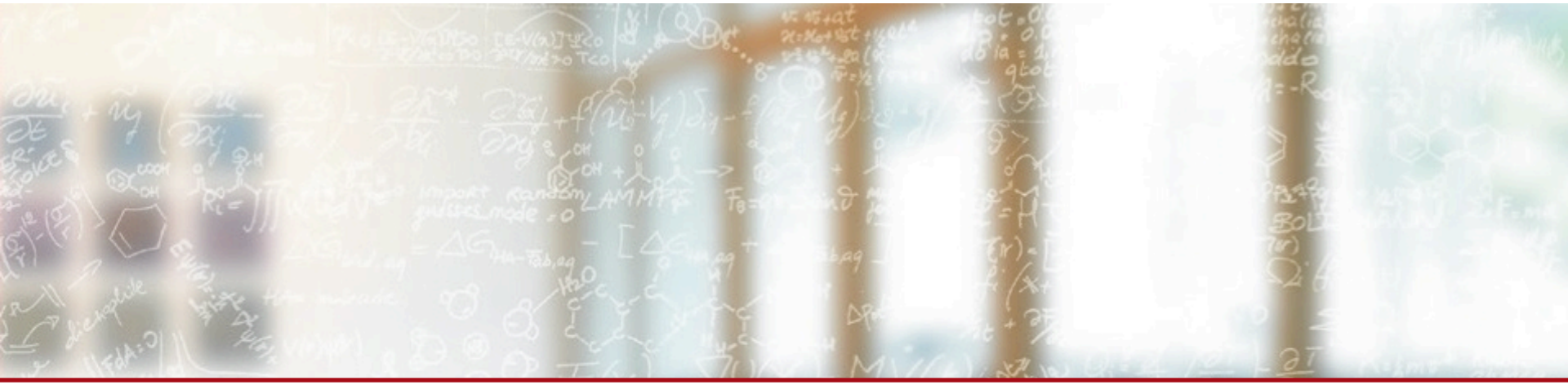
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

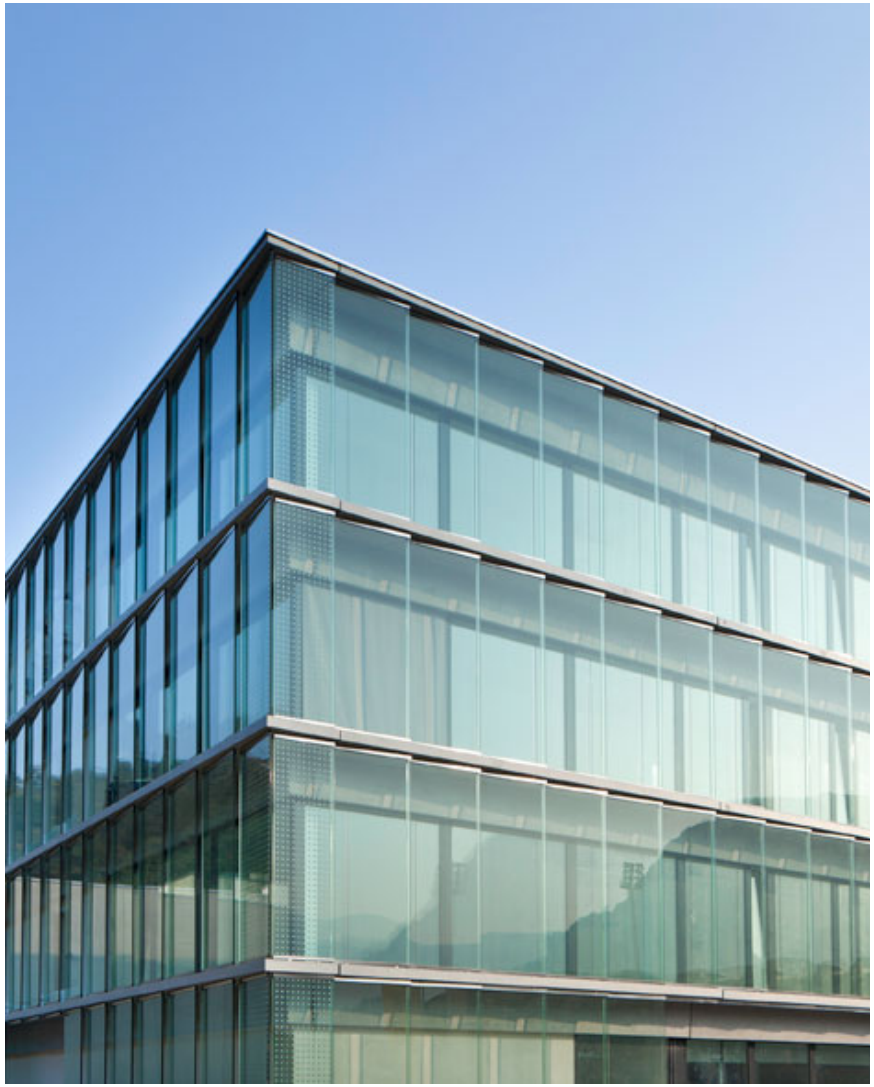
ETH zürich



Using and Modifying the BSC Slurm Workload Simulator

Slurm User Group Meeting 2015
Stephen Trofinoff and Massimo Benini, CSCS
September 16, 2015

Using and Modifying the BSC Slurm Workload Simulator



- The Project
- The BSC Simulator (Rough Overview)
- General Simulator Information
- Worked Performed
- Preliminary Tests and Results
- Future Work and Ideas
- System Diagrams
- Summary

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich

The Project

Purpose

Have ability to simulate a workload, past or theoretical (potentially simulating months of jobs in a relatively short amount of time) and to analyze the state of the system (priorities, states, shares, etc.) under different configurations. In so doing, we could:

- Make better decisions about which configurations options to use
- Answer questions such as when a given job should start
- Determine what the priority or values of shares would be for a job or an account at a given time

General

- Set up some initial test environments using:
  - VM's
  - Emulated Slurm system (with front-end nodes)
- Further simulation desired, mainly to quickly run loads
- BSC Simulator taken as a starting point towards this goal
- General concept is to control the time of the Slurm System and to speed it up

The BSC Simulator (Rough Overview)

General Components

- New system entity, "sim_mgr"--controls simulation
  - If slurmctld is the "brain" of the Slurm system, then sim_mgr is the "brain" of the simulation
  - controls the time as seen by it and Slurm daemons
  - reads job information from file
  - interacts with Slurm creating jobs based on info
- Shared memory used to store time and other common data
- Adds sim_lib--contains wrappers for needed routines (e.g. time())
- Various modifications to slurmctld and slurmd

BSC Workload Simulator Contents

- BSC code written for 2.5.0-rc2--not maintained since
- Launch helper scripts:
  - exec_sim.pl
  - exec_controller.sh
  - exec_slurmd.sh
- sim_sbatch--simple script that preloads sim_lib and runs sbatch)
- Uses various special input files:
  - test.trace: information on jobs to simulate
  - rsv.trace: information on reservations to simulate
  - users.sim: list of simulated user names (used by trace_builder tool and sim_lib.c)

BSC Workload Simulator Contents (cont.)

- A couple of tools for creating and viewing a workload (test.trace) file:
  - trace_builder: builds a very basic “synthetic” workload
  - list_trace: displays contents of file “test.trace”
  - update_trace: allows a partial modification to test.trace
    - Reservation field
    - Dependency field
    - Account field
- reset.sh: Cleaning script to remove the DB entries and various logs from a previous run

BSC Workload Simulator Contents (cont.)

Wrapper functions provided by sim_lib:

- pthread_create
- pthread_exit
- pthread_join
- time
- gettimeofday
- sleep

General Simulator Information

About Simulated Jobs

- In simulated mode, jobs don't do anything, they simply exist
  - submit time (from trace file)
  - start time (Computed by Simulated Slurm--slurmctld)
  - end time (Computed by Simulated Slurm--slurmd)
- Simulator's concept of an event is a time when a job is to end
- The sim_mgr sends a special message, REQUEST_SIM_JOB, directly to the slurmd before submitting a job--informing it the duration of the job
- When the time is reached the slurmd will send the normal message back to the controller indicating that the job is finished.

About Simulated Jobs (Process Flow)

- In special thread “time_mgr”, the sim_mgr steps through time
- For each iteration, time step (1 simulated second):
  - Sim_mgr loops through list of job specs read in at start from test.trace*
  - Sim_mgr sends special message (REQUEST_SIM_JOB) directly to slurmd
    - Job id
    - Duration
  - Builds list of argument strings from the relevant fields in the job spec
  - Forks a process and exec's sim_sbatch with the argument strings
  - Increment time (time step fixed at 1 second)

Running the BSC Simulator

- The following are the basic instructions from BSC's “INSTALL” file (step #10)
  - cd into the SIM_DIR directory
  - $ ./exec_sim.pl SIM_DIR 100
  - This scripts calls sim_mgr which calls slurmctld and slurmd.
  - After 10 seconds or so do:
    - $ ps axl | grep slurm
  - If you can not see sim_mgr, slurmctld and slurmd, there's a problem
  - Check exec_sim.log, sim_mgr.log, slurmctld.log and slurmd_sim.log in SIM_DIR
- Above shows how to run the simulator until 100 jobs complete
- Can also run the simulator for a specified amount of time

Running the BSC Simulator

- Original Simulator determines number of jobs completed by counting lines in file: SIM_DIR/slurm_varios/acct/job_comp.log
- Should run reset.sh in between simulations in order to:
  - Clean database
  - Remove the log files
  - Most importantly—remove the job_comp.log (otherwise simulator will prematurely exit due to thinking that the jobs have already been completed)
- To run for a specified amount of time, must start manually:
  - Start the sim_mgr with a specified end time (in UNIX epoch form):
    - $ sim_mgr 1442393100 # End time is 16-Sep-2015 10:45
  - Start the slurmctld after about 5 seconds:
    - $ exec_controller.sh
  - Start the slurmd
    - $ exec_slurmd.sh
- Note that the start time of simulation should be start of first job in workload

Running the CSCS Simulator

- Doesn't use wrapper script for running
- Instead run sim_mgr directly
  - cd into the SIM_DIR directory
  - sim_mgr [end time] [fork daemons] [throttle]
    - End time—As before--simulated time at which simulation should end. A value of 0 means to run indefinitely.
    - Fork daemons—indicates that sim_mgr should fork/exec the slurmctld and slurmd [optional]
    - Throttle—time interval to increment in each pass instead of just 1 simulated second--[optional and experimental]

Running the CSCS Simulator

- Should run reset.sh in between simulations to:
  - Clean database
  - Remove the log files
- Example: Run simulator indefinitely and start daemons:
  - $ sim_mgr 0 fork
- Example: Run the simulation until 10:45am 16-Sep-2015, automatically starting the daemons:
  - $ sim_mgr 1442393100 fork # End time is 16-Sep-2015 10:45
- Note that the start time of simulation should be start of first job in workload (same as before)

Worked Performed

Worked Performed

- Ported code to 14.03.8
- Created additional tools for generating/editing workloads:
  - mysqltracebuild – Take historical job information from Slurm DB
  - qsnap – Take a “snapshot” of currently running Slurm system (uses C-API)
  - edit_trace – Allows user to edit any field in any number of records of a trace file and for the deletion of records (an enhanced version of BSC's update_trace).
- Encountered various minor issues—started resolving these
- Encountered significant issue with simulation with ~10+ jobs simultaneously completing
  - The slurmctld reported always a failure to create a pthread

Worked Performed (cont.)

- Initially had written enhancements to expand upon the number of threads allowed by the BSC design—encountered various problems
  - Tried increasing MAX_INDEPENDENT_THREADS from 50 to 150
  - Tried increasing MAX_THREADS—the slurmctld would have various problems
  - Increased the amount of shared memory used (product of MAX_THREADS*thread_data_t size)
  - Tried using getrusage at various code points but didn't see any “smoking gun” difference between a “good” and a “bad” run.
  - Tried increasing the per-thread stack space from 1MB to 8MB
  - Checked the code to see if all thread functions used in Simulator had explicit pthread_exit—they did
  - Performed some tracing and noticed that threads did not seem to run for more than a couple seconds and there was never more than 64 threads at one time.
- Thus, it didn't appear to be a true shortage of resources

Worked Performed (cont.)

- Simple idea: just double current capacity of threads (from 32 to 64 threads apiece)
  - Doubled the size of shared memory to 16384 bytes
  - Increased the size of the bitmask arrays in shared memory from one to two long long int
  - Made various other corresponding adjustments
- Result: Increased number of threads could be run—but still had various problems (hanging/crashing)
- Due to the many hangs and of the heavy use of semaphores, changed sem_wait to sem_timedwait
- Result: Simulator ran better, still had issues but was not hanging anymore
  - Without slurmdbd: slurmd would immediately die (1st time), once restarted it would remain up and jobs would run to completion
  - With slurmdbd: slurmd would remain running but jobs would be stuck in pending state
    - This was due to the time_mgr being stuck in wait_thread_running

Worked Performed (cont.)

- Concluded we focused on wrong problem--# of threads
- Real problem appeared to be an issue with the many different locks in use
- Number of threads rapidly increasing was probably due to locking issues
- Due to time and sense that the nature of the design was limiting and overly sensitive to locking, decided to take a different approach
- Still intercept time calls (time, gettimeofday) but not sleep
- Without sleep being wrapped, no need for wrapping the pthreads functions and using the various bit masks, semaphores, etc.

Worked Performed (cont.)

Have begun re-writing much of the code.

- Eliminated the wrappers for pthread functions and supporting code
- Replaced fork/exec of special sbatch with call to Slurm C-API (still use BSC RPC as well)
- Added logic for sim_mgr to fork/exec both the slurmctld and slurmd
- Use a signal from slurmctld and slurmd to notify sim_mgr when they are ready
- Use only a single semaphore to synchronize the sim_mgr with the controller and slurmd

Work Performed (cont.)

Wrapper functions provided by sim_lib:

- ~~pthread_create~~
- ~~pthread_exit~~
- ~~pthread_join~~
- time
- gettimeofday
- ~~sleep~~

Preliminary Tests and Results

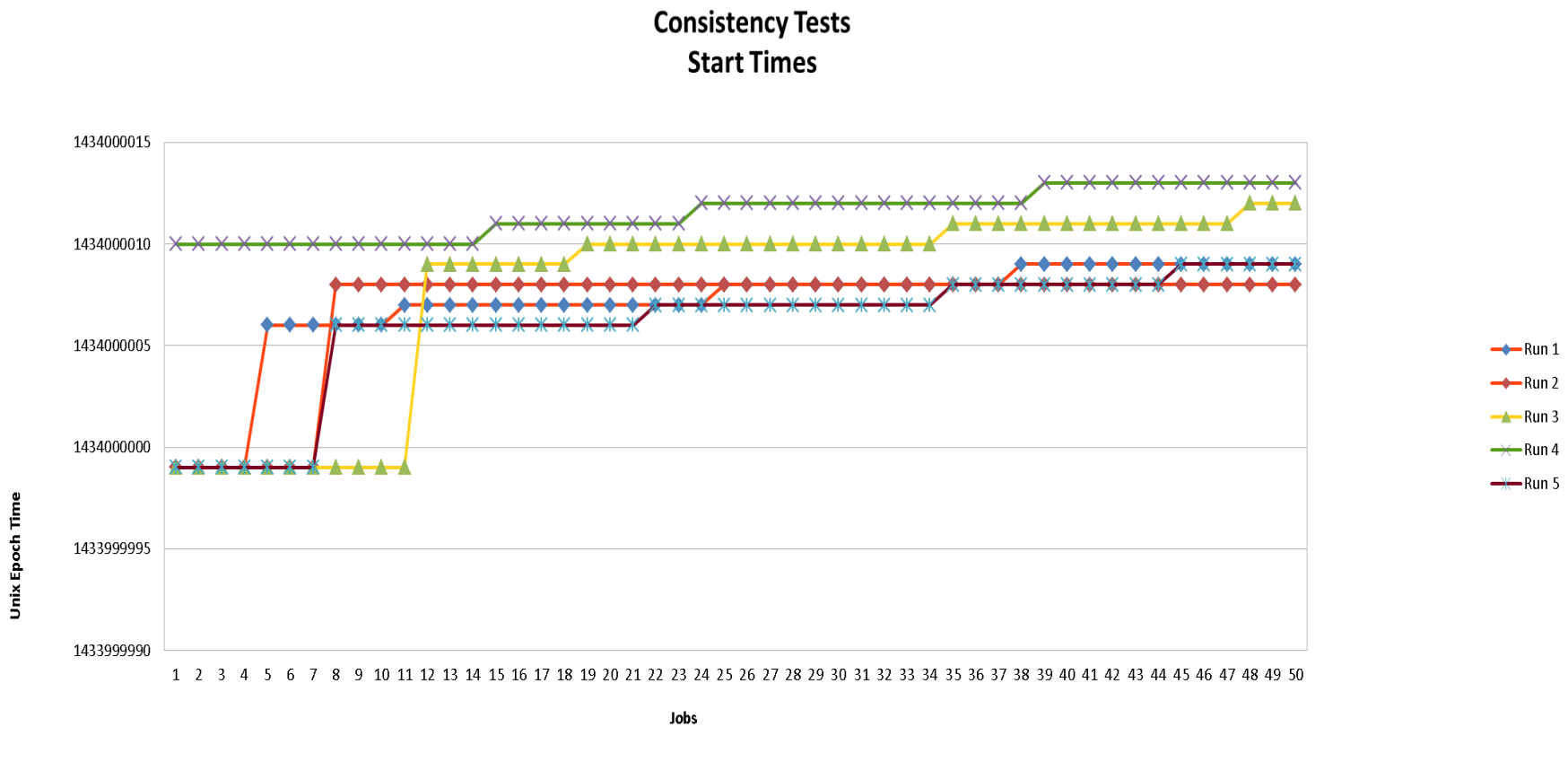
Tests

- Still in early phase
- Defining categories and specific tests
- Will focus on the following points:
  - accuracy
  - consistency
  - performance
  - utility

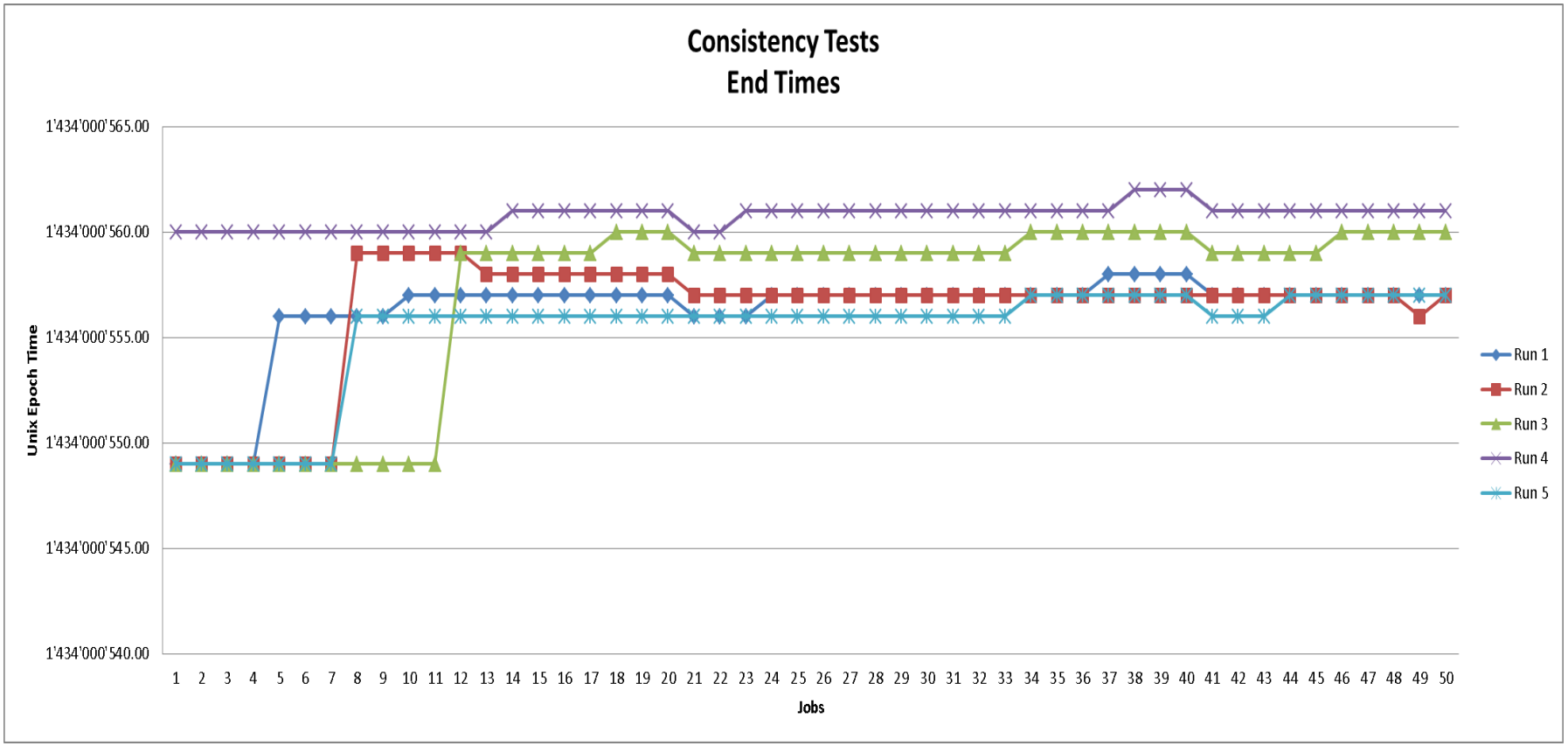
A few early results

- Some crude consistency tests performed
- Took some of our existing workload files and ran multiple times comparing the output from run to run
- Submission times are always exactly the same
- Job durations have slight variation (~1 second)
- Have seen variations (usually just seconds) in both start time and end time
- Need to investigate what is causing this

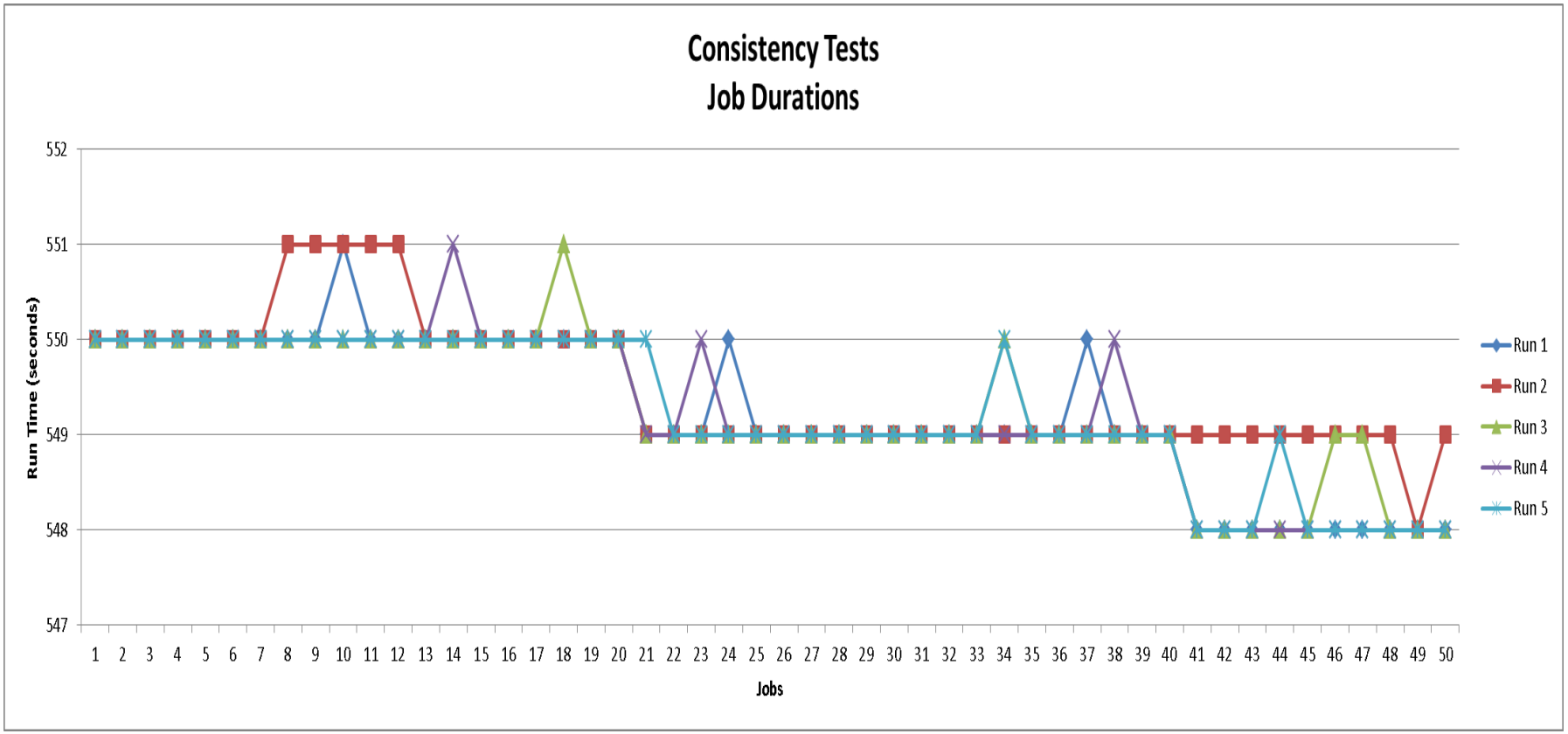
Small 50-Job Workload Results



Small 50-Job Workload Results



Small 50-Job Workload Results



Future Work and Ideas

Future Work and Ideas

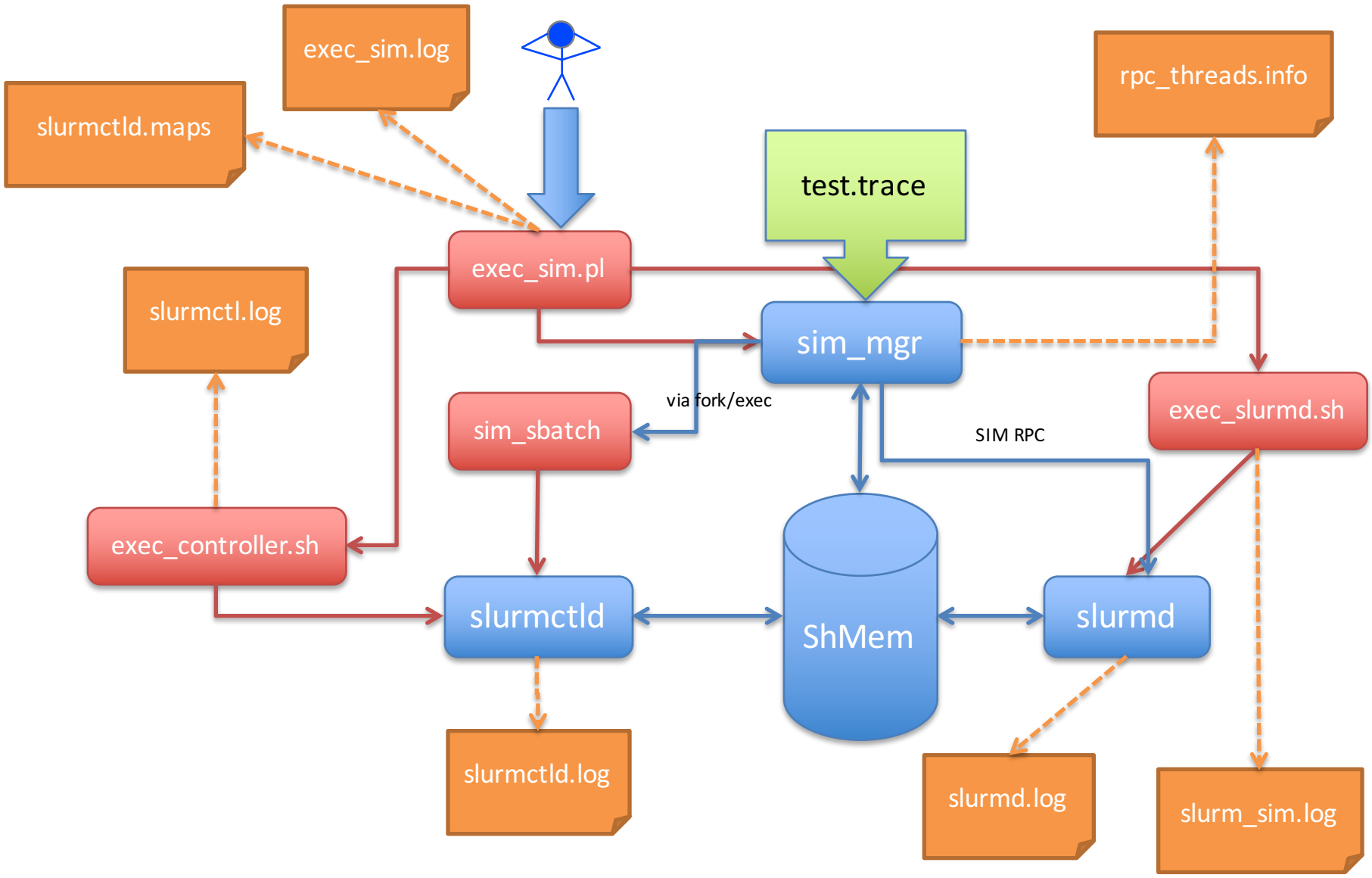
- Continue to define and execute more thoroughly and rigorously the tests
  - Try to improve consistency
  - Fix any problems encountered along the way
- Remove any hardcoded paths where appropriate
- Change squeue and scontrol to use simulated time in printing (sq -i …) and in computing run times of jobs
- Resolve occasional CG state job
- use more job attributes and ensure use of correct attributes
- Should enable slurmbdb to use the simulated time [may not be necessary]
- place code in correct location of the source directory tree
- Potentially create a real “sim_sbatch” that, using C-API, would submit the simulated job to the simulated Slurm, preloading the library automatically and sending the necessary RPC directly to the slurmd for it to build an end-of-job event
- Potentially incorporate the sim_lib into the standard Slurm library so that there would be no need to specially preload anything. Also would allow, logically, for the placement of the code into the normal branches of the source tree [may not be possible, may not make sense]

Future Work and Ideas (cont.)

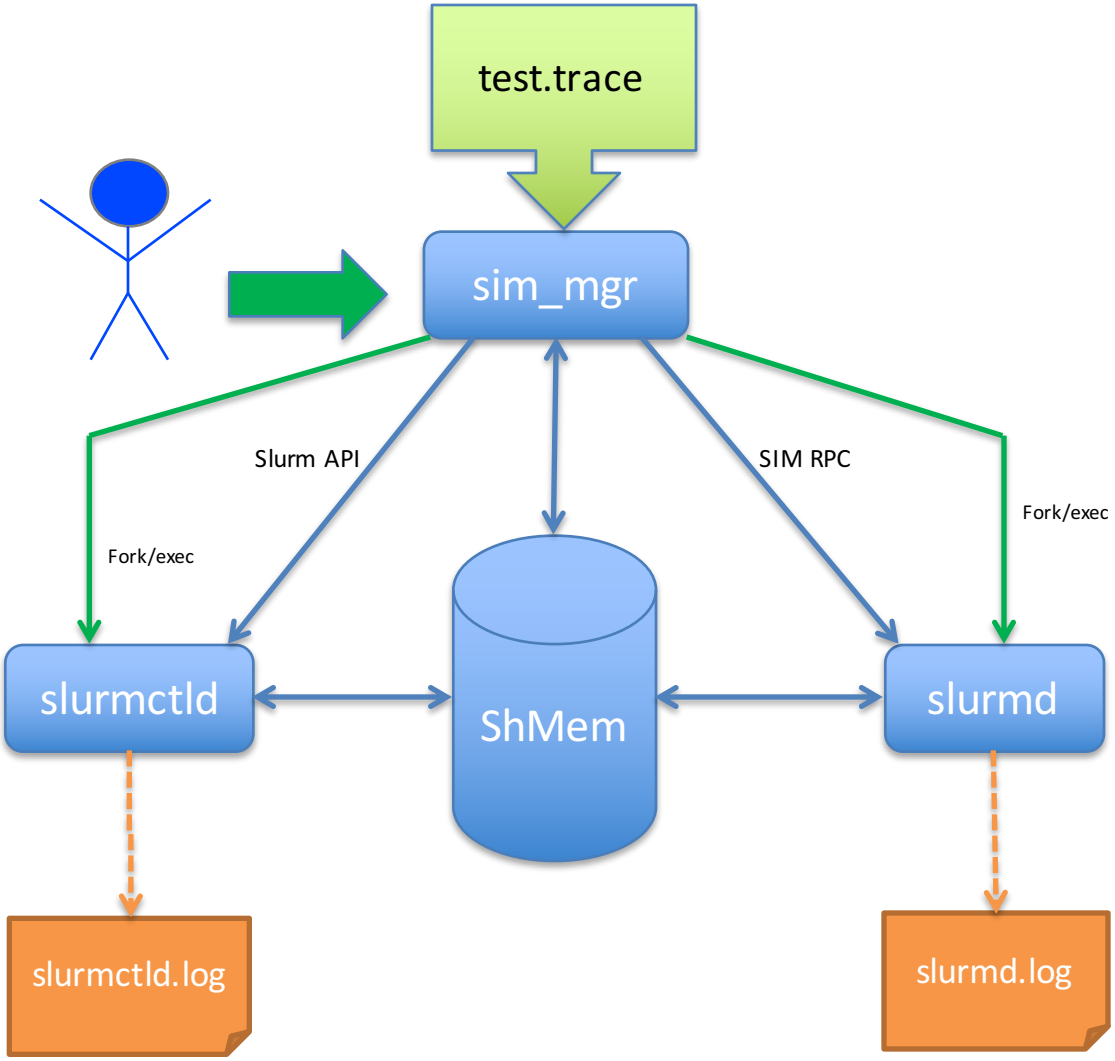
- add time-throttling features (arbitrary time increment)
  - Experiment more with the arbitrary throttle--see if it can be fixed
  - possibly have "intelligent" throttle where the sim_mgr would automatically skip ahead to the next time with an event such as job submit, or expected job completion
  - Note that all arbitrary throttling inherently runs risk of losing accuracy with real system especially the greater the jumps in time
  - Note that the need for speeding up the simulation is especially important due to the tremendous slow down due to the synchronization of the time

System Diagrams

System Overview of Original BSC WS

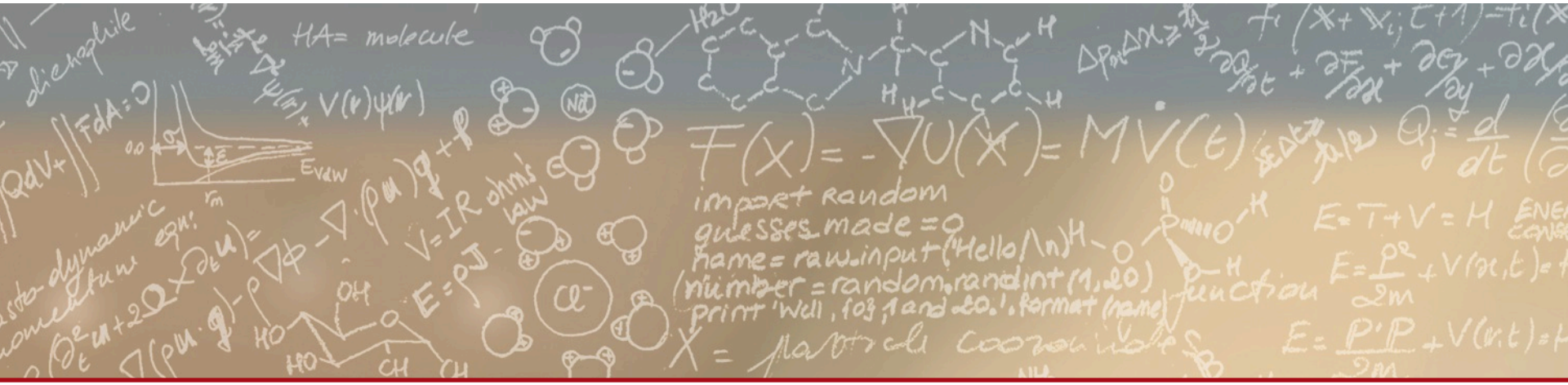


System Overview of CSCS-modified BSC WS



Summary

- Site wants a tool to simulate Slurm workloads
- Started with old BSC Slurm Workload Simulator and are modifying it
- Some more work to do but have a basic simulator running
- Main question to answer—is it useful???



Thank you for your attention.

Appendix

Job Attributes Supported by Simulator

- job id
- username
- submit—the UNIX epoch time at which the job should be “submitted”
- duration—run time in seconds of job (unless getting timed out due to wclimit)
- wclimit—wall clock limit. Max time of job before it gets timed out.
- tasks
- qosname
- partition—job partition to use
- account
- cpus_per_task
- tasks_per_node
- reservation
- dependency

BSC Workload Simulator Contents (cont.)

Data stored in shared memory (timemgr_data pointer to shared memory)

```
global_sem              sem_t*          Scalar
thread_sem              sem_t*          Array [0:MAX_THREADS-1]
thread_sem_back         sem_t*          Array [0:MAX_THREADS-1]
current_sim             unsigned int*   Scalar--current simulated time
current_micro           unsigned int*   Scalar--
threads_data            thread_data_t*
current_threads         unsigned int*
proto_threads           unsigned int*
sleep_map_array         long long int*  Scalar(experimented w/2elements)--
thread_exit_array       long long int*  Scalar(experimented w/2elements)--
thread_new_array        long long int*  Scalar(experimented w/2elements)--
fast_threads_counter    unsigned int*   Scalar--
pthread_create_counter  unsigned int*   Scalar--
pthread_exit_counter    unsigned int*   Scalar--
slurmctl_pid            int*            Scalar--pid of slurmctld
slurmd_pid              int*            Scalar--pid of slurmd
```

BSC Workload Simulator Contents

The simulation_lib directory contains:

- trace_builder.c
- slurm_sim.pl
- sim_mgr.c: Source code for the simulator manager
- sim_ctlr.c
- rsv_trace_builder.c
- rpc_threads.pl: Simple script for producing the rpc_threads.info file
- list_trace.c: Source code for list_trace (viewer of test.trace)
- update_trace.c: Simple program that allows user to partially modify trace files
- sim_lib.c: Contains library code needed by Slurm Simulator system
- rpc_threads.info