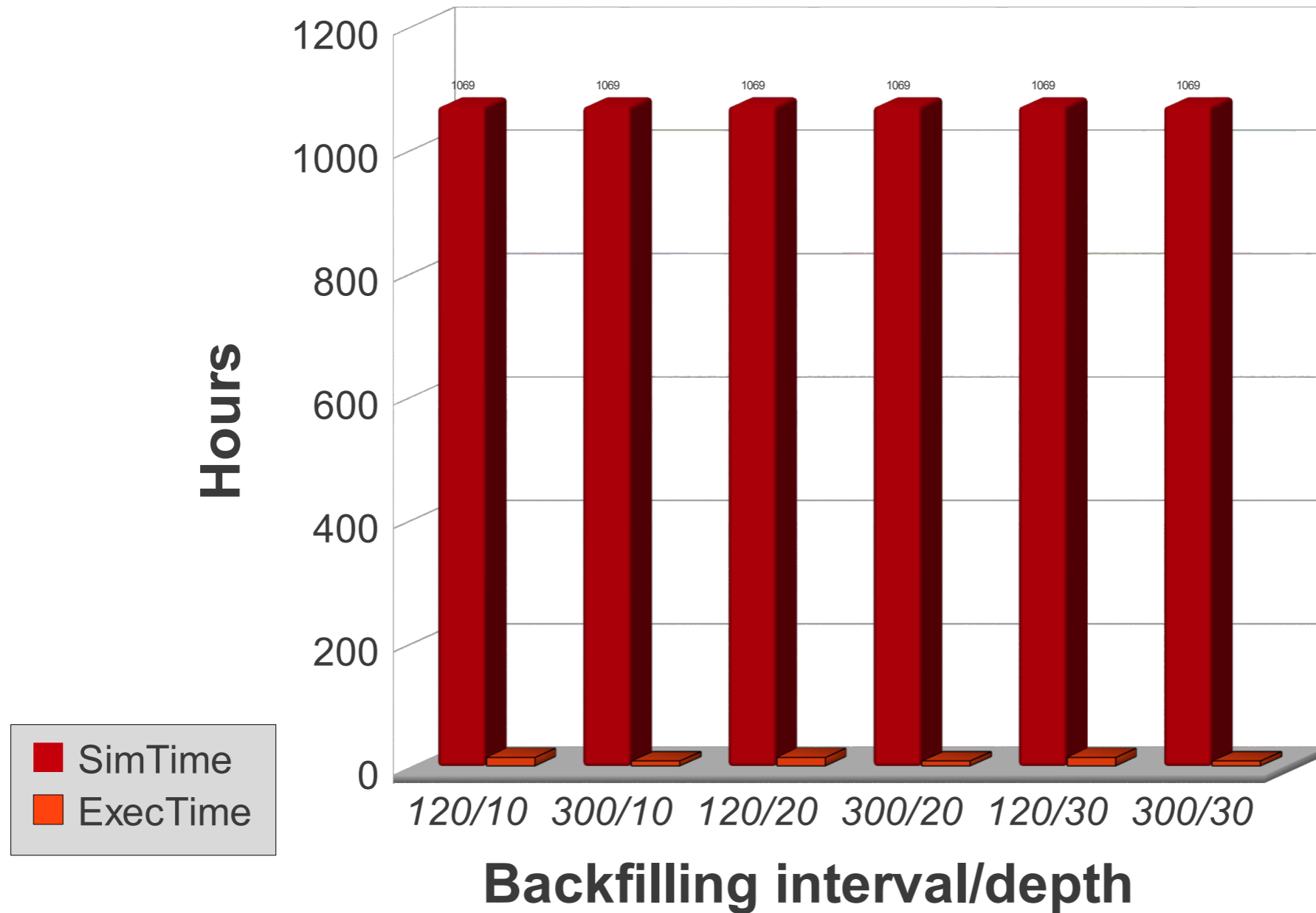


Performance (backfilling dependent)



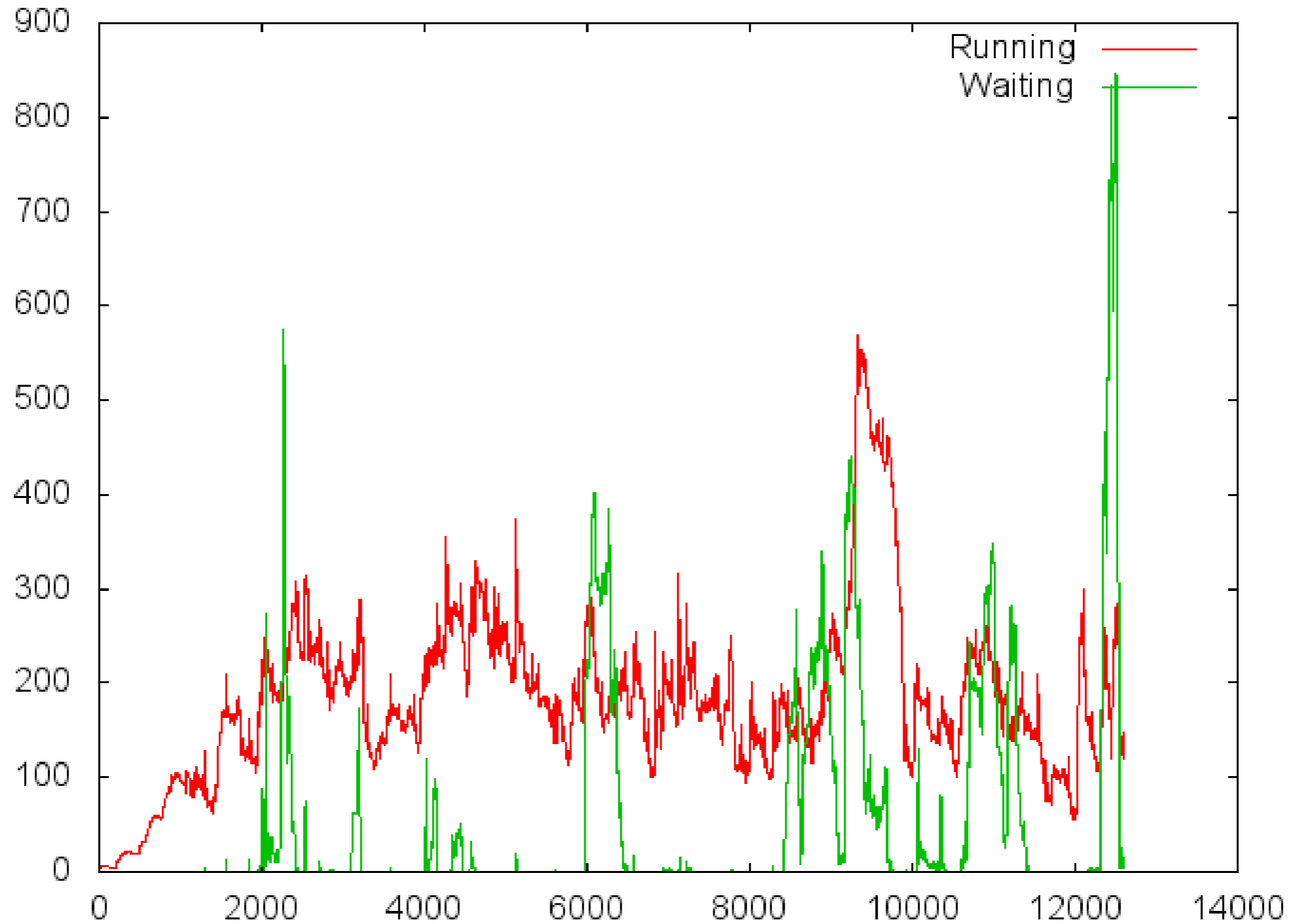


Performance (backfilling dependent)





Running/Waiting jobs (sim_ctrl)



SLURM SIMULATOR



1. Introduction: Why Slurm Simulator?
2. Design
3. Implementation
4. Results
5. **Use Examples**
6. Future Work



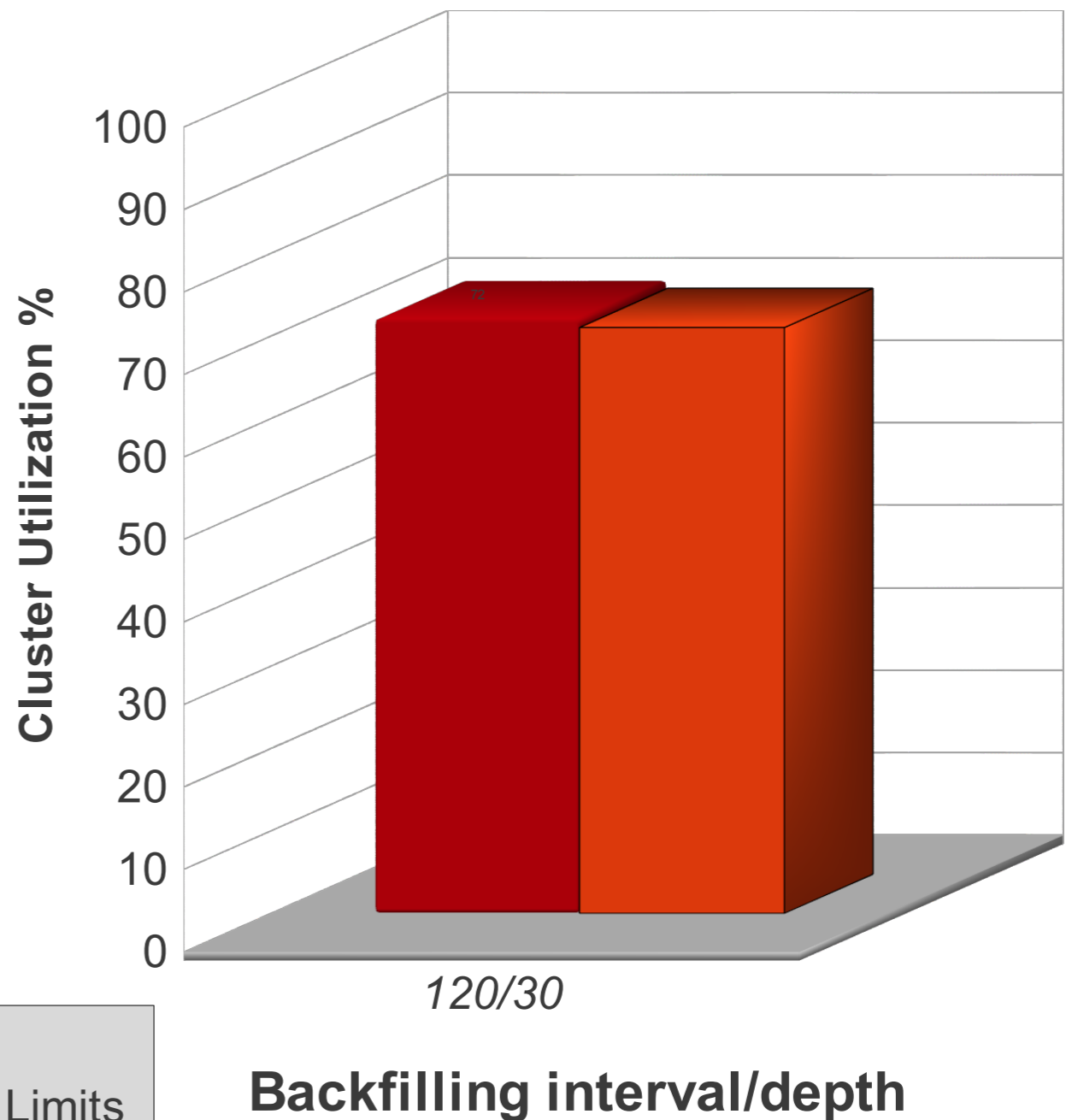
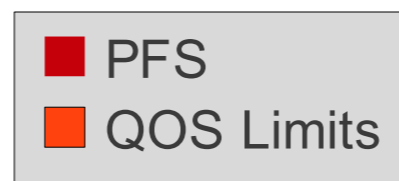
Configuration Tuning

- Our current Moab/Slurm configuration is based on fair sharing and qos limits
- Sometimes machine has idle nodes but limits avoid a better usage
- Let's do simulation using a pure fair sharing versus current MN configuration (Moab) using qos limits with fair sharing



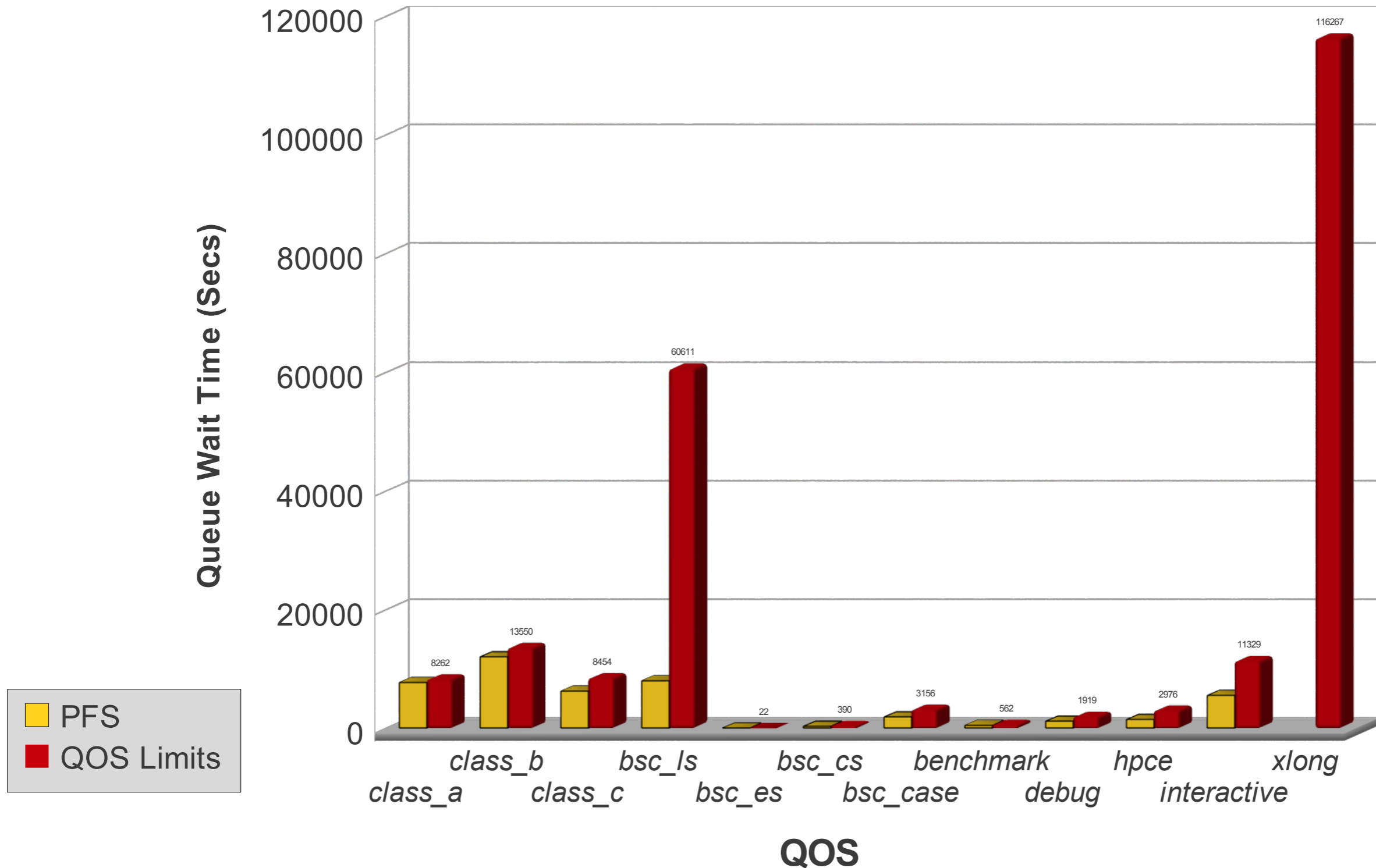
Pure Fair sharing VS QOS limits

- Utilization: 72% versus 71%
- 1% (2546 nodes * 4 cores * ~44 days)
= 391949062 cpu hours
= 10 hours per core
- Waittime?



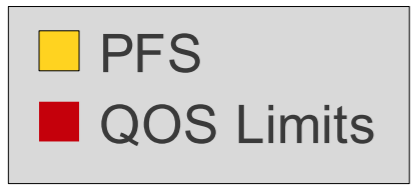
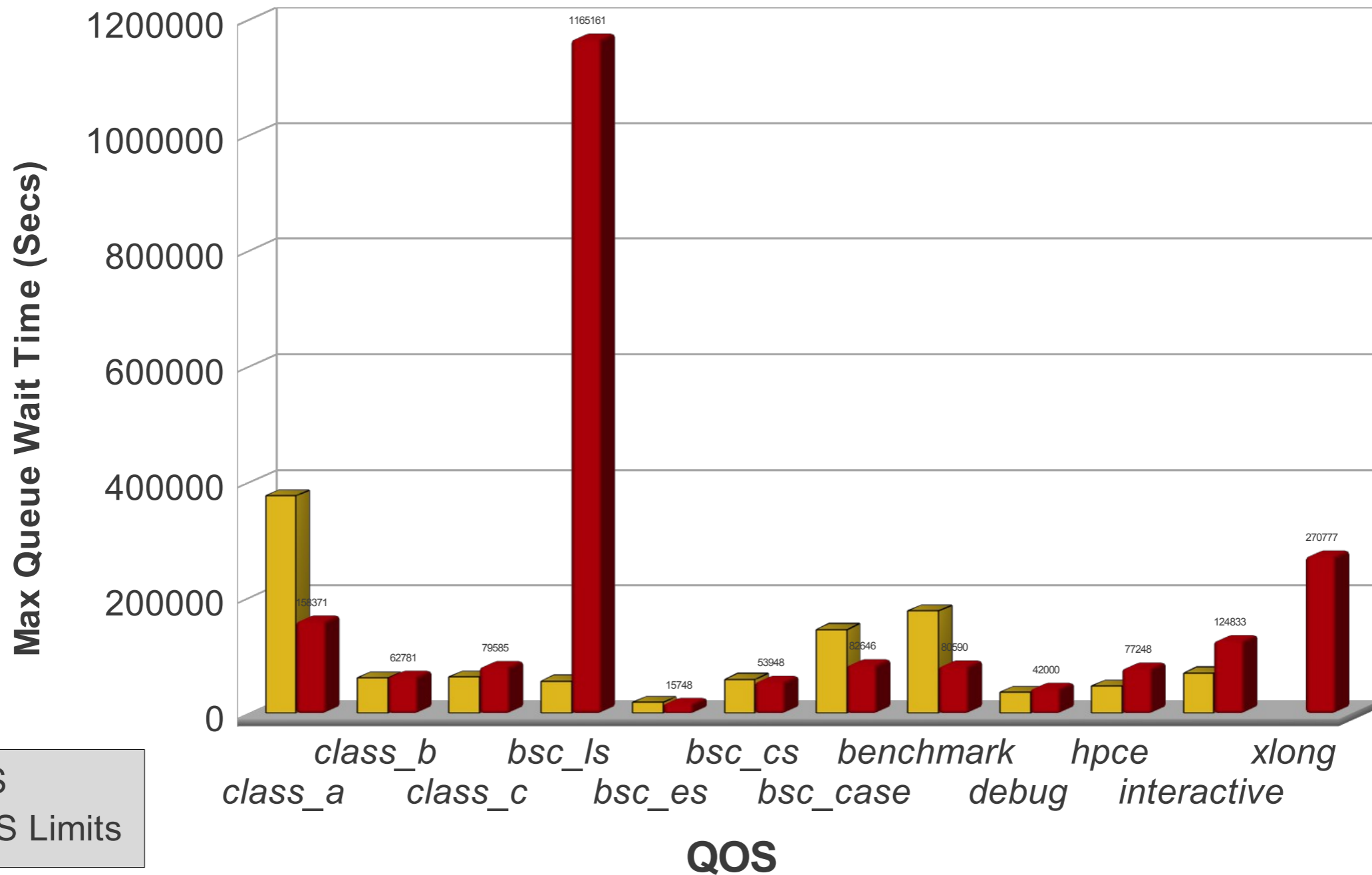


Pure Fair sharing VS QOS limits

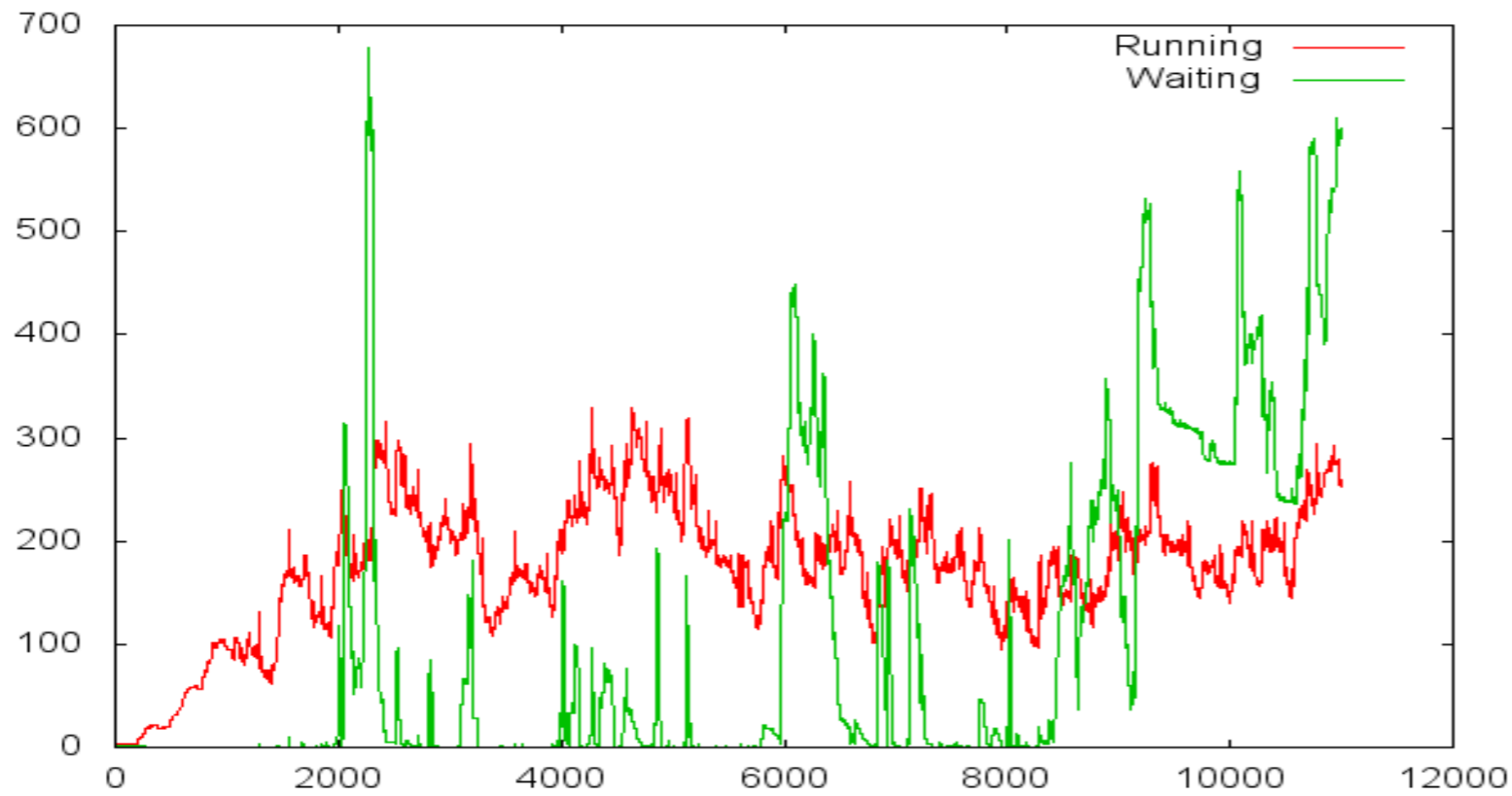




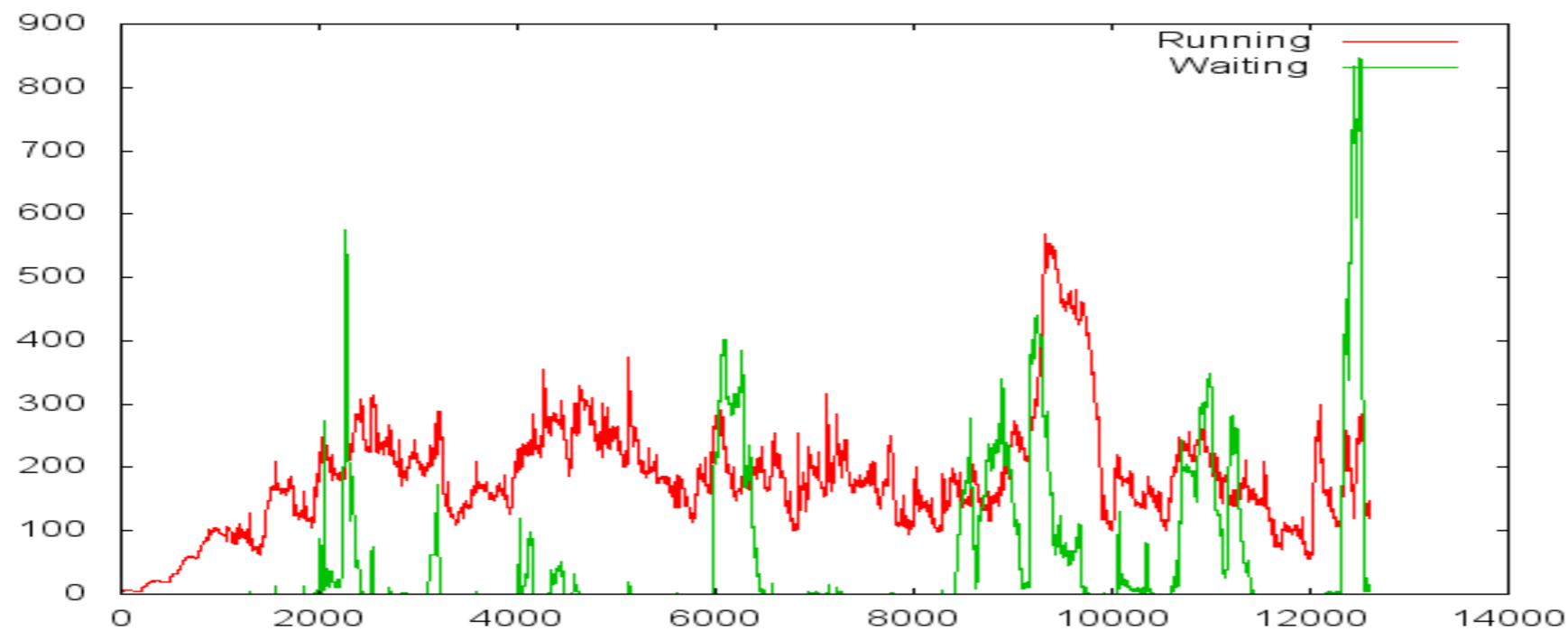
Pure Fair sharing VS QOS limits



SLURM SIMULATOR: RESULTS

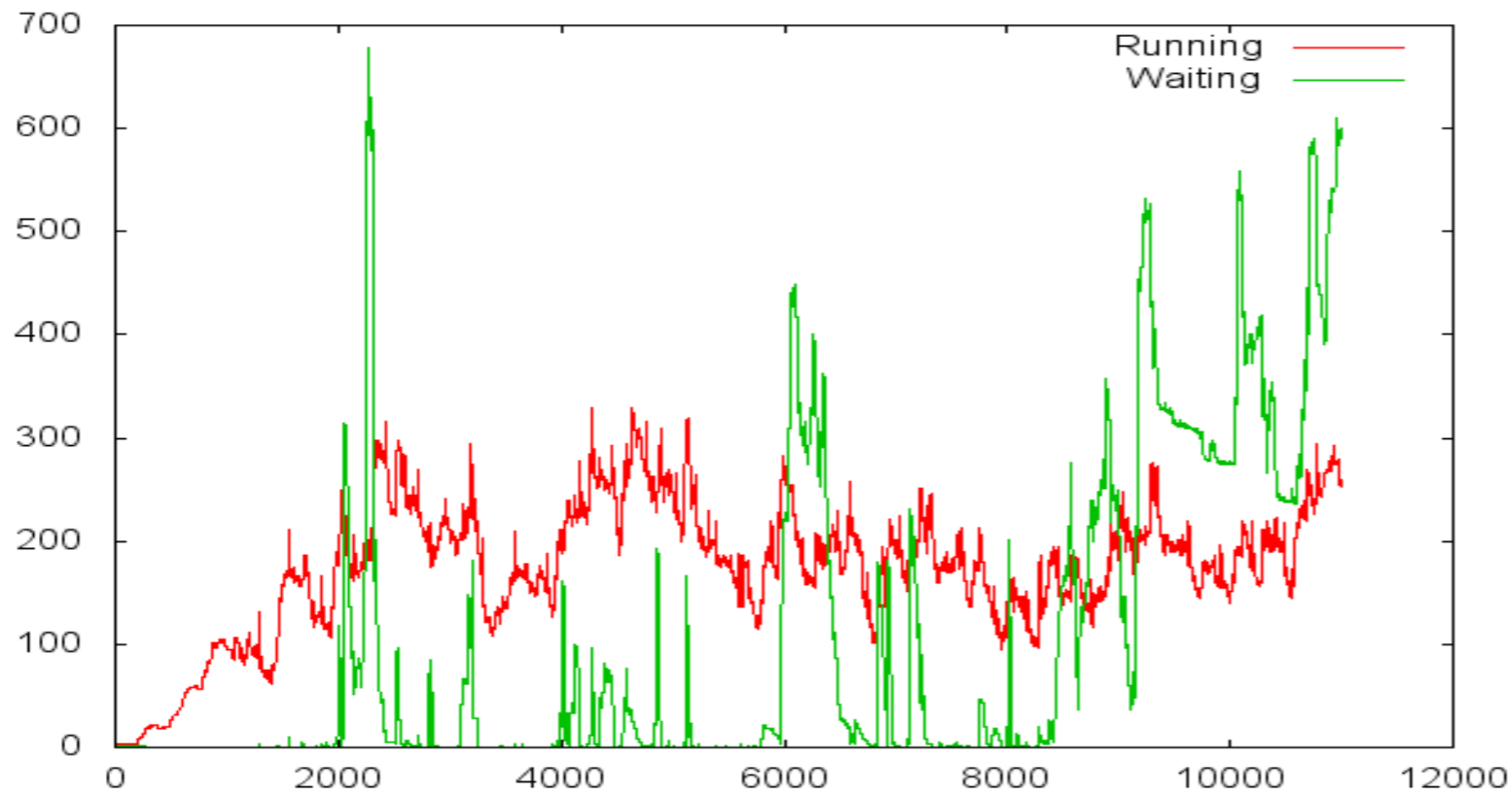


QOS Limits

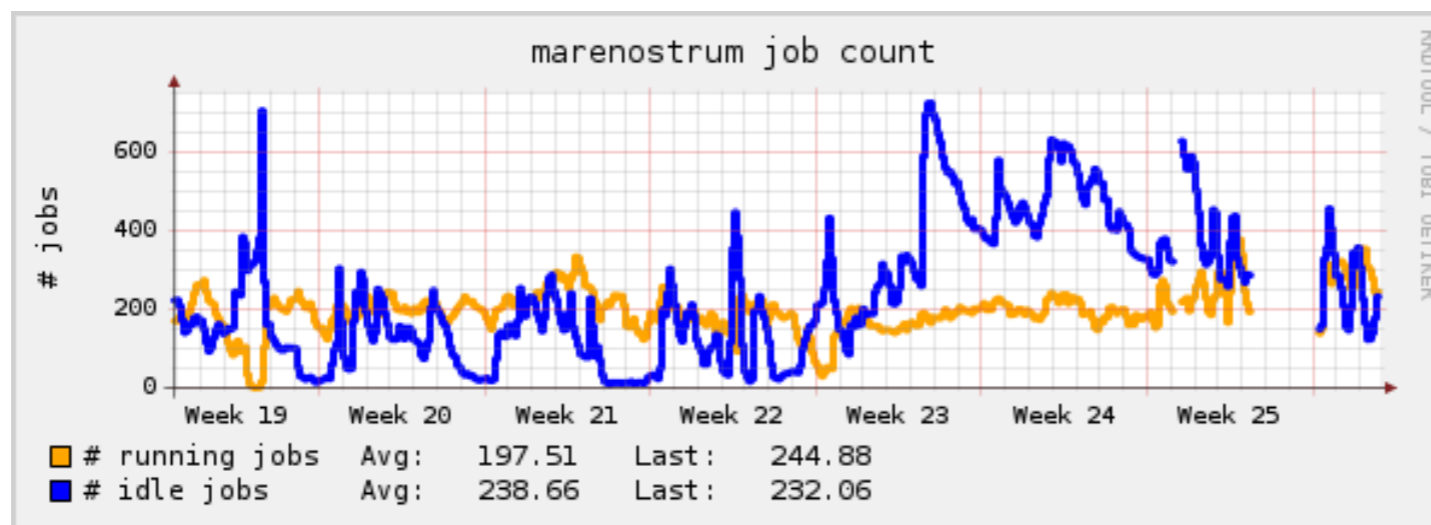


Pure Fair
Sharing

SLURM SIMULATOR: RESULTS



QOS Limits



Real MN
execution

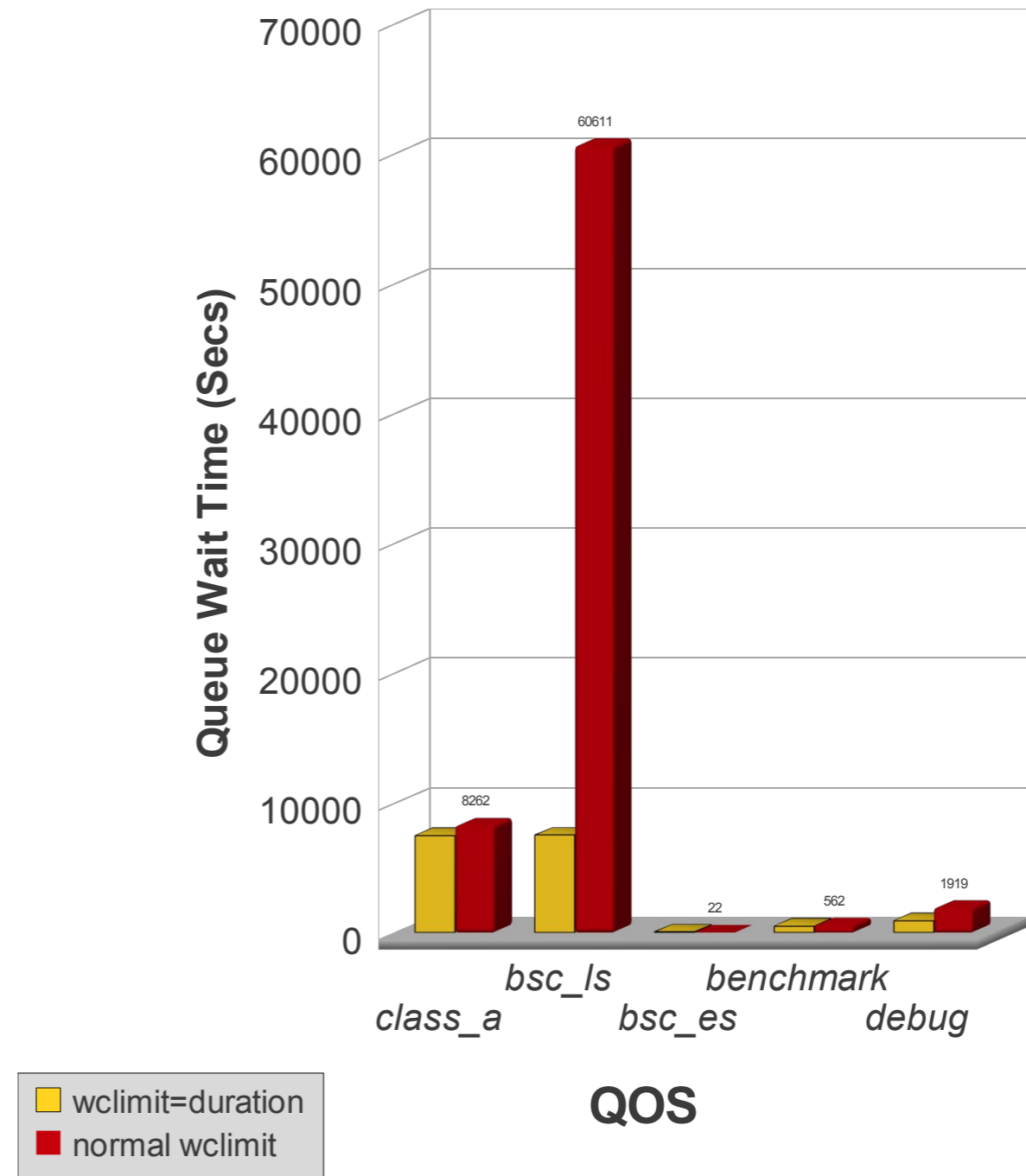
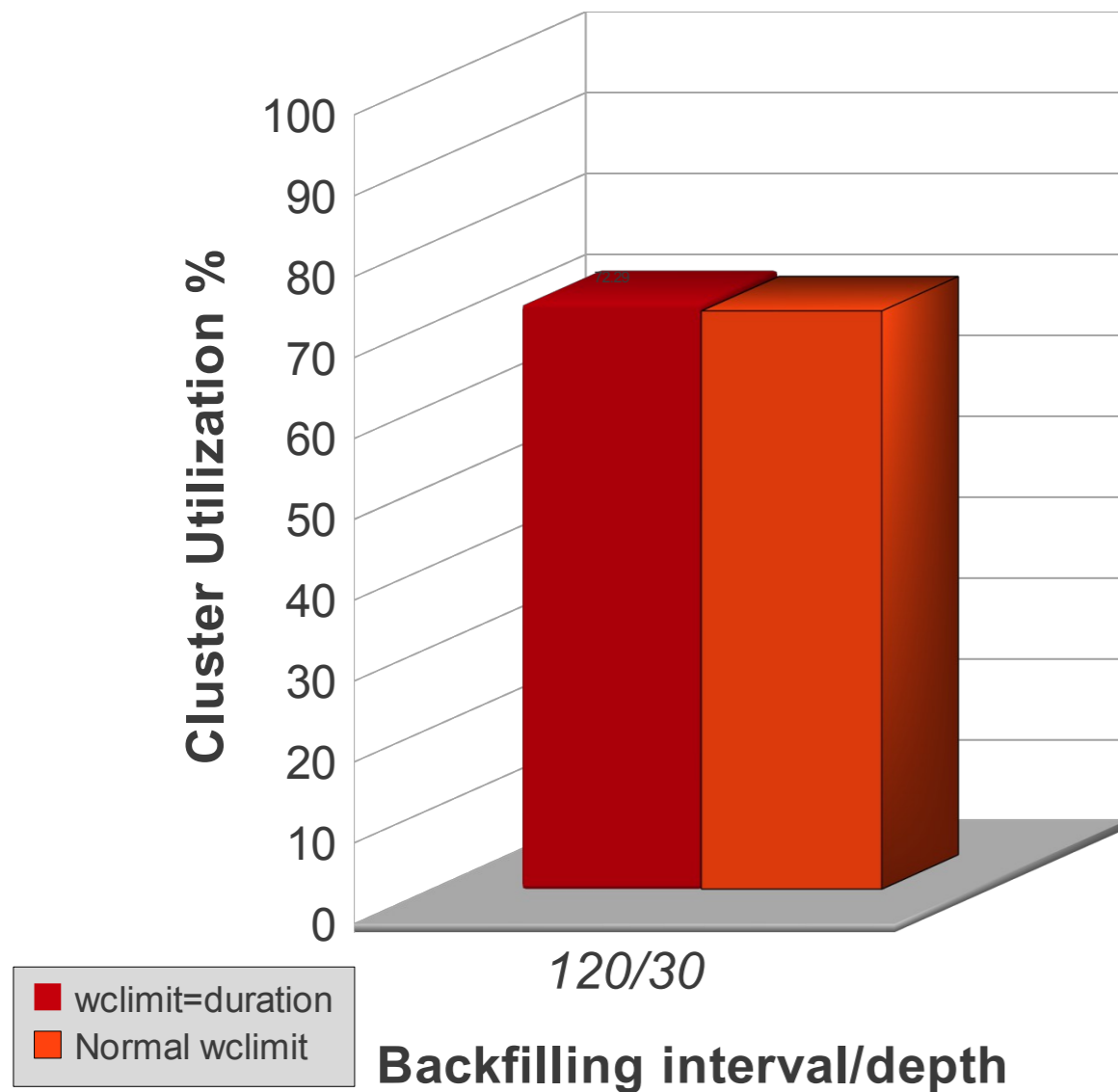


Jobs TimeLimit Impact

- How would be life with users more aware of job timelimit?
- Let's do two executions with same trace with one using a perfect wclimit guess by users



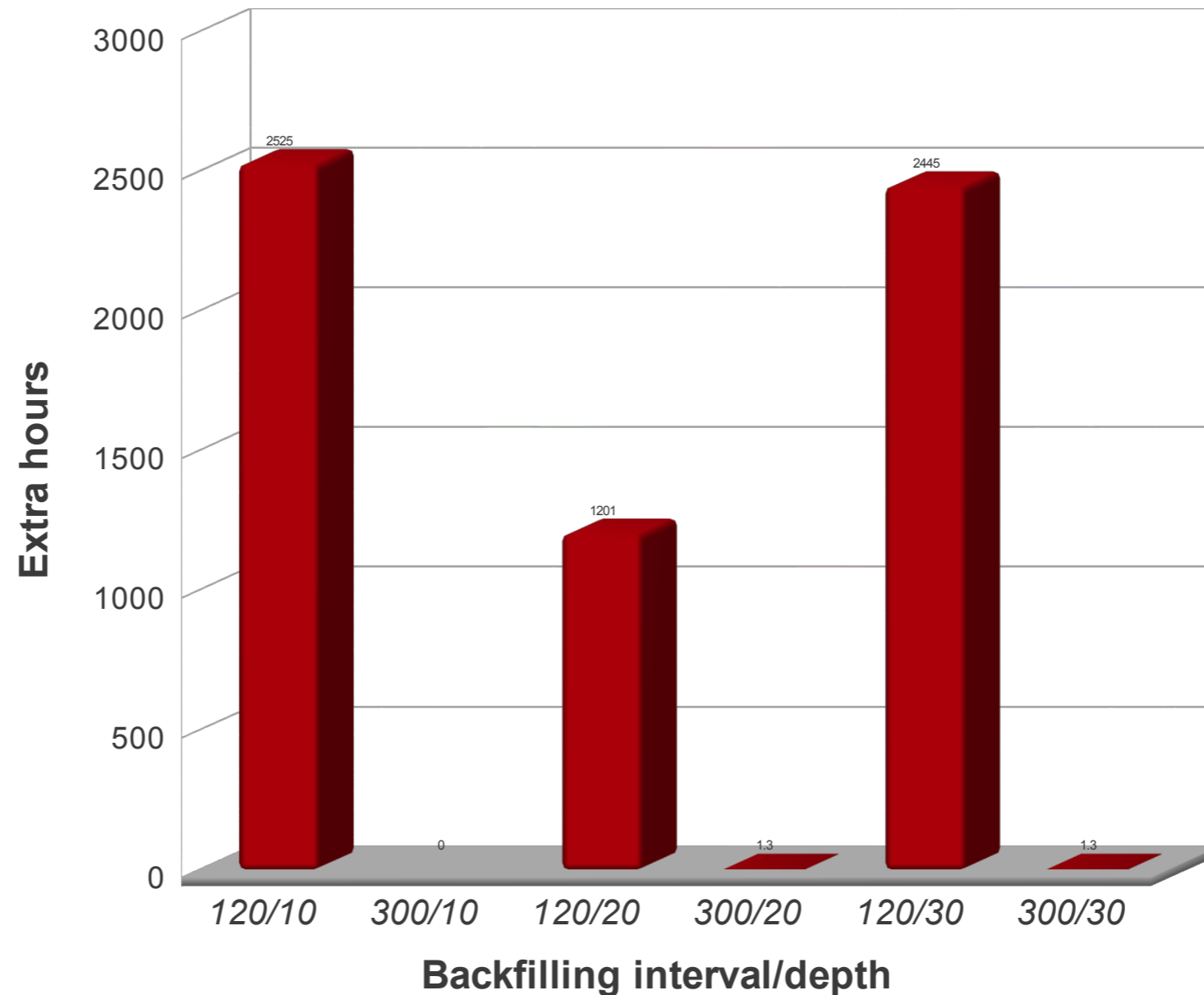
Jobs TimeLimit Impact





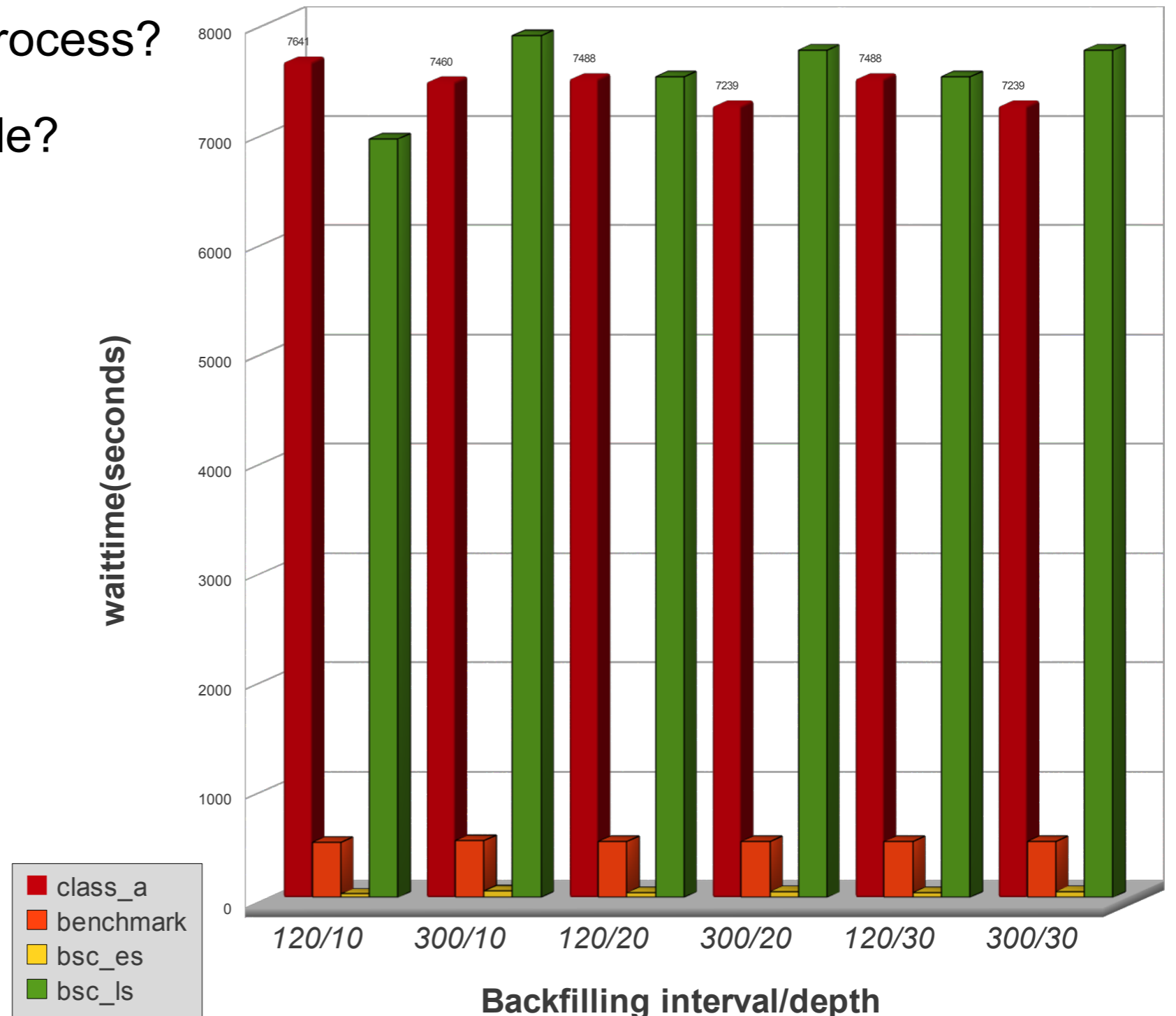
Backfilling Tuning

- How many pending jobs to process?
- How long the scheduling cycle?



Backfilling Tuning

- How many pending jobs to process?
- How long the scheduling cycle?





Backfilling Tuning

- Simulation is not real enough (it could) but ...
- Backfilling parameter tuning does not have a huge impact
- *Hypothesis:*

Marenostrum trace of ~50000 jobs:

- *average interval between job submission < 1 minute*
- *During working hours << 1 minute*
- *last_job_update modified really often*
- *Backfilling algorithm pausing after 5 seconds...*
- *Depth parameter not reached most of the time*
- ♦ *Is Defer parameter enough?*
- ♦ *Keep hold of submitted jobs by some time then inserting them all at a specific point?*



Backfilling Tuning

- How many pending jobs to process?
- How long the scheduling cycle?
- What if we do not respect priority? NoReserved QOS flag
- What if we use Moab-like BF chunk and timewait?

SLURM SIMULATOR



1. Introduction: Why Slurm Simulator?
2. Design & Implementation
3. Results
4. Use Examples
5. **Future Work**



- Killer app? Since first version released I got no news from potential users ...
- I would like to commit the work but, does it make sense?
- This could be used by users/admins, developers and researchers



- Job traces generation. Clasification? Repository?
- Adding flexibility to jobs submission
- Avoiding users accounts for simulation
- Adding node events
- Preemption
- Getting statistics/graphs from slurmdbd
- Slurm core request: statistics for some code functionalities like: backfilling, slurmdbd connections, queued time rate, submission time rate



QUESTIONS?

Thank you

alejandro.lucero@bsc.es