A Slurm Simulator: Implementation and Parametric Analysis

Nikolay A. Simakov[1], Martins D. Innus[1], Matthew D. Jones[1], Robert L. DeLeon[1], Joseph P. White[1], Steven M. Gallo[1], Abani K. Patra[1] and Thomas R. Furlani[1]

[1] Center for Computational Research, State University of New York, University at Buffalo, Buffalo, NY
nikolays@buffalo.edu

Abstract. Slurm is an open-source resource manager for HPC that provides high configurability for inhomogeneous resources and job scheduling. Various Slurm parametric settings can significantly influence HPC resource utilization and job wait time, however in many cases it is hard to judge how these options will affect the overall HPC resource performance. The Slurm simulator can be a very helpful tool to aid parameter selection for a particular HPC resource. Here, we report our implementation of a Slurm simulator and the impact of parameter choice on HPC resource performance. The simulator is based on a real Slurm instance with modifications to allow simulation of historical jobs and to improve the simulation speed. The simulator speed heavily depends on job composition, HPC resource size and Slurm configuration. For an 8000 cores heterogeneous cluster, we achieve about 100 times acceleration, e.g. 20 days can be simulated in 5 hours. Several parameters affecting job placement were studied. Disabling node sharing on our 8000 core cluster showed a 45% increase in the time needed to complete the same workload. For a large system (>6000 nodes) comprised of two distinct sub-clusters, two separate Slurm controllers and adding node sharing can cut waiting times nearly in half.

Keywords: HPC, SLURM, batch jobs scheduler, simulator

1 Introduction

Different fields of science and engineering exhibit different demands on computational resources. Responding to that, modern HPC clusters have significantly heterogeneous architecture, where in addition to traditional computational nodes a number of specialized nodes can be present including high memory nodes, GPU or MIC accelerated nodes and fast/large local file storage nodes. In addition, many HPC centers have various generations of these nodes operational at the same time. Managing such a facility efficiently can be challenging. Previously, centers would have a separate scheduler for each resource type, limiting users to use a specific resource even if multiple resources could serve the user request. This often led to resource imbalances where some resources had large queues while other were almost idle. To overcome this problem, Slurm workload manager [1, 15, 18] can manage all these resources under a single controller. Slurm is an open-source HPC resource manager used in a large number of HPC centers, including those which include systems on the Top500 list. There are a number of ways by which Slurm can support heterogeneous resource organization. For example, GPU nodes can be organized as a separate partition or as a part of a general partition with higher priorities for jobs requesting GPUs. There are many other adjustable parameters which influence the scheduler and in many cases their overall effect on the system performance can far from obvious. Performing a Slurm parametric optimization on a live-production system can have undesirable consequences for the end-users. Therefore, the ability to execute Slurm in a "simulated" mode, where the actual system is modelled and actual historical jobs are used as a probe can be very helpful to optimize job throughput at an HPC center.

Besides mentioned aid in organizing heterogeneous components of a cluster, the Slurm simulator can be useful in number of other parameters optimization affecting the Slurm and system performance. These can include: identification of adequate priority boost for priority users or users under deadline, optimization of parameters affecting the work of main and backfill scheduler. The simulator can also be used for testing and development of scheduling related components of Slurm.

Here we report on our Slurm simulator which is a further development of the Slurm simulator done by Lucero [9] and by Trofinoff and Benini [17]. The Slurm simulator is based on the actual Slurm source code and thus can be used to study most of the Slurm parameters. Special attention was given to simulation speed and the capability to simulate historical job load on medium and large clusters. We performed validation and analysis of the effect of several scheduler parameters on the HPC cluster performance.

2 Related Work

There are a number of job scheduler simulators for traditional HPC and Grid resources. Among them there are Bricks [16], SimGrid [8], Simbatch [3], GridSim [4] and Alea [7]. While providing a general framework for studying scheduling strategies they are only of limited interest for HPC centers looking to make specific changes to optimize their scheduling policies. Maui and Moab Scheduler has a built in scheduler simulator, however it also provides limited help for Slurm users as well [6, 10, 11].

One way to simulate the full workloads of a real scheduler is by scaling the actual job wall time and submission time. However, Slurm has many time-dependent qualities like discrete execution of schedulers (main and backfill) and priorities calculations. Therefore, conclusions drawn from such a scaled simulation may not be strictly applicable to the non-scaled system.

The Slurm Simulator originally developed by Alejandro Lucero [9] and improved by Trofinoff and Benini[17] is based on the actual Slurm source code and has the potential to be helpful for Slurm parametric optimization. Both schedules are capable of simulation of small clusters possibly with realistic and long workloads. However, our experience with them shows a low simulation speed for small clusters and an inability to handle mid-sized clusters with a realistic multiweek load. In addition, they are also based on an older version of Slurm.

3 Implementation Details

In this section, we describe modifications made to Slurm to allow its usage as its own simulator. A brief description of Slurm is given to help aid in understanding some of the concepts appearing later in the paper. For a detailed description refer to Slurm documentation [15] and the original articles[1, 18].

Slurm has a fault tolerant, multi-daemon and multi-thread design. The main daemon is the Slurm controller daemon (slurmctrld). It manages resources and allocates work on them. It also receives and services request calls from many Slurm utilities (for example squeue, sinfo and sbatch). Slurmd is a communication daemon running on every node managed by Slurm. It starts and finishes the user's jobs, per controller request, on that node and performs other node operations like execution of prolog and epilog. Slurmdbd is used for users accounts storage and is important for access rights and historical usage for fair-share calculation. Slurm control daemon (slurmctrld) has a multi-thread design where the most crucial parts are executed as separate threads with a number of health monitoring threads. For the simulation, the most important threads are ones which periodically execute the main scheduler, backfill scheduler, database synchronization and priority decay calculation. For every allocated new job Slurm spins-off a separate thread to initiate the user's job on the designated resources. A separate thread is also spun-off for each retiring job. This multi-thread design is used to achieve a high fault tolerance.

Although this multi-daemon, multi-thread design is good for the intended Slurm utilization, it has a drawback for simulation since it significantly affects performance due to the need for synchronization. For the simulator, there is no need for such a high tolerance but there is a need to perform a simulation in a reasonable amount of time. Previous Slurm simulators [9, 17] continued to use this feature of Slurm and maintain the overall Slurm workflow. The previous simulators were able to simulate small clusters with reasonable speed but had a hard time simulating real mid-size clusters. Although they occasionally were able to simulate a 256 nodes homogeneous cluster with a simulation speed of 10 simulated days per day, in many cases the simulator stalled at a certain point of the simulation without finishing it. The problem is the large number of threads which were spun-off to serve new job allocations and deallocation of resources from retired jobs. In the simulation the number of threads has a higher probability to accumulate than it does in real time due to time acceleration in the simulation. The unserved threads therefore tend to pile up and effectively hang the main process. In Slurm, it is possible to limit the thread counts, however, the yet-unspun threads will be accumulated in a special agent queue for later execution, which leads to similar problems and delays in job actual starting time and resource deallocation. To overcome this issue and obtain a higher simulation speed, we minimized the number of daemons and threads in our simulator.

Similar to previous implementations, our simulator was also developed with actual Slurm source code (a forked version of the original Slurm). The simulator was implemented as separate conditionally compiled source files with a relatively small number of modifications to the main Slurm code, allowing compilation of the code in normal and simulated modes. This layout in conjunction with Git tracking and merging capabilities would simplify simulator porting to newer Slurm versions. Unlike previous implementations, we didn't attempt to keep the entire Slurm functionality but used only the bare minimum necessary for a simulation and did not introduce an external simulation controller.

In this implementation the Slurm controller daemon, slurmctld, in addition to resource management and job scheduling, also controls the simulation. The controller daemon was serialized: all threads that are not crucial for simulation remain unstarted while the functions of crucial ones are called from the simulator main event loop in a serial way. In the simulation mode, instead of entering the main thread loop, slurmctrld enters the simulator main event loop where it remains until the end of the simulation. The simulator main event loop replaces the functionality of all crucial threads of normal Slurm and controls the execution of batch job submission, scheduling (both main and backfill), jobs priority decay calculation, synchronization with slurmdbd, jobs allocation and deallocation. The calls to respective Slurm functions are done within the loop based on expiration of their scheduled execution time, i.e. if the scheduled time is equal or less than the current simulated time. For periodically executed functions like the main and the backfill scheduler, the scheduled execution time is calculated as the previous execution time plus the sleeping time as defined in the respective Slurm agent thread. Scheduled execution time for batch job submission is an input parameter and job deallocation time is calculated as job start time plus job duration time which is also an input parameter. The simulator does not start a separate job launching and termination thread; instead it simulates the positive slurmd response which eliminates the need for the actual slurmd daemon. Thus, the simulator uses only two daemons: slurmctld and slurmdbd.

Even though slurmctrld in simulated mode is sequential, still the calls to mutex locks cost up to 40% of the total execution time, overwriting of the locking function with a dummy placeholder allows one to save that time. By default, Slurm is compiled in debug mode with assert enabled, since it is not crucial in simulation mode compilation without debugging and without asserts gains additional speed. The overall improvement in comparison with the real Slurm for the backfill scheduler is of factor of 10.

Because many Slurm functions rely on calls to time and gettimeofday functions, these functions were overwritten in the simulator to return current simulation time, which is similar to the previous simulator implementation. In addition, simulated time is allowed to tick along with the real clock and the simulator would increment the simulated time by one second at the end of the event loop if no important event happened (submission of a new job or resources deallocation).

The significant increase in backfill scheduler speed can affect the job placement. In real Slurm a single backfill cycle can take several minutes while in the simulation it takes seconds, that can lead to jobs starting much earlier in the simulation. To compensate for that we scale the execution time taken by the backfill scheduler by the speed-up factor calculated as the average ratio of execution time of real and simulated Slurm over the number of jobs it is attempting to schedule.

The overall simulation process is as follows: 1) compile Slurm in simulation mode, 2) prepare the Slurm configuration files, 3) initiate and populate the Slurm accounting system, 4) prepare the job trace file containing the description of jobs submitted to the simulator, 5) prepare the simulation configuration files, 6) run the simulation (execute slurmdbd and slurmctld) and 7) analyze the results. Our Slurm Simulator code is available at https://github.com/nsimakov/slurm_simulator. A number of utilities were developed to help and to improve usability during these steps. These utilities as well as a documentation are available at https://github.com/nsimakov/slurm_sim_tools.

4 Studied HPC Cluster systems

4.1 Micro Cluster

For the validation of the Slurm simulator, a small theoretical cluster, named Micro-cluster, was modelled using regular Slurm. Slurm has a front-end mode where a single slurmd communication daemon is used for all nodes. This mode is often used by Slurm developers for validation and developmental purposes. In this mode, most of the Slurm infrastructure is in-place and it functions in the same way as the real system; the users batch scripts are submitted through sbatch command and squeue and other utilities function in the same way. Because in this mode there is only one real compute node the users batch jobs simply consist of a sleep command with a requested duration as an argument. The utilization of this model allows us to execute real Slurm under the same workload multiple times in order to evaluate the variability intrinsic to real Slurm. The utilization of a single historical workload from the actual cluster does not offer such an option.

The characteristics of the Micro-cluster's nodes are shown in Table 1. This configuration includes two different types of compute nodes, large memory nodes and GPU nodes. This selection allows us to validate job placement based on the resource requests within the batch job. The 500 trace jobs were generated requesting either non-specific nodes or specific nodes. Jobs were distributed between five users grouped in two accounts (three users in one account and two in other account). Specific requests can be either for a CPU type or for a large size memory or for a GPU as a generic resource (GRES for GPU). The job sizes were randomly selected and varying from serial to 1-8 node parallel jobs. The wall time request was randomly selected between 5 and 30 minutes. Actual execution time was randomly selected between 0 and the requested wall time time (10% of the jobs were set to use the entire requested time and 10% to use none, to model failed jobs). The distribution of core hours assigned to users and accounts in the generated job trace is shown in Fig. 1. The network topology was each of the two types of compute node were connected to their own switch, the large-memory node and the GPU node were connected to the same switch and all of these switches connected to the top switch.

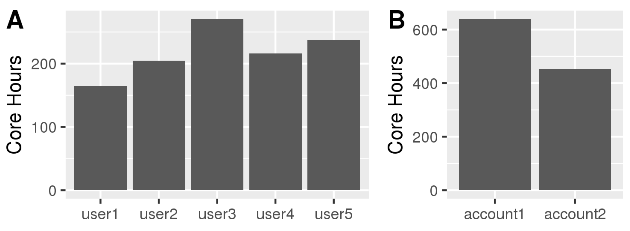

Fig. 1. (A) Core hours consumed by users on Micro cluster. (B) Core hours charged to account (users 1,2,3 belong to account 1 and users 4 and 5 belong to account 2).

Table 1. Specification of Modeled Micro Cluster.

Node Type	Number of Nodes	Cores per Node	CPU Type	RAM, GB
Compute	4	12	CPU-N	48
Compute	4	12	CPU-M	48
High Memory	1	12	CPU-G	512
GPU Compute	1	12	CPU-G	48

Table 2. Specification of Real Rush Cluster.

Node Type	Number of Nodes	Cores per Node	CPU Type	RAM
Compute	32	16	Intel E5-2660	128GB
Compute	372	12	Intel E5645	48GB
Compute	128	8	Intel L5630	24GB
Compute	128	8	Intel L5520	24GB
High Memory	8	32	Intel E7-4830	256GB
High Memory	8	32	AMD 6132HE	256GB
High Memory	2	32	Intel E7-4830	512GB
GPU Compute	26	12	Intel X5650	48GB

4.2 Rush – HPC Production Cluster

Rush cluster is a cluster used in our center for academic users. Its' specification is summarized in Table 2. The workload was simulated using historical jobs running on the actual cluster between October 4, 2016 and October 28, 2016 comprising 23.8 days. The start and end days corresponds to two system scheduled services days when all the nodes were drained down. All historical jobs were included in the simulation including cancelled jobs, because although not running they affect the job placement on the cluster. The requested jobs' resources were extracted from the Slurm accounting and actual users' batch jobs. Slurm accounting does not store complete information about resources requested by jobs, particularly it doesn't store requested CPU types or job dependencies. This information was extracted from the users' scripts, however some portion of this information can be specified as arguments to the sbatch utility and therefore was not captured. The historical job set consists of 65,000 jobs from 161 users utilizing 83 accounts consuming a total of 3,300,000 core hours. The distribution of jobs over core counts and wall time is shown in Fig. 2, these properties were obtained from the OpenXDMoD tool installed in our center[12]. The complete Slurm configuration parameters can be found in the supplementary material.

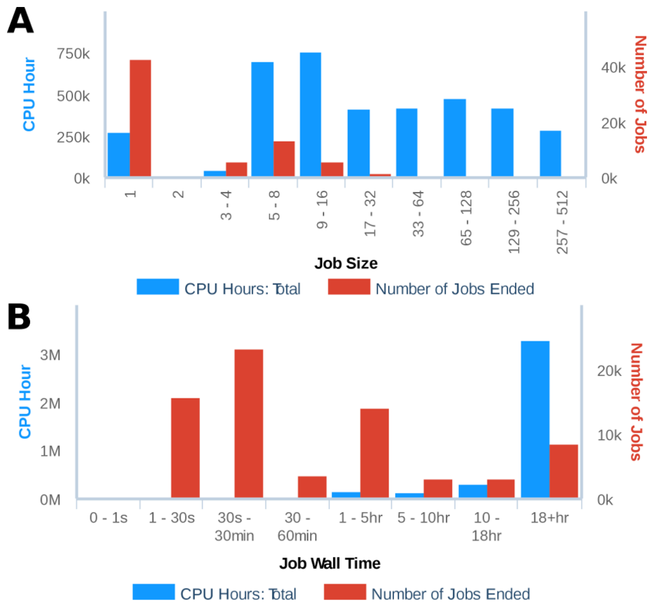

Fig. 2. Rush Cluster workload characterization for the period between October 4, 2016 and October 28, 2016. (A) Distribution of core hours and number of jobs over the number of allocated cores. (B) Distribution of core hours and number of jobs over the jobs' duration.

4.3 Stampede2 – Supercomputer

Stampede2 is a supercomputer at The University of Texas at Austin's Texas Advanced Computing Center (TACC). It consists of 4,200 Intel Xeon Phi Knights Landing (KNL) nodes and 1,736 Intel Xeon Skylake-X (SKX) nodes.

At the time of this article writing Stampede2 was still in the deployment stage and thus there was little historical workload available. Furthermore, the available workload is mainly from the initial deployment stage and it would not be representative of the workload during production stage. For the simulation, the workload was generated from the historical workload of Stampede1 supercomputer which is 6,400 node supercomputer at TACC.

The workload was generated as follows. First, a job bank was created from stampede1 historical jobs running after 2015-05-16 and submitted before 2015-08-08 (12 weeks period). All single node jobs were converted to cores-requested jobs using average CPU utilization with rounding to closest biggest core count of 1,2,4,6,8 and 12. Jobs with CPU utilization higher than 12 was considered as requested a whole node. The CPU utilization data was obtained from XDMoD[13] using TACC-Stats data[5]. Next jobs were randomly selected from job bank (without replacement). Number of selected jobs was proportional to node counts of Stampede 1 and 2. Next portion of jobs was set to be executed on KNL nodes (number of jobs proportional to portion of KNL nodes).

Two configuration for Slurm controllers were tested: one controller for all nodes and separate controllers for KNL and SKX nodes. Three options for node sharing on SKX nodes was simulated: no sharing, sharing by cores (each shared job has dedicated cores) and sharing by sockets (each shared job has all cores from same physical CPU).

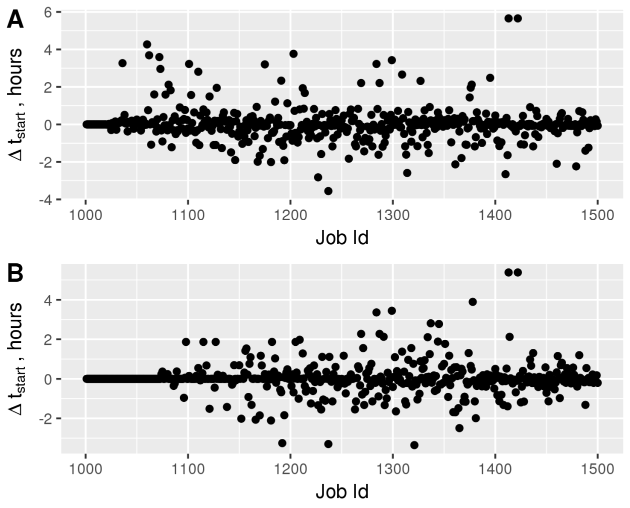

Fig. 3. Job start time difference between (A) simulated and real Slurm runs and (B) two real Slurm runs.

5 Results and Discussion

5.1 Validation

The Slurm simulator is essentially the normal Slurm resource manager with several modifications and simplifications. These include: serialization, faster performance of individual components, neglect of jobs epilog time and node failures. Therefore, for the proper use of the simulator it is important to know how these simplifications affected the simulator performance.

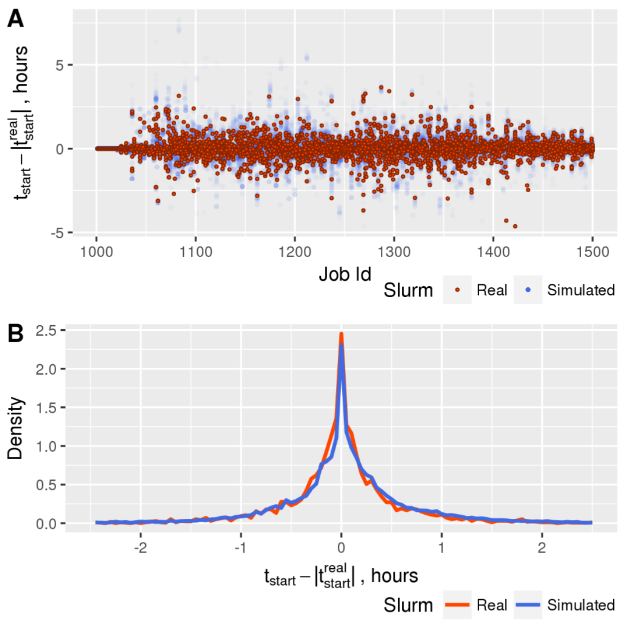

Fig. 4. Comparison of job starting times between multiple real and simulated Slurm Runs. (A) Difference between job start times and mean job start times over all real Slurm runs for real and simulated Slurm runs. (B) Distribution of start time differences from A for real and simulated Slurm runs.

Validation on Micro-Cluster. The first validation of the Slurm simulator was done against the small, 10 nodes, model cluster. The multiple simulated runs were compared to multiple runs of the regular Slurm under the same workload. The workload consisted of 500 jobs and takes 12.9 hours to complete. In all cases, Slurm preformed a proper job placement based on user requests: jobs requesting large amount of memory, GPUs or specific CPU type executed on nodes with suitable characteristics.

The job start time was used to characterize the differences between the Slurm runs. Other characteristics like waiting times and system utilization are essentially derivatives of the individual job start times. It is interesting to compare multiple runs of Slurm in both simulated and normal mode. The start time difference between a single simulated and a real Slurm run is shown in Fig. 3.A. For that particular runs, the mean start time difference is 0.8 minutes with a standard deviation of 57.0 minutes and number of outliers exceeding 4 hours. For a 12.9 hour workload, such variability is comparable to the differences between two real Slurm runs with similar initial conditions (Fig. 3.B). The mean difference for that real Slurm runs is 1.4 minutes with a standard deviation of 50.3 minutes. In Slurm, many routings are executed regularly with a sleep cycle between the executions. Among these routings are main and backfill schedulers and priority decay calculations. Because of the system jitter and varying workload, the execution time of these functions vary leading to differences in relative function start-up times. This can create an opportunity hole for some jobs, which can be placed on a resource earlier and eventually lead to different job placement for other jobs leading to relatively large differences in scheduling from run to run.

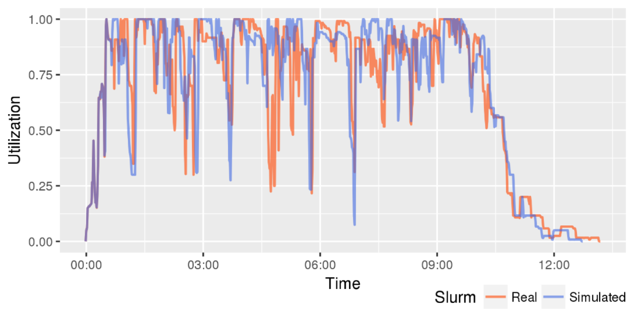

Fig. 5. Comparison of Micro-cluster utilization between single real and simulated Slurm runs, aggregation was done over 1 minute period.

The multiple Slurm runs were obtained by varying the time between the Slurm controller boot time and the first job submission time, the submission time of other jobs relative to the first job remains the same. This serves as initial seed and varies the initial calls to the main and the backfill schedulers relative to the first job in the workload. To study the variability, seven real Slurm runs and 120 simulated runs were performed. The job start times were compared to the average job start times over all real Slurm time.

The start time differences for all real and simulated jobs is shown in Fig. 4.A along with its distribution in Fig. 4.B. The difference between mean values for real and simulated Slurm distribution is 3.2 minutes (on average the simulated Slurm jobs start later) and the standard deviation is 36.9 and 42.5 minutes for real and simulated Slurm runs respectively. There are a number of jobs in both simulated and real Slurm exceed 3 hours. For each run the mean and standard deviation of all job start time differences were calculated. The Student t-test and the Kolmogorov-Smirnov test were used to compare the simulated and the real runs. The tests showed that although the means cannot be distinguished the standard deviations are different (p-values < 0.01). Some of this 15% difference in standard deviation between real and simulated runs can be attributed to a smaller number of independent runs there (7). There is an interesting time dependence exhibited during the runs. In the beginning of the run the deviation is small and starts to grow as the accumulation in job placement varies. However instead of growing indefinitely it reaches certain level and stays there. This is probably caused by a steady stream of new jobs and the retirement of old jobs resulting in historical job placement having a decaying memory when it affects the allocation of new jobs. As for the starting times the system utilization (Fig. 5) and the job priorities (Fig. 6) change over time in a similar fashion between the normal and the simulated runs. For the Slurm simulator it took 17 seconds to complete this 12.9 hours workload, that is the simulation speed for the Micro-cluster was 112 simulated days per hour.

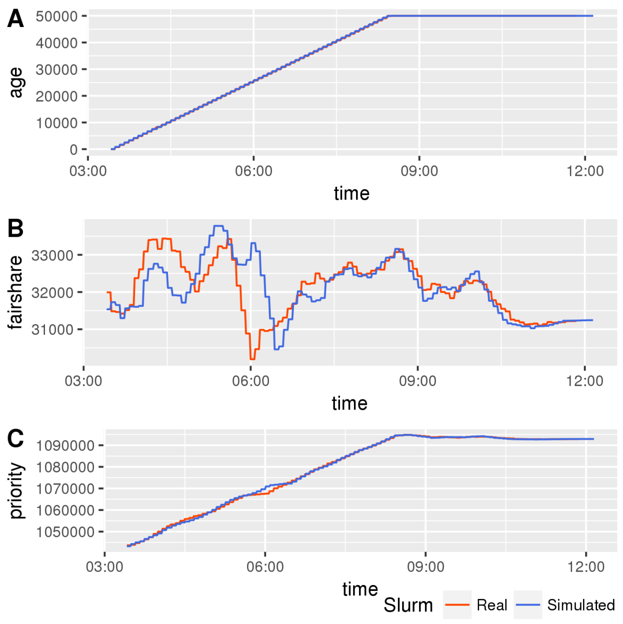

Fig. 6. Comparison of job priority (C) and its' components, age factor (A) and fair-share factor (B), change over time between single real and simulated Slurm runs for one of the long pending jobs. The fair-share priority factor has a step-like form because these values are recalculated every 5 minutes.

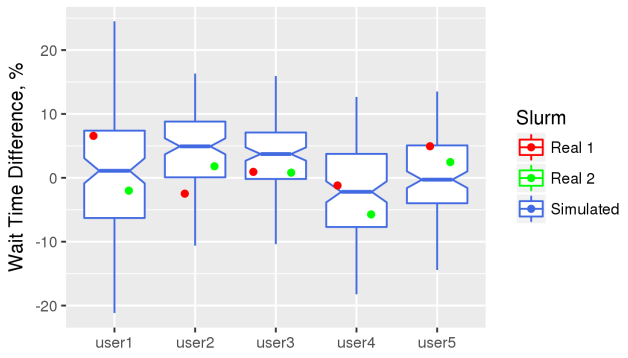

Fig. 7. Effect of increasing fair-share priority factor weight by 20% on waiting time for each user. Change in waiting time for simulated runs shown as boxplot where the lower and higher sides corresponds to first and third quartiles, the horizontal bar in the box is median and the whiskers extends to the furthest value not exceeded 1.5 of inter-quartile range. The wait time differences from two sets of real Slurm runs are shown as red and green points.

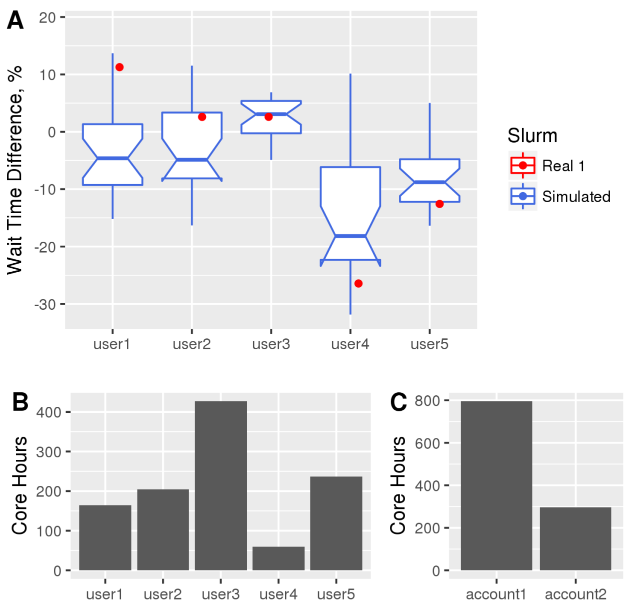

Fig. 8. (A) Same as Fig. 7 but with modified workload shown in (B) and (C).

Often instead of absolute prediction there is an interest in predicting a change upon modification of some parameter. In order to study this, fair-share priority factor weight was increased by 20% and the simulator and the normal Slurm was rerun using the new value of this parameter. The fair-share priority factor alters the job's priority based upon the user for the resource allocation order in the main scheduler and in the backfill scheduler. The fair-share factor takes into consideration the individual user's previous resource usage as well as the resource usage of all other users from same account. The fair-share factor allows users with lower previous usage to receive an allocation to resource faster than users with higher usage. The resulting change in waiting times is shown in Fig. 7. The effect of 20% increase in fair-share weight is small and does not exceed 5% of jobs wait time. This is due to similar resource utilization by all users (Fig. 1.A). The first three users belong to account one and the last two users belong to account two. The resource utilization by account one users is 30% higher than that by users from account two users (Fig. 1.B). The simulation predicts that on average users from account one would have longer waiting times while users from account two shorter. However, the variation in predicted values is high and a significant portion of the distributions lays on both sides of zero. The wait time differences from two sets of real Slurm runs stays within predicted values (Fig. 7). Because of relatively small difference in resource utilization by all users and associated accounts the resulting difference is small as well. To produce a more pronounce difference a second experiment was performed where 70% of user 4 was reassigned to user 3 (Fig. 7.B and C). The simulation predicts that user 4 will have 18% smaller waiting time and more than 75% of distribution for user 4 from account two lays below zero. Indeed, in real Slurm run users 4 and

5 have significantly lower wait time (Fig. 7..A), however unlike the mean values from simulation the user 1 and 2 are higher. Nevertheless, the values from real Slurm are within simulator predicted values.

The model Micro cluster allowed us to study the simulator in comparison to multiple real Slurm runs. The simulator predicted mean start times statistically undistinguishable from actual Slurm start times with a slightly larger standard deviation. The simulator properly allocates resources and updates jobs priority values. The simulator predicted ranges for waiting time change upon modification of fair-share priority factors similar to those of the real Slurm runs.

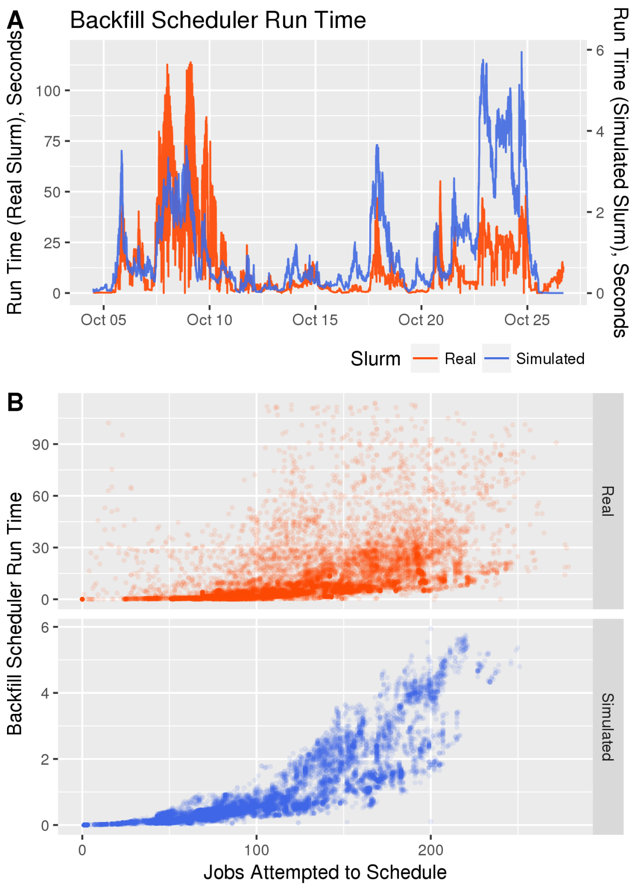

Fig. 9. (A) Backfill scheduler execution time historical variation in real Slurm along used workload. (B) Dependency of backfill scheduler execution time on the number of jobs attempted to schedule for single historical and simulated runs

5.2 Validation on the Rush Cluster

In order to evaluate how the simulator compares with real Slurm executed on a real system we performed a simulation of using an actual historical workload from our cluster. As opposed to the modeled Micro-cluster here we only have one historical result. In the simulation, we do not take into consideration the cluster usage prior to the beginning of the simulation or node failures. Both have a significant effect on the job scheduling as the first one affects the job priorities calculation and the second makes the effective simulated cluster bigger. Node failures is a significant component of aging clusters and can be added to the simulator in the future.

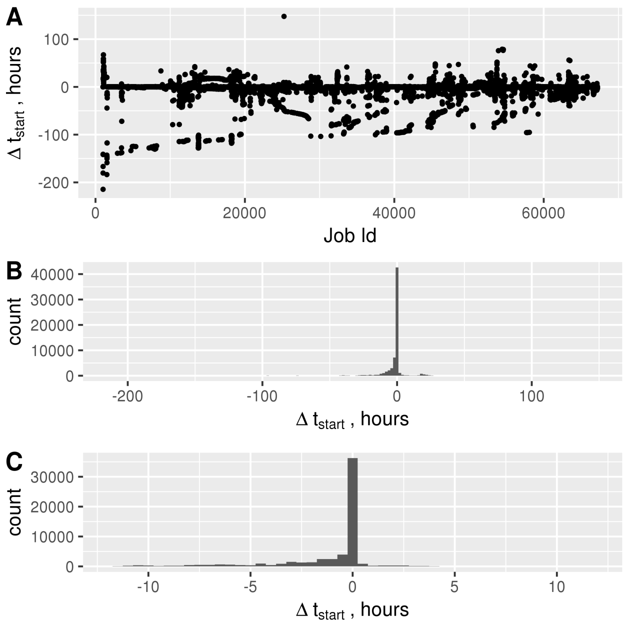

Fig. 10. (A) Difference in jobs start time between real and simulated Slurm runs. (B) Distribution of that start time difference. (C) Same as B, zoomed to -12 to 12 hours region.

In normal Slurm running on medium to large clusters the backfill scheduler takes a significant amount of time requiring more than several minutes per single loop. Due to simplification in the simulator the backfill scheduler runs more than 10 times faster. An artificially faster scheduler can affect the job placement and produce significant deviation from the real Slurm performance. Although the backfill scheduler execution time for real and simulated Slurm have similar patterns (Fig. 9.A) the actual Slurm has much more noise than the simulated one (Fig. 9.B). Such variability is because real Slurm needs to spin-off a number of threads to serve various users requests, jobs allocation and deallocation. These threads can slow the backfill scheduler due to thread locking and the decreased amount of CPU time available to the scheduler thread. Slurm in simulated mode does not have such interruptions. Ideally a good simulator would incorporate such effects. In this article, we simply scale the backfill scheduler loop by the speed-up factor. In the real Slurm the dependency of the backfill scheduler run time on the number of jobs attempted to schedule is non-linear. However, we were not able to produce a better model than simple scaling from the available data. Collecting a wider range of Slurm performance metrics might help to improve the backfill scheduler model.

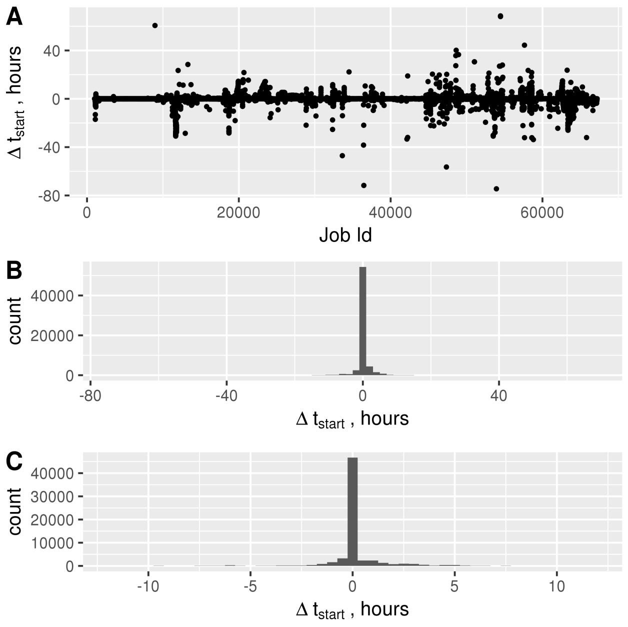

Fig. 11. (A) Difference in jobs start time between two simulated Slurm runs. (B) Distribution of that start time difference. (C) Same as B, zoomed to -12 to 12 hours region.

The difference of jobs start time between historical and simulated Slurm runs is shown in Fig. 10. The average difference in jobs start time between historical and simulated Slurm run was -2.4 hours (on average simulated jobs start earlier) with a standard deviation of 12.0 hours. There is a large number of outliers with time difference more than 4 days (. The outliers' deviation from zero decreases over time, this is due to neglect of the initial resource usage. Because in fair-share the previous utilization contribution decays over time, the influence of initial historical usage diminished resulting in smaller deviation further from the starting point. It is interesting to compare this standard deviation with one between two simulated runs. The difference of job start times between two simulated runs is shown in Fig. 11. It has a mean of 1.3 minutes and a standard deviation of 2.5 hours. Fig. 12 shows the whole resource utilization. Both historical and simulated results show a similar pattern. Interestingly in the beginning the simulated run has a higher utilization. Probably the omission of initial historical usage by users allows the simulator to allocate resources more efficiently. Throughout the middle part of the timeline there are a number places where the historical run has a higher utilization than the simulated runs. Most likely some of the job resources requests were specifically tailored by users to fit the gaps in the cluster at that time and in the simulator these gaps were different preventing the job placement.

Given the differences in initial conditions and lack of node failures modeling the simulator shows a reasonable approximation of historical workflows. Combined with results from the Micro-cluster the simulator can be used for studies of the effects of various Slurm parameters on the system performance.

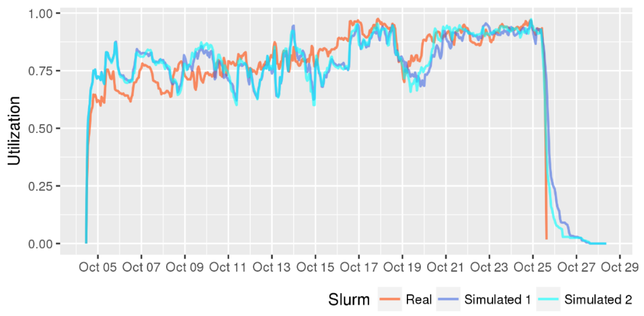

Fig. 12. Comparison of resource utilization between historical Slurm run and two simulated runs.

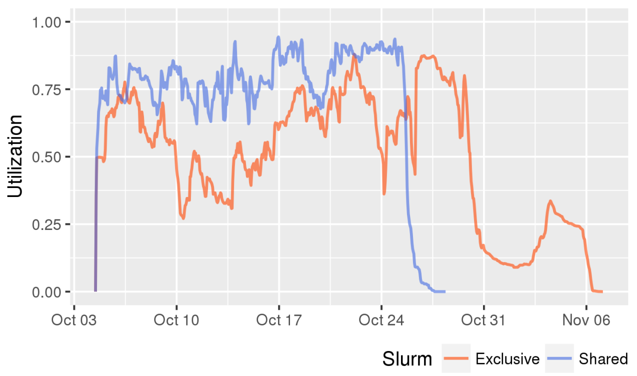

Fig. 13. Comparison of resource utilization for exclusive and shared node modes.

5.3 Study of Various Parameters

Node Sharing. Many smaller HPC centers, like ours, have a large portion of serial jobs and jobs which do not fully utilize an entire node (Fig. 1.A). Allowing multiples of such jobs to be executed on the same node increases the overall system throughput. This is often referred to as node sharing mode as opposed to node exclusive mode where the whole node is allocated to a single job. It was shown that many applications running on a fraction of a node's cores have less than 5% decrease in their performance when running in none sharing mode [14]. In certain cases of large parallel jobs, the jobs can actually complement each over in sub-system usage leading to faster execution time

[2]. In order to quantitatively determine the increase in the overall system gained by node sharing the simulator was run in node sharing and exclusive modes.

The exclusive mode takes 10.8 more days (45% more time) to complete the same workload (Fig. 13). The average increase in waiting time is 5.1 days with a standard deviation of 6.6 days. The 45% increase in time to complete the same load can be translated into the need to have a 45% larger cluster to serve the same workload. This is a major savings even after considering a potential 3% slow down by some jobs. Long wait times are a significant difficulty and users could potentially adjust their computational work load to decrease the wait time potentially endangering the quality of their work. In other words, if we would switch back to exclusive mode, the users would adjust by decreasing their usage and in this way the benefit of shared mode can be rephrased as it allows users to do more than 40% more computational job. Therefore, there is a good reason to have shared mode enabled and the users preferred to use exclusive mode still can ask for the whole node.

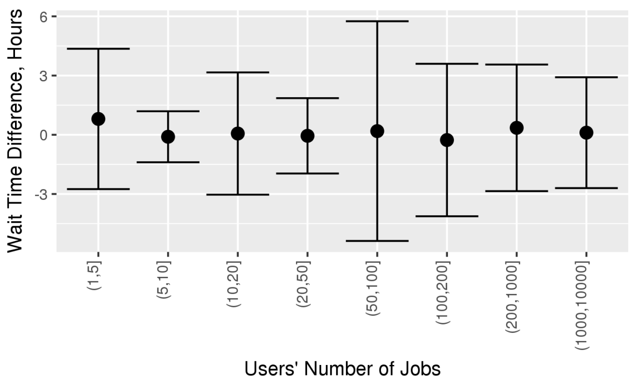

Fig. 14. Wait time difference between bf_max_job_user parameter set 10 and 20 for users grouped by their total number of jobs. bf_max_job_user parameter define maximal number of user's job considered by backfill scheduler. The mean values shown as points and error bar correspond to standard deviation.

Maximal Number of User's Jobs Considered by Backfill Scheduler for Scheduling. The backfill scheduler allows resource allocation to jobs which do not affect the start time of higher priority jobs. In many centers a large portion of the jobs are scheduled by the backfill scheduler. This scheduler scans through pending jobs and attempts to allocate resources for them without affecting the highest priority jobs. Such a single scan can take more than several minutes and due to multiple operations requiring controller locking can significantly affect the controller responsiveness. Extremely long run times can also lead to a decreased number of jobs being scheduled. Therefore, there should be a good balance between execution time of single scan of the backfill scheduler and the quality of its scheduling. There are a number of parameters which affect the scheduler performance. Here we will consider the bf_max_job_user parameter which defines the maximal number of user's jobs considered by the backfill scheduler for scheduling. In the reference simulation bf_max_job_user was set to 20 user's job, we will show how the performance of the cluster would be affected by decreasing this parameter to 10 user's jobs.

The simulation showed only small increase in time needed to complete the workload, namely 40 minutes or 0.1% from the referenced time. The mean wait time is 8 minutes slower and the standard deviation of the wait time differences is 3 hours. Surprisingly, there is no strong dependency of the job wait time on the total number of jobs submitted by the user during the simulated period (Fig. 14). With this small decrease in resource utilization and increase user's wait time, there is a 25% decrease in the number of jobs to consider for scheduling which leads to a 30% decrease in backfill scheduler run time. Therefore, a reasonable decrease of the maximal number of user's jobs considered by a backfill scheduler for scheduling have small average effect on job placement and offers significant improvement in backfill scheduler run time. This can be a good choice for systems where backfill scheduler run time is an issue.

5.4 Stampede2: Node Sharing and Multiple Slurm Controllers

Stampede2 is a new supercomputer at The University of Texas at Austin's Texas Advanced Computing Center (TACC). We chose to model it to show the capability of the simulator to handle large systems with a realistic workload and to illustrate the simulator potential for optimization of the Slurm scheduler for such systems.

Stampede2 is composed of two distinct parts: the first consists of Intel Xeon Phi Knights Landing (KNL) nodes (4,200 nodes) and the second consists of Intel Xeon Skylake-X (SKX) nodes (1,736 nodes). Because of the architectural differences between these nodes, users most likely would specifically request a particular node type for their jobs. This allows us to consider using separate controllers for each part to reduce the load on the controller and thus potentially improve performance. The drawback of separate controllers is the extra labor associated with maintenance of an additional Slurm controller. Therefore, it would be useful to estimate the benefits of separate controllers first prior to the actual implementation.

Because of the large number of cores per node on SKX nodes (48 cores per node) the benefits of node sharing can be substantial. However, Stampede2 is a large system and enabling node sharing will increase the number of consumable resources (cores and memory are needed to be tracked now instead of only nodes) and would lead to a significantly higher computational load on the backfill scheduler. Such an increase could potentially render the entire system inefficient or inoperable. For this reason, the Slurm documentation does not recommend node sharing for large systems. Therefore, it is of interest to compare the simulated performance of Slurm without node-sharing with different node-sharing schemes enabled to estimate the feasibility of node sharing on the SLX nodes. In this article two configurations for node sharing were modeled: sharing by sockets, where all the cores from the same physical CPU are assignable to a single job and sharing by cores, where each core can be allocated individually.

Table 3. Simulated waiting times on Stampede2. Mean wait hours weighted by node hours increases the contribution of large jobs to the mean.

Controller	Node Sharing on SKX Nodes	Wait Hours, Mean	Wait Hours, Mean Weighted by Node Hours
Jobs on SKX Nodes			
Single	no sharing	10.9 (0%)	17.0 (0%)
	sharing by sockets	8.2 (-25%)	15.5 (-9%)
	sharing by cores	8.2 (-24%)	15.5 (-9%)
Separate	no sharing	7.1 (-35%)	15.0 (-12%)
	sharing by sockets	5.3 (-51%)	13.8 (-19%)
	sharing by cores	5.5 (-49%)	13.9 (-18%)
Jobs on KNL Nodes			
Single	no sharing	8.6 (0%)	9.2 (0%)
	sharing by sockets	7.2 (-16%)	9.2 (-1%)
	sharing by cores	7.3 (-15%)	9.1 (-1%)
Separate	no sharing	8.2 (-4%)	9.4 (2%)

The mean wait time for different configurations is summarized in Table 3. For SLX jobs, dedicating separate controller resulted in a 35% improvement in waiting time. In this case the smaller number of jobs per controller and smaller number of resources to track (3.4 times less nodes) for each controller lead to much better performance by the backfill scheduler. Indeed, the backfill scheduler with only a single controller reached the run time limit in 70% of all runs. A separate dedicated controller reached the limit on only 32% of the runs. Interestingly, that there is almost no benefit for KNL jobs, this is probably due to a much smaller decrease in nodes for the KNL controller than for SKX node controller (1.4 times vs 3.4 times).

Another 22%-25% improvement in wait time can be achieved by allowing node sharing. Surprisingly sharing by sockets shows a marginally shorter waiting time than node sharing by cores. This may be due to the smaller number of tractable resources or it may be an artifact of the workload composition. The percentage of backfill scheduler runs which hit the run time limits are very similar in both cases (30% for sharing by sockets and 32% for sharing by cores).

These simulations show that it is possible to enable node sharing on SLX nodes of Stampede 2 to improve wait time., The best performance is achieved with node sharing and with dedicated controllers for the SLX and KNL nodes. The difference between allocations by cores and sockets is very small and it is problematic whether it is more beneficial to do node sharing by cores or by sockets. For this system, we only ran a single simulation for each configuration due to time constraints. Similar to the micro-cluster, it is expected that there should be a variation in wait time and ideally multiple simulations must be done to generate reasonable statistics. Therefore, these findings should be regarded as preliminary.

5.5 Simulation Speed

The simulator speed heavily depends on the cluster size, workload and Slurm configuration. For a small 120 core cluster the simulation speed was 112 simulated days per hour while for a medium 8000 core cluster it was in the range of 0.8 to 17.3 simulated days per hour depending on the Slurm configuration. In the reference configuration, it was 5.4 simulated days per hour. In exclusive node job allocation mode it was 0.8 simulated days per hour. With a smaller maximal number of user's job considered by the backfill scheduler, it was 17.3 simulated days per hour. For the tested large system, it was around 1 simulated day per hour. In most cases the simulation speed correlates with the backfill scheduler run time and can serve as an indicator of whether the backfill scheduler run time needs to be optimized.

6 Conclusions

A new Slurm simulator was developed capable of simulation of large clusters with a simulation speed of multiple days per hour. Its validity was established by a comparison with actual Slurm runs which showed similar mean values for job start times between the simulated and actual data with a slightly larger standard deviation for the simulation results. We have exercised this simulator in studying a number of Slurm parameters that affect system utilization and throughput such as fair share policy, maximum number of user jobs considered for backfill, and node sharing policy. As expected fair share policy alters job priorities and start times but in a non-trivial fashion. Decreasing the maximal number of user's job considered by the backfill scheduler from 20 to 10 was found to have a minimal effect on average scheduling and serves to decrease the backfill scheduler run time by 30%. The simulation study of node sharing on our cluster showed a 45% increase in the time needed to complete the workload in exclusive mode compared to shared mode. An initial analysis of Stampede2 supercomputer scheduling shows that it can benefit from separate Slurm controllers and node sharing.

7 Acknowledgments

This work was supported by the National Science Foundation under awards OCI 1025159, 1203560, and is currently supported by award ACI 1445806 for the XD metrics service for high performance computing systems.

8 References

[1] Balle, S.M. and Palermo, D.J. 2007. Enhancing an Open Source Resource Manager with Multi-core/Multi-threaded Support. *Job Scheduling Strategies for Parallel Processing*. Springer Berlin Heidelberg. 37–50.

[2] Breslow, A.D., Porter, L., Tiwari, A., Laurenzano, M., Carrington, L., Tullsen, D.M.

and Snavely, A.E. 2016. The case for colocation of high performance computing workloads. *Concurrency and Computation: Practice and Experience*. 28, 2 (Feb. 2016), 232–251.

[3] Caniou, Y. and Gay, J.-S. 2009. Simbatch: An API for Simulating and Predicting the Performance of Parallel Resources Managed by Batch Systems. Springer, Berlin, Heidelberg. 223–234.

[4] Casanova, H., Giersch, A., Legrand, A., Quinson, M. and Suter, F. 2014. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*. 74, 10 (Oct. 2014), 2899–2917.

[5] Evans, T., Barth, W.L., Browne, J.C., DeLeon, R.L., Furlani, T.R., Gallo, S.M., Jones, M.D. and Patra, A.K. 2014. Comprehensive Resource Use Monitoring for HPC Systems with TACC Stats. *2014 First International Workshop on HPC User Support Tools* (Nov. 2014), 13–21.

[6] Jackson, D.B., Jackson, H.L. and Snell, Q.O. Simulation based HPC workload analysis. *Proceedings 15th International Parallel and Distributed Processing Symposium. IPDPS 2001* 8.

[7] Klusácek, D. and Rudová, H. 2010. Alea 2: job scheduling simulator. *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques* (2010), 61.

[8] Legrand, A., Marchal, L. and Casanova, H. 2003. Scheduling Distributed Applications: The SimGrid Simulation Framework. *Proceedings of the 3st International Symposium on Cluster Computing and the Grid* (Washington, DC, USA, 2003), 138--.

[9] Lucero, A. 2011. Slurm Simulator. *Slurm User Group Meeting* (2011).

[10] Maui Scheduler: *http://www.adaptivecomputing.com/products/open-source/maui/*. Accessed: 2017-04-03.

[11] Moab HPC Suite: *http://www.adaptivecomputing.com/products/hpc-products/moab-hpc-basic-edition/*. Accessed: 2017-04-03.

[12] Palmer, J.T. et al. 2015. Open XDMoD: A Tool for the Comprehensive Management of High-Performance Computing Resources. *Computing in Science & Engineering*. 17, 4 (Jul. 2015), 52–62.

[13] Palmer, J.T. et al. 2015. Open XDMoD: A Tool for the Comprehensive Management of High-Performance Computing Resources. *Computing in Science & Engineering*. 17, 4 (Jul. 2015), 52–62.

[14] Simakov, N.A., Sperhac, J., Yearke, T., Rathsam, R., Palmer, J.T., DeLeon, R.L., White, J.P., Furlani, T.R., Innus, M., Gallo, S.M., Jones, M.D., Patra, A. and Plessinger, B.D. 2016. A Quantitative Analysis of Node Sharing on HPC Clusters Using XDMoD Application Kernels. *Proceedings of the XSEDE16 on Diversity, Big Data, and Science at Scale - XSEDE16* (New York, New York, USA, 2016), 1–8.

[15] Slurm Workload Manager: *https://slurm.schedmd.com/*. Accessed: 2017-03-24.

[16] Takefusa, A., Matsuoka, S., Aida, K., Nakada, H. and Nagashima, U. 1999. *Proceedings of the 8th IEEE International Symposium on High-Performance Distributed Computing Augustn 3-6, 1999.* IEEE Computer Society.

[17] Trofinoff, S. and Benini, M. 2015. Using and Modifying the BSC Slurm Workload Simulator. *Slurm User Group Meeting* (2015).

[18] Yoo, A.B., Jette, M.A. and Grondona, M. 2003. SLURM: Simple Linux Utility for Resource Management. Springer, Berlin, Heidelberg. 44–60.