Slurm Simulator

HPC Knowledge Meeting 2016
Massimo Benini, Stephen Trofinoff, CSCS
April 21, 2016
Slurm Simulator

- The Project
- The BSC Simulator – Rough Overview
- Simulator General Concepts
- Work Performed
- Demo
- Preliminary Test and Results
- Future works and Ideas
- Appendix
The Project
The Project – Purpose

- Have ability to simulate a workload, past or theoretical and to analyze the state of the system (priorities, stats, shares, etc.).
- Potentially simulate months of jobs in a relatively short amount of time.
- Make better decision about which configuration options to use.
- Answer question about 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.
The Project – Purpose

- 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 (trace files).
- 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()`).
BSC Simulator contents

- BSC code written for 2.5.0-rc2, not maintained since.
- Various modification to slurmctld and slurmd code.
- 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 → jobs to simulate
  - rsv.trace → reservation to simulate
  - users.sim→ list of simulated usernames
A couple of tools for creating and viewing a workload 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
- **reset.sh**: Cleaning script to remove the DB entries and various logs from a previous run.
BSC sim_lib wrappers

- Wrapper functions provided by sim_lib:
  - pthread_create
  - pthread_exit
  - pthread_join
  - time
  - gettimeofday
  - sleep
Simulator General Concepts
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--slrmd)
- 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.
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)
Work Performed
Work performed

- Started in 2014, evaluating and understanding BSC simulator code.
- 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).
  - **sim_date**: Print the current simulation time.
- Encountered significant issue with simulation with ~10+ jobs simultaneously completing (e.g. The slurmctld reported always a failure to create a pthread).
Work performed

- Apparently the problem seemed to be a shortage of resources.
- Initially had written enhancements to expand upon the number of threads allowed by the BSC design encountered various problems:
  - Double current capacity of threads (from 32 to 64 threads apiece)
  - Increased size of MAX_INDEPENDENT_THREADS, MAX_THREADS, per-thread stack space from 1MB to 8MB.
  - Doubled the size of shared memory to 16384 bytes.
  - Increased the size of the bitmask arrays in shared memory.
  - Made various other corresponding adjustments.
- 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.
Work performed

- 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.
Work performed

- 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
- Ported code to 15.08.6.
System overview BSC Simulator

- **exec_sim.log**
- **slurmctld.maps**
- **slurmctld.log**
- **sim_mgr**
- **test.trace**
- **sim_sbatch**
- **slurmctld**
- **slurmctld.log**
- **sim_sbatch**
- **exec_controller.s**
- **exec_sim.p**
- **rpc_threads.info**
- **exec_slurmd.s**
- **slurmd.log**
- **slurmd**
- **ShMem**
- **test.trace**

Connections and processes:
- `slurmctld` communicates with `sim_mgr` via `ShMem`.
- `sim_sbatch` invokes `slurmctld` via fork/exec and communicates with `sim_mgr` using SIM RPC.
- `exec_sim.p` handles simulation output log files.
- `rpc_threads.info` contains information about threads used in RPC communications.
CSCS Simulator Overview

- test.trace
- sim_mgr
- slurmd
- Slurm API
- SIM RPC
- slurmd
- slurmctld
- slurmctld.log
- slurmd.log

Slurm Simulator
Running the CSCS Simulator

```
sim_mgr [endtime]
  [-c | --compath <cpath>]
  [-n | --nofork]
  [-a | --accelerator <secs>]
  [-w | --wrkldfile <filename>]
  [-s | --nodenames <nodeexpr>]
  [-h | --help]
```

Notes:

'endtime' is specified as seconds since Unix epoch. If 0 is specified then the simulator will run indefinitely.

'cpath' is the path to the slurmctld and slurmd (applicable only if launching daemons). Specification of this option supersedes any setting of SIM_DAEMONS_PATH. If neither is specified then the sim_mgr looks in a sibling directory of where it resides called sbin. Finally, if still not found then the default is /sbin.

'secs' for the accelerator switch is the interval, in simulated seconds, to increment the simulated time after each cycle instead of merely one.

'filename' is the name of trace file containing the information of the jobs to simulate.

'nodeexpr' is an expression representing all the slurmd names to use when launching the daemons—should correspond exactly with what is defined in the slurm.conf.
mysql_trace_builder & list_trace commands

- Take a snapshot of a past workload from a Slurm DB and show it:
Slurm Simulator in Action

```
root@bre:~# watch -n 1 simdate
```

```
root@bre:~/cscs4038bscVIII#
root@bre:~/cscs4038bscVIII#
```

```
root@bre:~/cscs4038bscVIII#
root@bre:~/cscs4038bscVIII#
```
Preliminary Tests and Results
Preliminary Tests and Results

- Still in early phase: 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.
Future works and Ideas
Future Works and Ideas

- Continue to define and execute more thoroughly and rigorously the tests:
  - Try to improve consistency
  - Fix any problems encountered along the way
- Resolve occasional CG state job.
- Use more job attributes and ensure use of correct attributes.
- Potentially incorporate the sim_lib into the standard Slurm library so that there would be no need to specially preload anything.
- Create a GUI to better manage all the simulator flow: workload snapshot -> run simulator -> collect data -> analyze data -> clean the environment.
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**
Get the code

- https://github.com/beninim/slurm_simulator
Thank you for your attention.