Implementing and Evaluating Multi-Resource Scheduling in Slurm

Sirisha Cherala

ILLINOIS INSTITUTE OF TECHNOLOGY

Implementing and Evaluating Multi-Resource Scheduling in Slurm

Sirisha Cherala

Contents

Chapter 1

Project Overview

The exponential growth in computing power has enabled High Performance Computing(HPC) Systems to attack problems that are much larger and more complex. One example would be Data-Intensive applications. To meet intense I/O demand for these applications, Burst buffers are often employed in production systems. This heterogeneity forces HPC Schedulers which are mainly CPU Centric to consider multiple resources like Burst Buffers, network bandwidth in order to make optimized decisions for efficient performance [1–5].

In this Project, we consider Slurm, a well-known Scheduler in HPC Systems and incorporate multi-resource scheduling scheme. This is done by formulating the scheduling problem into a Multi-Objective optimisation problem and solving it using a multi-objective genetic algorithm [6]. Existing Slurm Scheduler supports Burst-Buffer Scheduling and allocates jobs from the waiting queue until either CPU or burst buffer is exhausted. This approach has a limited efficiency as the depletion of one resource can prevent the queued jobs from allocation and thus causing the under-utilisation of the other resources [7–9]. As such, we implement a plugin called BBSched which when configured allows Slurm to perform multi-resource scheduling. The implementation is tested using Slurm Simulator [10].

This Project has been performed for a period of 12 weeks starting from January 14 2019 - April 26 2019 under the supervision of Dr.Zhiling Lan and her doctoral student Yuping Fan. The status of the project and any clarifications were discussed during the weekly meetings which are held every Thursday from 2:00 - 2:30 PM. Besides for every two weeks,a Biweekly report has been submitted which summarises the work done in the past two weeks and the goal for the next two weeks. The reports can be viewed from the webpage at `http://www.cs.iit.edu/~lan/moo.html`.

Week No.	Task
1 and 2	Understanding Slurm and MOO Scheduler
3 and 4	Installation of Slurm and Running Slurm Controller with Slurmd daemons
5 and 6	Coding for the new Scheduler Plugin "BBSched"
7 and 8	Understanding Slurm Simulator,installing it on Docker
9 and 10	Integrating the plugin with Slurm Code and Slurm Simulator
11 and 12	Generating the workload trace, debugging to fix errors and result Analysis

Table 1.1: *Overview of the Work specified Biweekly.*

Chapter 2

Project Tasks

2.1 Understanding Slurm

The main Objective of this project is to implement multi-resource scheduling scheme (named as "BBSched") in Slurm. For this, the first task was to understand Slurm. This is done by reading the Slurm design from [11] and going through the tutorials which have been delivered by the Slurm Programmers at various conferences and can be viewable at [12]. These tutorials and the design manual gave a good insight of Slurm by describing what Slurm is and what it is not; it's architecture which includes detailed description of Slurmd, Slurmctld, Command Line Utilities, Plugins, Communication layer and Security model; the design of Slurm Control daemon, it's subsystems: Node manager,Partition manager and Job manager, it's configuration and fault tolerance; the design Slurmd daemon; the working of each of the following command line utilites: scancel, scontrol, squeue, sinfo and srun; and the Job Initiation design.

This deep understanding of Slurm and its plugins helped in knowing that the multi-resource scheduling scheme needs to be incorporated as a new scheduling plugin so as to keep the existing functionality undisturbed and configure the new scheduling scheme only when it is required by the system administrator. As such, the programmer guide and plugin implementation details of Slurm are read from [13] and [14]. Further, the Slurm code from github [15] has been analysed by installing it in the local machine. Specifically, the prevailing scheduling plugins namely sched/builtin and sched/backfill have been explored so as to know how the slurm control daemon (precisely the Job Manager) communicates with the scheduling plugins. Besides, various Slurm scheduling design papers [16] [17] [18] [19] [20] have been examined.

2.2 Understanding Slurm Simulator

Time accelerated simulation of workloads is needed to verify the implementation of Multi-resource scheduling in Slurm. As such, an Open source Slurm Simulator based on [21], [22] and [23] is used in this project. As, it is important to understand the work flow of the simulator before using it, the aforementioned papers have been read. Further, the code of the simulator from the github [10] has been analysed line by line to get a clear overview of its working.

2.3 Design of Multi-resource Scheduling in Slurm

The design of Multi-resource Scheduling in Slurm is shown in the figure below.

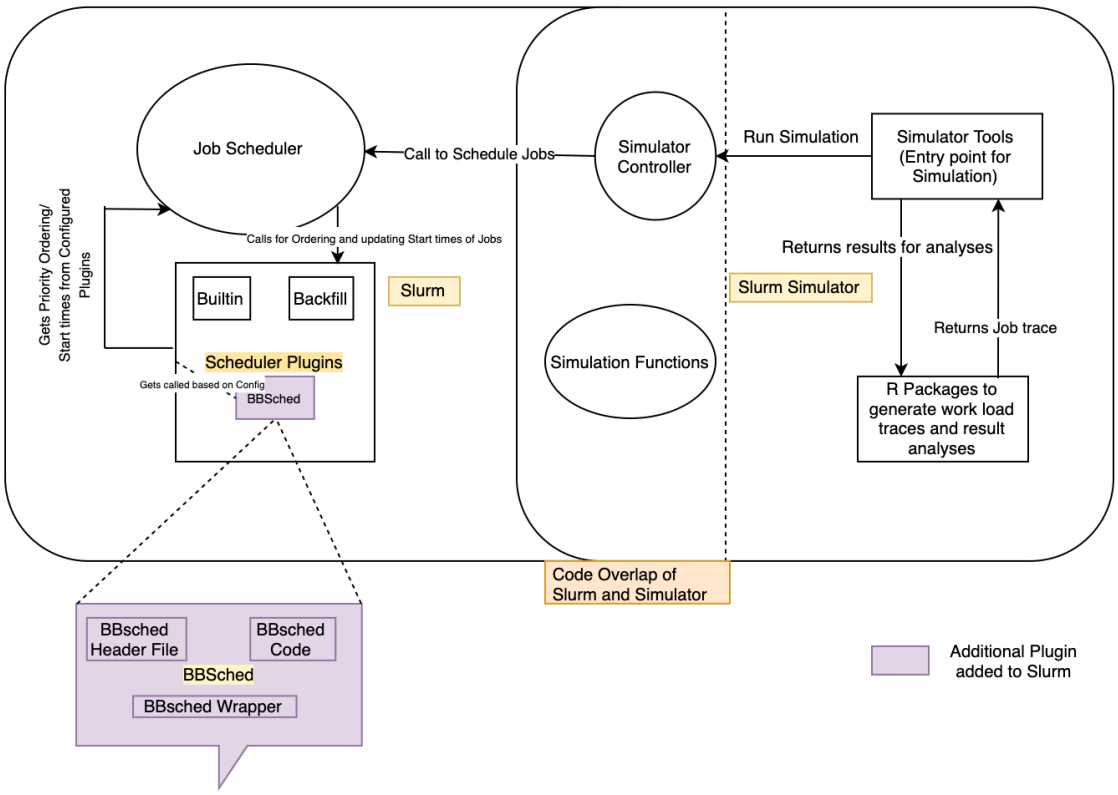

Figure 2.1: *Design of Multi-Resource Scheduler in Slurm*

In this project, a new Scheduler Plugin namely "BBSched" has been added to the Slurm Scheduler Plugins. As such, when scheduler type is configured as SchedulerType=sched/bbsched in slurm.conf, the Job Scheduler of Slurm Control Daemon schedules jobs based on the priority ordering from the BBSched plugin. This Priority ordering used by BBSched plugin to optimise multiple resources is based on the Genetic Algorithm for Multi-objective Optimisation.

Since, the simulator uses the Slurm Code for scheduling, the change in configuration automatically reflects to the Simulator and as such no specific design changes are needed in Simulator.

2.4 Implementation of BBSched in Slurm

The implementation of the new plugin "BBSched" is done and tested on docker container running on a virtual machine on Lightning Server of IIT. The IP Address of lightning server is 216.47.142.240 and virtual machine is 192.168.122.99. Inside the docker container, all the modified changes related to Slurm and Slurm Container are present in /home/slurm

folder. The Step by Step manual written as a part of Appendix would give more details on Installation. Here, we specify only the implementation part.

2.4.1 Adding BBSched Plugin to Slurm

The Slurm related code is present in /home/slurm/slurm_sim_ws/slurm_simulator and slurm simulator code is present in /home/slurm/slurm_sim_ws/slurm_sim_tools. Inorder to add the new plugin to Slurm code, a new directory called bbsched is created and added to Makefile.am both of which are located at /src/plugins/sched. Inside bbsched folder, four files are added namely bbsched.c, bbsched.h, bbsched_wrapper.c and Makefile.am. Once all the files have been added, ./autogen.sh is run on the Slurm's top level directory so as to update the existing Makefile.in files and create the new Makefile.in file for the bbsched plugin. Besides, configure.ac is modified to identify the new Makefile to be built at Slurm configure time. Precisely the following line: src/plugins/sched/bbsched/Makefile is added to configure.ac inorder for the plugin to be built at Slurm Configure time.

2.4.2 Coding for Multi-resource Scheduling in Slurm

Here, the functionality added in each file of the new plugin: src/plugins/sched/bbsched is described.

- ***bbsched_wrapper.c*** :
  This file has the functions which are mandatory for Scheduler plugins and directly follows the Slurm Scheduler Plugin Api documentation [24]. It includes:
  1. Entry point function **init()** where the global initialisation is placed. This includes creation of threads specific to the bbsched plugin, acquiring locks and code specific to simulation mode like running the bbsched plugin from main simulation loop.
  2. **fini()** function which releases the acquired mutexes, stops the plugin agent and suspends the execution of the bbsched thread.
  3. **slurm_sched_p_reconfig** function which rereads the configuration files in case on any changes.
  4. All the other functions which have no specific implementation have been stubbed.

- ***bbsched.c***
  This file has the function which implements Multi-resource scheduling. Inorder to implement the code, the base paper [6] has been thoroughly read. Besides, CQSim [25] [26] [27] [28] has been installed and the Multi-Objective Optimisation (MOO) Scheduler in CQSim has been examined so as to code Slurm in a Similar way.
  As such the following functions are implemented:
  1. **run_genetic_algorithm** function: This is the entry point of the Multi-objective optimization algorithm. It is called by the sort_job_queue function to update the priority ordering of the jobs in the waiting queue. The basic functionality includes, considering the jobs only pertaining to the window size and performing population initialization, parent selection, crossover, mutation and survivor selection. This process is done for around 500 iterations (generations) and the population after the last generation is returned.

2. **init_population** function: This function generates a random binary value for each of the jobs in the window. So, For example if the number of jobs in window is 5 and random ordering is 10110, it implies that 1,3 and 4 jobs are selected; 2 and 4 are not. We then check if this ordering of jobs is actually feasible. This is done by computing CPU Utilisation(described in detail in compute_fitness _value function). All the feasible orderings of jobs are returned at the last, the size of which is equal to the population size considered.
3. **parent_selection** function: This function returns two randomly selected orderings from the populations initialised above.
4. **crossover** function: This function uses the parent list generated above and swaps the ordering at random position.It then checks for the feasibility of the swapped ordering and returns the feasible list as result.
5. **mutation** function: This function is used to introduce diversity so that the algorithm would not be trapped in the local optima. Specifically, it looks over all the orderings in the feasible population and flips the binary value at a random position. It them checks for the feasible orderings in the flipped list and returns them.
6. **survivor_selection** function: This function returns the best among the recently generated orderings and separates them into two sets. Pareto Solutions in Set1 and rest in Set2. A solution is chosen as a Pareto solution, if improving one of its objectives would deteriorate at least one other objective [6]. If the set1 has less number of orderings than the population size considered, all the orderings in set1 are returned and the best among set2 are appended.
7. **compute_fitness_value** function: For each of the job orderings, this function takes into account the jobs which have a binary value of 1 (implies those that are selected) and sums up the number of nodes required by each of the jobs. It then looks for the total number of nodes using assoc_mgr functions and also gets the currently allocated nodes. Then utilisation is computed as (total nodes - available nodes + required nodes)/total nodes. Here available nodes = total nodes - allocated nodes and required nodes is the sum of number of nodes required by each of the jobs in the ordering.
8. **sort_job_queue** function: This function takes the waiting list of Jobs updated each time by the Job Manager of Slurm and calls the run_genetic_algorithm function described above. It then takes the best among the orderings returns and the jobs which have binary value of 1 are added to the queue.

- ***bbsched.h***
  This is a header file for the bbsched plugin

- ***Makefile.am***
  This is written using the existing Makefile.am of sched/builtin as a model. It basically specifies the top source directory of Slurm and the files inside the current directory which need to be built at compile time.

Further, slight modifications are made to /src/slurmctld/simulator.c file to accomodate for bbsched plugin. These modifications include calling the plugin from Simulator main loop for job scheduling and including bbsched output in the log file. Further, bbsched job scheduling statistics are added to /src/slrumctld/slurmctld.h and debug flags to /slurm/slurm.h.in.

Chapter 3

Results

Here, Slurm is configured such that it manages 10 compute nodes and 5 users with 2 accounts. The Job trace consisting of 500 jobs is generated using a R Script and is similar to the log from Theta machine.

3.1 Experiment 1

Here, SchedulerType is configured as sched/bbsched in slurm.conf present in /home/slurm/slurm_sim_ws/sim/m folder. The simulation time for 500 Jobs was 23 seconds.

- Average Job Slow down: Between 5-7 Seconds
- CPU Utilisation - 70-77%

3.2 Experiment 2

Here, SchedulerType is configured as sched/builtin in slurm.conf. The simulation time for 500 Jobs was 15 Seconds.

- Average Job Slow down: Around 6 Seconds
- CPU Utilisation - 73%

Note: The results with builtin (FIFO) scheduling are not direct as the simulator doesn't seem to use the builtin plugin of the Slurm and it only schedules 50 jobs out of 500 jobs given as input. The above result is an extrapolated value.

3.3 Observations

Here, we are optimising a single resource, so the effect of the Multi Objective Optimisation algorithm doesn't seem much. But if we add more resources like Burst Buffer, the efficient utilisation of multiple resources would be seen.

Besides, the following is the graph that shows the Simulation time Vs Number of generations by BBSched Scheduler.

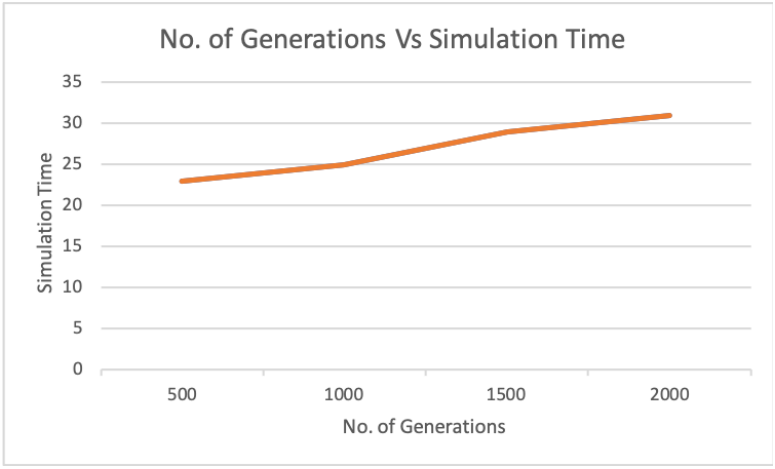

Figure 3.1: *Simulation Time Vs No. of Generations*

Chapter 4

Conclusion

In this project, Slurm and Slurm Simulator have been deeply explored and their installation is done on Docker. Further, Multi-resource scheduling has been implemented in Slurm by adding a new plugin called BBSched to the Scheduler Plugins. The workload trace to test the implementation has been generated based on theta log and the results were examined for 500 Jobs. Since, the CPU optimisation is considered in this project pertaining to time limitations, the utilisation seems to be nearly same. The future work would involve fixing the issue with slurm simulator when scheduler type is builtin and to incorporate burst buffer management in Simulator so as test the Multi-resource scheduling.

References

[1] Yuping Fan, Paul Rich, William Allcock, Michael Papka, and Zhiling Lan. Trade-Off Between Prediction Accuracy and Underestimation Rate in Job Runtime Estimates. In *CLUSTER*, 2017.

[2] Peixin Qiao, Xin Wang, Xu Yang, Yuping Fan, and Zhiling Lan. Joint Effects of Application Communication Pattern, Job Placement and Network Routing on Fat-Tree Systems. In *ICPP Workshops*, 2018.

[3] Peixin Qiao, Xin Wang, Xu Yang, Yuping Fan, and Zhiling Lan. Preliminary Interference Study About Job Placement and Routing Algorithms in the Fat-Tree Topology for HPC Applications. In *CLUSTER*, 2017.

[4] Yuping Fan and Zhiling Lan. Exploiting Multi-Resource Scheduling for HPC. In *SC Poster*, 2019.

[5] Li Yu, Zhou Zhou, Yuping Fan, Michael Papka, and Zhiling Lan. System-wide Trade-off Modeling of Performance, Power, and Resilience on Petascale Systems. In *The Journal of Supercomputing*, 2018.

[6] Yuping Fan, Zhiling Lan, Paul M. Rich, William E. Allcock, Michael E. Papka, Brian Austin, and David Paul. Scheduling beyond cpus for hpc. *ACM HPDC*, November 2019.

[7] Y. Fan. Job Scheduling in High Performance Computing. In *Horizons in Computer Science Research*, 2021.

[8] Boyang Li, Sudheer Chunduri, Kevin Harms, Yuping Fan, and Zhiling Lan. The Effect of System Utilization on Application Performance Variability. In *ROSS*, 2019.

[9] William Allcock, Paul Rich, Yuping Fan, and Zhiling Lan. Experience and Practice of Batch Scheduling on Leadership Supercomputers at Argonne. In *JSSPP*, 2017.

[10] Nikolay Simakov. Slurm simulator github. `https://github.com/ubccr-slurm-simulator`.

[11] Morris Jette and Mark Grondona. Slurm design. `https://slurm.schedmd.com/slurm_design.pdf`.

[12] Morris Jette. Slurm video tutorials. `https://slurm.schedmd.com/tutorials.html`.

[13] SchedMD. Slurm programmer guide. `https://slurm.schedmd.com/programmer_guide.html`.

[14] SchedMD. Slurm plugin guide. `https://slurm.schedmd.com/plugins.html`.

[15] SchedMD. Slurm github. `https://github.com/SchedMD/slurm`.

[16] Yuping Fan, Zhiling Lan, Taylor Childers, Paul Rich, William Allcock, and Michael Papka. Deep Reinforcement Agent for Scheduling in HPC. In *IPDPS*, 2021.

[17] SchedMD. Slurm scheduling design. `https://slurm.schedmd.com/slurm_ug_2012/SUG-2012-Scheduling.pdf`.

[18] Yuping Fan and Zhiling Lan. DRAS-CQSim: A Reinforcement Learning based Framework for HPC Cluster Scheduling. In *Software Impacts*, 2021.

[19] Michal Novotn. Slurm scheduling thesis. `https://is.muni.cz/th/c4s5x/thesis.pdf`.

[20] Yuping Fan, Paul Rich, William Allcock, Michael Papka, and Zhiling Lan. ROME: A Multi-Resource Job Scheduling Framework for Exascale HPC System. In *IPDPS poster*, 2018.

[21] Ana Jokanovic, Marco D'Amico, and Julita Corbalan. Evaluating slurm simulator with real-machine slurm and vice versa. *The 9th International Workshop on Performance Modeling, Benchmarking, and Simulation of High-Performance Computer Systems*, November 2018. `https://ieeexplore.ieee.org/document/8641556`.

[22] Nikolay Simakov and Martins D. Innu. Slurm simulator: Improving slurm scheduler performance on large hpc systems by utilization of multiple controllers and node sharing. *Proceedings of the Practice and Experience on Advanced Research Computing Article No. 25*, July 2018. `https://dl.acm.org/citation.cfm?id=3219111`.

[23] Nikolay Simakov and Martins D. Innu. *A Slurm Simulator: Implementation and Parametric Analysis*. January 2018. `https://www.researchgate.net/publication/322026152_A_Slurm_Simulator_Implementation_and_Parametric_Analysis`.

[24] SchedMD. Slurm scheduler api plugin. `https://slurm.schedmd.com/schedplugins.html`.

[25] Y. Fan. Application Checkpoint and Power Study on Large Scale Systems. In *IIT Tech. Report*, 2021.

[26] SPEAR-IIT. Cqsim: A trace-based, event-driven scheduling simulator. `https://github.com/SPEAR-IIT/CQSim`.

[27] Y. Fan, P. Rich, W. Allcock, M. Papka, and Z. Lan. Hybrid Workload Scheduling on HPC Systems. In *Advances in Computer and Network Simulation and Modelling*, 2021.

[28] Xu Yang, Zhou Zhou, Sean Wallace, Zhiling Lan, Wei Tang, Susan Coghlan, and Michael E. Papka. Integrating dynamic pricing of electricity into energy aware scheduling for hpc systems. *SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, November 2013. `https://ieeexplore.ieee.org/document/6877493`.

Appendices

Appendix A

Manual for Multi-resource Scheduling in Slurm

The following Steps for installation of Slurm and Slurm Simulator can be directly executed on Centos distribution of Linux. If not, docker needs to be installed.
If Docker is used, the following steps need to be followed before the actual installation [10].

1. Pull a centos Image from docker.
   command: sudo docker pull centos:7

2. Run this centos Image.
   command:
   sudo docker run -d -it --privileged ⟨image-id⟩ /usr/sbin/init
   sudo docker exec -it ⟨container-id⟩ /bin/bash

   Note :
   We need to run in privileged mode as we need systemctld requires CAP_SYS_ADMIN capability but docker drops this permission to non privileged containers. We can't start slurm, otherwise.

This would give access to Shell on the container running with Centos Operating System.
Steps for installation of Slurm and Slurm Simulator:

1. Install Sudo
   command:
   yum install sudo

2. Creating a user called slurm:
   command:
   useradd -d /home/slurm -ms /bin/bash slurm
   usermod -aG wheel slurm
   yum clean all

3. Installing Dependencies:

   a. Install MySQL (Maria DB)
   command:
   yum -y install mariadb-server
   yum -y install mariadb-devel
   systemctl enable mariadb
   systemctl start mariadb
   mysql_secure_installation

   b. Install Python
   command:
   yum -y install epel-release
   yum -y install python34 python34-libs python34-devel python34-numpy python34-scipy python34-pip
   yum install python34-setuptools
   sudo easy_install-3.4 pip
   pip3 install pymysql
   pip3 install pandas

   Note :
   If there is an error saying "Broken toolchain: cannot link a simple C program" then use the following command to install gcc:
   Command :
   yum install python-devel
   yum -y install gcc-c++

   Note :
   If there is an error related to installation of numpy packages while installing pandas. Do the following:
   cd /usr/lib64/python3.4/site-packages
   rm -rf numpy
   rm -rf ⟨numpy-egg-info⟩

   c. Install R
   yum -y install wget
   yum -y install R R-Rcpp R-Rcpp-devel
   yum -y install texlive-*

   d. Install R Studio
   wget https://download2.rstudio.org/server/centos6/x86_64/rstudio-server-rhel-1.2.1335-x86_64.rpm
   yum -y install rstudio-server-rhel-1.2.1335-x86_64.rpm

e. Installing Depending Packages of R
The following commands are to be run inside R. So in the command Prompt type 'R' This takes to the R Workspace where in the following commands needs to be executed
install.packages("ggplot2")
install.packages("gridExtra")
install.packages("cowplot")
install.packages("lubridate")
install.packages("rPython")
install.packages("rstudioapi")

Note :
If the install package command asks to select a domain, please select the one corresponding to the country you are located at. This would give a faster download speed.

4. Create a workspace in /home/slurm:
command:
cd /home/slurm
mkdir slurm_sim_ws
cd slurm_sim_ws

5. Copy the Slurm Simulator code from Virtual machine:
Command:
sudo docker cp slurm_simulator ⟨containerid⟩:/home/slurm/slurm_sim_ws/
cd slurm_simulator

6. Prepare Building directory
Command:
cd ..
mkdir bld_opt
cd bld_opt

7. Run slurm simulator configure command
Command:
/home/slurm/slurm_sim_ws/slurm_simulator/configure \
–prefix=/home/slurm/slurm_sim_ws/slurm_opt \
–enable-simulator –enable-pam –without-munge –enable-front-end \
–with-mysql-config=/usr/bin –disable-debug CFLAGS="-g -O3 -D NDEBUG=1"

Command to compile the Slurm code:
make -j install

8. Copy Slurm Simulator Code
sudo docker cp slurm_sim_tools ⟨containerid⟩:/home/slurm/slurm_sim_ws/

9. Create a top level directory for Simulation
   Command:
   cd /home/slurm/slurm_sim_ws
   mkdir -p /home/slurm/slurm_sim_ws/sim/micro

10. Copy existing Slurm configuration to Simulation Directory
    Command:
    cd /home/slurm/slurm_sim_ws
    slurm_sim_tools/src/cp_slurm_conf_dir.py -o -s slurm_opt \
    slurm_sim_tools/tutorials/micro_cluster/etc sim/micro/baseline

11. Populate SlurmDB
    Run Slurmdbd in foreground mode

    Commands:
    export SLURM_CONF=/home/slurm/slurm_sim_ws/sim/micro/baseline/etc/slurm.conf

    Run :

    /home/slurm/slurm_sim_ws/slurm_opt/sbin/slurmdbd

    In a separate terminal populate SlurmDB using Slurm sacctmgr utility:

    export SLURM_CONF=/home/slurm/slurm_sim_ws/sim/micro/baseline/etc/slurm.conf

    export SACCTMGR=/home/slurm/slurm_sim_ws/slurm_opt/bin/sacctmgr

    $SACCTMGR -i modify QOS set normal Priority=0
    $SACCTMGR -i modify QOS Name=supporters Priority=100
    $SACCTMGR -i add cluster Name=micro Fairshare=1 QOS=normal,supporters
    $SACCTMGR -i add account name=account1 Fairshare=100
    $SACCTMGR -i add account name=account2 Fairshare=100

    $SACCTMGR -i add user name=user1 DefaultAccount=account1 MaxSubmitJobs=1000
    $SACCTMGR -i add user name=user2 DefaultAccount=account1 MaxSubmitJobs=1000
    $SACCTMGR -i add user name=user3 DefaultAccount=account1 MaxSubmitJobs=1000
    $SACCTMGR -i add user name=user4 DefaultAccount=account2 MaxSubmitJobs=1000
    $SACCTMGR -i add user name=user5 DefaultAccount=account2 MaxSubmitJobs=1000
    $SACCTMGR -i modify user set qoslevel="normal,supporters"

    unset SLURM_CONF
    Terminate the first Slurmdbd

12. Perform Simulation
    Modify Scheduler type as SchedulerType=sched/bbsched in slurm.conf present at /home/slurm/slurm_sim_ws
    Run:
    /home/slurm/slurm_sim_ws/slurm_sim_tools/src/run_sim.py -e \
    /home/slurm/slurm_sim_ws/sim/micro/baseline/etc -s \
    /home/slurm/slurm_sim_ws/slurm_opt -d

    Note :
    Results are present in /home/slurm/slurm_sim_ws/slurm_sim_tools/src/results folder. Trace is present at /home/slurm/slurm_sim_ws/slurm_sim_tools/tutorials/results/micro_cluster/test.trace.

To brief the directory structure which is based on [10], the actual slurm code is present in slurm_directory and the simulator source code is in slurm_sim_tools folder. All the compiled Slurm code is present in bld_opt folder and as such if any change in slurm code is made, we need to run the command: " make -j install " in this folder. slurm_opt folder has the command line utilities in bin folder and Slurm control daemon, Slurm DB daemon and Slurmd daemon runnable files in sbin folder.

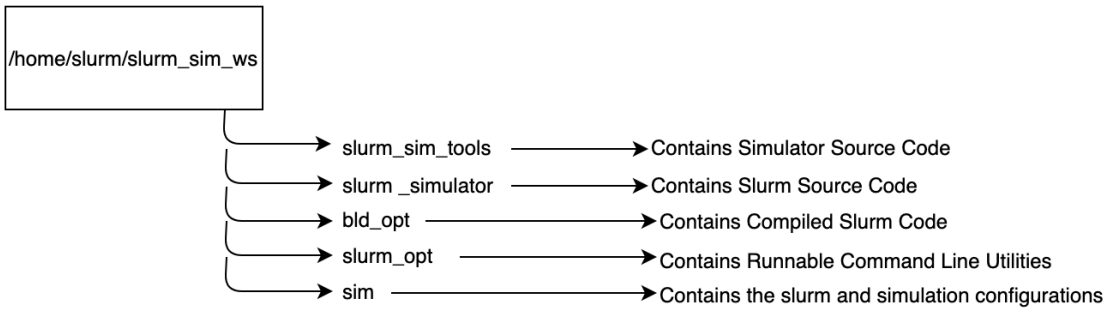

Figure A.1: *Directory Structure of Installation*