SLURM SIMULATOR HOWTO

Alejandro Lucero <alejandro.lucero@bsc.es>

30/09/11

1. Introduction

Slurm is a batch scheduler for supercomputer clusters, highly scalable and open source under GNU GPL license. In a HPC site with hundreds/thousands of nodes and users it is not an easy task to implement a scheduling policy covering every single scenario. Slurm offers a good set of configuration possibilities but HPC systems on production are not the best place for testing scheduling changes. It is not negligible at all the problem of obtaining the feedback of those changes since just after hours or maybe days can those changes make an impact on global cluster performance.

Slurm simulator is a specific execution mode for Slurm where it is possible to obtain results of scheduling policies in a shorter space of time. Under simulation jobs are not really executed. Instead, a job execution trace from a real HPC system is used or a synthetic trace made on purpose for testing a specific slurm functionality.

This document explains how to use this execution mode under Slurm.

Simulation regression tests have been done just under x86 systems.

2. Slurm configuration and installation

This document does not address how to work with Slurm, installation or configuration. Information about this can be obtained from Slurm web site and Slurm man pages. However, it is necessary to know some special requirements for simulator mode.

Installing the Slurm with simulator support

You can obtain the slurm sources patched with simulator support from http://www.bsc.es/services …

Last slurm version supported is slurm-2.2.6.

Then with slurm sources:

```
$ tar xvfj slurm-2.2.6.tar.bz2
$ cd slurm-2.2.6
```

Configuring Slurm

Some changes introduced by simulator implementation modifiy configure files which need to be rebuilt:

```
$ ./autogen.sh
```

Configuring and compiling Slurm is not addressed here. However, there is some configuration options needed for working with slurm simulator. Others like installation directories are up to you.

```
$ ./configure …  –enable-front-end  …
```

If you are going to use the configuration files we offer in our web site you need to create you own ssh key for slurmctld – slurmd communications. Using commands like this:

```
$ openssl genrsa -out $instdir/etc/slurm.key
$ openssl rsa  -in $instdir/etc/slurm.key -putbout-out $instdir/etc/slurm.cert
```

and then referencing *slurm.cert* file at slurm.conf using parameter ***JobCredentialPublicCertificate***.

3. Simulation requirements

There are some previous steps before running slurm under simulation mode:

- You need to create a ***slurm.conf*** . See *man slurm.conf* for details.

  Most Slurm configuration options are avalaible except ***preemption***. Regression tests have been focused on simulator being deadlock free (a lot of thread-to-thread talking) and being deterministic.

  FirstJodId parameter is important since simulation manager needs to know job identifiers in advance. Use ***FirstJobId=1000*** here since this is the first identifier used by ***trace_builder*** program (keep reading). For a real trace it should be first JodId submitted and all job id using through the trace need to be consecutive. Yes, of course, we need to improve this soon.

  You can start using slurm configuration files available at www.bsc.es/services . You can choose between one slurm.conf file using ***priority/basic*** an another one using ***priority/multifactor***. With the last one you will need a complete slurmdbd installation and to create the slurmdbd.conf. There are some parameters with XXXXXXXXX value which you need to fill up accordingly.

- You need to create a ***slurm.nodes***. Slurm is configured with ***FRONTEND*** option actived so you do not need real nodenames here. Again you can use a ***slurm.nodes*** file available at BSC website but remember to replace machine name just typed as XXXXXXXXX inside the files to the one where you will run the simulator.

- If ***slurmdbd*** is configured you need to create the right ***slurmdbd.conf*** file. Installing ***mysql*** is not addressed here neither how you have to manage ***slurmdbd***. Probably it is a good idea to start it independently of the other slurm components, ***slurmctld*** and ***slurmd***, and ***sim_mgr*** program.

- You need a trace file and by now it is easier to use a synthetic one. A ***trace_builder*** program, created during compilation under ***simulation_lib*** Slurm directory, is able to build a simple trace file for you. Random values are used for job execution time and required nodes by job. Users able to send jobs need to be listed on a text file named ***users.sim***, one username per line and a user which such a name needs to be created on the system. The same mechanism is used for qos, but using another file named as qos.sim. Parameters required for trace_builder are:

    - total cpus available in the cluster
    - number of jobs to create
    - default partition
    - default account
    - default cpus_per_task
    - default tasks_per_node
    - submit time (unix time) for first job

  A trace file named ***test.trace*** will be created. Total cpus needs to match *slurm.nodes* file definitions. Another program, ***list_trace***, can be used for seing details about jobs created.

  ```
  $ cd slurm-2.2.6/src/simulation_lib
  $ echo  alucero > users.sim
  $ echo cfenoy  >> users.sim
  $ echo normal > qos.sim
  $ echo highprio  >> qos.sim
  $ echo lowprio  >> qos.sim
  $ trace_builder --cpus=3000 --jobs=5000 –partition=projects
        --account=bsc_es --cpus_per_task=1 --tasks_per_node=4
        --submit_time=1316223705
  ```

- The simulation manager needs to know which threads need a special treatment. A ***rpc_thread.info*** file needs to be created with that information. Once slurm is compiled a script at ***slurm-2.1.9/src/simulation/lib*** can be executed to create the file.

  ```
  $ cd slurm-2.19/src/simulation_lib
  $ ./rpc_threads.pl
  ```

  The new file created, ***rpc_thread.info***, needs to be at the same directory where sim_mgr will be executed.

- Last thing is to create a file named test_command.cmd at same directory where sim_mgr will be executed. As with Slurm simulator no jobs are really executed, it is not needed a complete file script with commands. Just with the usual header for bash shell scripts is enough.

4. Simulation execution

After a successful Slurm compilation and once slurm configuration files are created, next step is just using the simulator. There are several components we should have right now:

- ***sim_mgr*** executable
- ***rpc_threads.info*** at same directory where *sim_mgr will be executed*
- a ***test.trace*** file at same directory where *sim_mgr will be executed*
- *test_command.cmd file at same directory where sim_mgr will be executed*
- ***slurmctld*** and ***slurmd*** executables

Before working with the simulator is a good practice to see if your slurm is configured properly. You can start slurmctld and slurmd directly and using -D parameter for debug. Remember to add lines at slurm.conf for SlurmctldDebug=9 and SlurmdDebug=9 and SlurmctldLogFile parameter. If you can see slurmctld and slurmd running and ***sinfo*** shows the expected output then you can go forward to work with the simulator.

Executing the simulation manager ***sim_mgr*** needs ***SLURM_PROGRAMS*** variable pointing to sbatch directory. There is a parameter for *sim_mgr* execution: simulation ending point. It allows to stop simulation in a specific point in time. By now it is better to set the value to 0, which has no effects at all.

```
$ cd 'WHERE sim_mgr executable is installed '
$ sim_mgr 0
```

Once *sim_mgr* starts it waits for both *slurmctld* and *slurmd*. Remember that just one slurmd is needed even if we have thousand of nodes since FRONTEND is configured.

Next steps are executing *slurmctld* and *slurmd*. On simulation mode some time and thread related functions are wrappered inside simulation library ***libslurm_sim*** . For those functions to be executed instead of real ones, ***LD_PRELOAD*** needs to be configured for slurmctld and slurmd.

```
$ cd $instdir/slurm_programs/sbin
$ LD_LIBRARY_PATH=$instdir/lib/slurm    LD_PRELOAD=libslurm_sim.so
        ./slurmctld -f $instdir/slurm_conf/slurm.conf -D 2&> slurmctl.log &

$ SLURM_CONF=$instdir/slurm_conf/slurm.conf
        LD_LIBRARY_PATH=$instdir/slurm_varios/lib/slurm
```

```
LD_PRELOAD=libslurm_sim.so
./slurmd -D 2&> slurmd_sim.log &
```

And that's it.

Now the console where sim_mgr was launched should show a lot of information.

You are a good unix teachie so you can easily create your own scripts for running simulation easily.

5. Future Work

Getting slurm simulation running can be painful if all the steps commented above are necessary each time slurm is updated. We have a simulation bench for nightly tests where all the process is done: from getting last slurm sources (with simulation code changes), configuring and compiling slurm, creating configurarion files, creating those specific files for simulation like test.trace and rpc_threads.info, launching sim_mgr, slurmctld and slurmd, and waiting until all jobs have completed. We do a final step cleaning up slurm state files for being able to run another simulation from a clean starting point.

We expect simulator being used for testing slurm configuration changes based on real job traces from HPC sites. Also it could be interesting for slurm development testing easily new implementations. Both scenarios would need such an automatic execution as we have now so it would be interesting to create just one tool flexible enough for most people requirements.

There are other things on the TODO list, of course, as including preemption support (this is just trivial for simulation), a tool for creating real traces from slurm log files or accounting data, improving the synthetic trace builder, and porting simulator patch to last slurm versions. We think such a porting should be easy as long as Slurm architecture remains without main changes.