Slurm Simulator: Improving Slurm Scheduler Performance on Large HPC systems by Utilization of Multiple Controllers and Node Sharing

Nikolay A. Simakov
SUNY University at Buffalo
Buffalo, New York
nikolays@buffalo.edu

Robert L. DeLeon
SUNY University at Buffalo
Buffalo, New York
rldeleon@buffalo.edu

Martins D. Innus
SUNY University at Buffalo
Buffalo, New York
minnus@buffalo.edu

Matthew D. Jones
SUNY University at Buffalo
Buffalo, New York
jonesm@buffalo.edu

Joseph P. White
SUNY University at Buffalo
Buffalo, New York
jpwhite4@buffalo.edu

Steven M. Gallo
SUNY University at Buffalo
Buffalo, New York
smgallo@buffalo.edu

Abani K. Patra
SUNY University at Buffalo
Buffalo, New York
abani@buffalo.edu

Thomas R. Furlani
SUNY University at Buffalo
Buffalo, New York
furlani@buffalo.edu

ABSTRACT

A Slurm simulator was used to study the potential benefits of using multiple Slurm controllers and node-sharing on the TACC Stampede 2 system. Splitting a large cluster into smaller sub-clusters with separate Slurm controllers can offer better scheduling performance and better responsiveness due to an increased computational capability which increases the backfill scheduler efficiency. The disadvantage is additional hardware, more maintenance and an incapability to run jobs across the sub-clusters. Node sharing can increase system throughput by allowing several sub-node jobs to be executed on the same node. However, node sharing is more computationally demanding and might not be advantageous on larger systems. The Slurm simulator allows an estimation of the potential benefits from these configurations and provides information on the advantages to be expected from such a configuration deployment. In this work, multiple Slurm controllers and node-sharing were tested on a TACC Stampede 2 system consisting of two distinct node types: 4,200 Intel Xeon Phi Knights Landing (KNL) nodes and 1,736 Intel Xeon Skylake-X (SLX) nodes. For this system utilization of separate controllers for KNL and SLX nodes with node sharing allowed on SLX nodes resulted in a 40% reduction in waiting times for jobs executed on the SLX nodes. This improvement can be attributed to the better performance of the backfill scheduler. It scheduled 30% more SLX jobs, has a 30% reduction in the fraction of cycles that hit the time-limit and nearly doubles the jobs scheduling attempts.

PEARC '18, July 22–26, 2018, Pittsburgh, PA, USA

ACM ISBN 978-1-4503-6446-1/18/07…$15.00
https://doi.org/10.1145/3219104.3219111

CCS CONCEPTS

• **Computer systems organization** → **Distributed architectures**; • **Computing methodologies** → **Planning and scheduling**; *Modeling and simulation*; • **Software and its engineering** → *Real-time schedulability*;

KEYWORDS

HPC, Scheduling, Workload, Simulation

ACM Reference Format:
Nikolay A. Simakov, Robert L. DeLeon, Martins D. Innus, Matthew D. Jones, Joseph P. White, Steven M. Gallo, Abani K. Patra, and Thomas R. Furlani. 2018. Slurm Simulator: Improving Slurm Scheduler Performance on Large HPC systems by Utilization of Multiple Controllers and Node Sharing. In *PEARC '18: Practice and Experience in Advanced Research Computing, July 22–26, 2018, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3219104.3219111

1 INTRODUCTION

Slurm is an open-source HPC resource management and job scheduling system [1, 8, 10]. It is highly configurable and it is used on a large variety of HPC systems ranging from small group clusters to large-scale supercomputers. A single controller is capable of handling a heterogeneous resource comprised of multiple generations of compute nodes, GPU nodes, large memory nodes and others resources. The Slurm scheduler is also capable of supporting different priority policies including high priority users, deadline priority boosting and cycle scavengers. Such flexibility allows for the accommodation of different requirements, however, an improper configuration can lead to unintended results or subpar performance such as long waiting times or the inability to allocate requested resources. In order to facilitate the search for optimal parameters while avoiding jeopardizing the actual system, we have developed the Slurm simulator.

Our Slurm simulator [6] is a modification to the actual Slurm code. Due to this, the simulator supports most scheduling related options and most importantly it inherits the actual Slurm behavior. The simulator can perform a several months workload simulation in several real days for a large HPC system. This way the simulator allows the study of the actual Slurm scheduler on an accelerated time frame.

In our previous work [6] we reported a simulator implementation and vigorous testing on three different HPC systems including an initial simulation of the TACC Stampede 2 system. Unfortunately, due to the way the workload was set up, the simulated system configurations had different job size (i.e. the number of cores) priority factors than the values used in the job mix that was actually run. This additional variability complicates the analysis and the attribution of effects to a particular change. Therefore in the present work, we corrected for that problem, added a more detailed analysis of the simulations and performed additional simulations to supplement the findings in order to improve the statistical confidence.

Stampede 2 is a new supercomputer at the Texas Advanced Computing Center (TACC) which is currently (as of March 2018) the biggest HPC computing resource in the XSEDE organization. It consists of 4,200 Intel Xeon Phi Knights Landing (KNL) compute nodes (68 cores per node) and 1,736 Intel Xeon Skylake-X (SLX) compute nodes (48 cores per node). The significant difference in KNL and SLX architectures suggests a natural separation of the whole cluster to KNL and SLX sub-clusters. Because it is unlikely that users would run a single computational job across this partition it is possible to dedicate a separate Slurm controller for each cluster section. It is very important to realize that the Slurm backfill scheduler can become a bottleneck for a large system. Hence dedicated controllers for KNL and SLX sections can increase the computational power of the backfill scheduler and lead to a more responsive and efficient scheduler. However, an additional controller would require more work from system administrators and can result in additional operational complexity. Therefore prior subscribing to this additional work it is constructive to use the Slurm simulator to determine if the potential benefits of separate controllers are justified for this particular system.

In the past, because of the computational workload on the Slurm backfill scheduler, node sharing was rarely used on large systems. During node sharing, multiple jobs from the same or different users are allowed to be executed on the same node where each job has its own dedicated cores. Previous studies shows that such jobs have a small influence on each other [2, 5, 9]. Node sharing can increase the system throughput by placing serial and low-core count parallel jobs on the same node. This, however, might not work out well in reality because node-sharing enabled scheduling is a much more computationally demanding task due to a significant increase in allocatable resources. Cores and memory must be tracked now instead of only nodes. This can render the whole system inefficient or inoperable. For example, the Slurm controller might spend a lot of time calculating potential scheduling rather than doing the actual scheduling. For this reason, Slurm Large Cluster Administration Guide [7] recommends avoiding node sharing for a large system. However, computers are getting faster and Slurm is getting better to the extent that it is worthwhile to consider if it is possible to take advantage of the considerable increase in throughput that node share potentially can provide. A full Slurm scheduling simulation of the Stampede 2 workload will allow us to determine if node sharing is feasible for this particular system, what will be the quantitative benefit and what will be the most optimal parameters for the scheduler.

In this work we continued to study the potential benefits of the multiple controller configurations with node sharing enabled for KNL and SLX nodes. Our preliminary simulation showed [6] that the waiting time improved by nearly a factor of two in comparison with a single controller without node-sharing. Here we demonstrate the reasons for this improvement and provide additional simulations to improve the statistical power of the results.

2 METHODS

2.1 Slurm Simulator

In this section the Slurm simulator implementation is briefly described; for more details refer to our initial Slurm Simulator article [6]. The Slurm simulator is implemented within the actual Slurm code and building Slurm in simulation mode is requested during configuration by specifying the corresponding option. A number of python and R utilities were developed in order to automate Slurm launching, workload generation and for the analysis of the results.

The modifications to Slurm to create the can be largely split into the following categories: simulating calls to timing functions to allow the simulation of the desired time period, disabling unnecessary Slurm functionality, inputting workload and modifications to allow the time accelerated simulations.

In order to facilitate the simulation of the desired time period, time returning standard library functions, namely `time` and `gettimeofday`, were re-implemented within the Slurm code. Because in the case of name conflict the application implementation has a higher priority than the shared library implementation, the simulator versions of `time` and `gettimeofday` will be executed instead of their namesakes from the standard library. In simulator mode the re-implemented `time` and `gettimeofday` functions return a shifted time value. The initial time shift is set in a manner such that the initial simulation time will be a little bit earlier than the first job submission time. This way Slurm operates in simulated time and the simulated clock ticks at the same rate as the real clock. This allows for the proper handling of all time-dependent functionality of Slurm, including time-outs.

Slurm utilizes a multi-thread design to achieve high fault tolerance. For the simulator, there is no need for the high tolerance and we only need the scheduling related part of Slurm. Therefore, in simulation mode, a large portion of Slurm is disabled or bypassed. This includes health monitoring threads, state backup, communication with compute nodes and many other functions. Most Slurm installations use two schedulers: a priority based main scheduler (MS) and a backfill scheduler (BF). The priority-based main scheduler loops through the priority ordered job list and attempts to schedule jobs, it will quit from the loop at the first failure to schedule a job. The backfill scheduler will allocate resources for jobs which would not affect the scheduling of top priority jobs. In normal mode, MS and BF schedulers would be executed by separate threads. In the simulator mode, functions normally executed in separate threads, are called from the main simulator event loop in a serial fashion. This

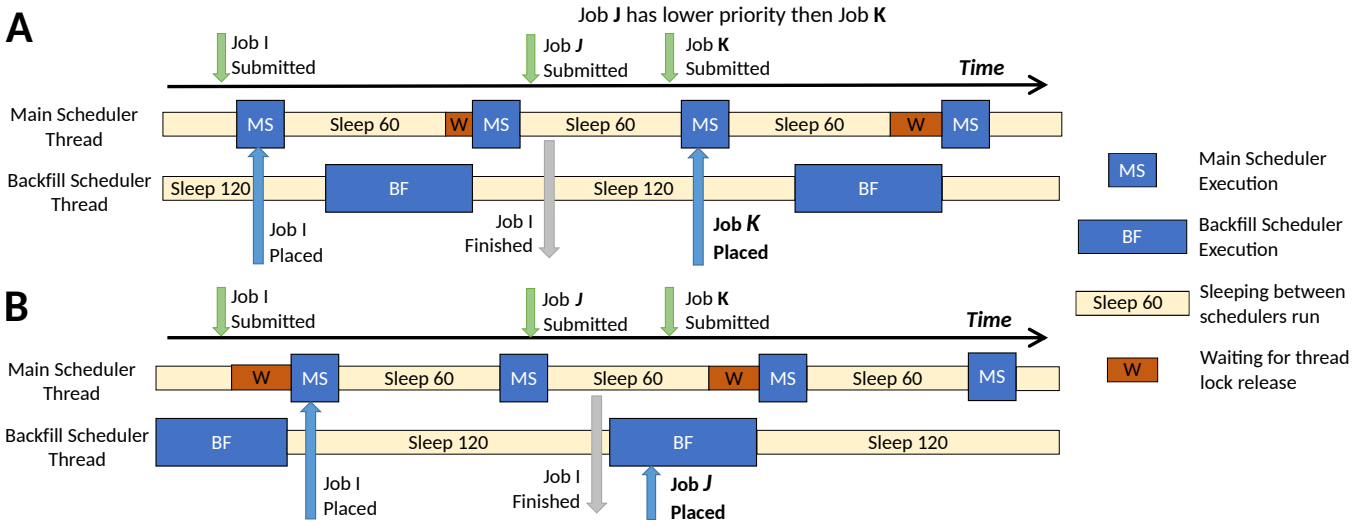

Figure 1: A timing diagram of job scheduling by the two Slurm scheduler components the main, priority based, scheduler (MS) and the backfill scheduler (BF). Although MS and BF schedulers are run by separate threads, due to the presence of the scheduling thread lock, the corresponding scheduling cycles run pseudo sequentially, that is, only one scheduler can perform job placement at a given time. After completing scheduling cycle both scheduler components go into a sleep cycle for a defined period of time. This causes the beginning of the scheduling cycle to be quasi-periodic in a sense that it has an irregular period which is equal to sleeping time plus scheduling time plus time waiting for the lock release. Here we show that such quasi-periodicity, system jitter and a difference in initial starting time can result in different job placements. In A on the top, Job K with a higher priority than Job J is scheduled first. However, in B on the bottom, even with the same submission timing and the same priorities Job J is scheduled before Job K because of where they happened to occur in the scheduling cycle.

is not really a critical difference because the Slurm schedulers are executed essentially serially (see the timing diagram in Figure 1).

In order to simulate a historical or synthetic workload a job trace file is created which contains all necessary information about the job. This includes: submission time, requested resources (nodes, memory, GPUs, etc.), requested and actual wall time. Because canceled jobs do interfere with scheduling we also include the capability to model them. For the case of a canceled job, the cancellation time is added. At the beginning of the simulation, the job trace file is read in and list of events are created. At the time of the event, the simulator performs the requested work, such as job submission or job cancellation.

As we have described the simulator so far, it can only perform simulations at a real-time rate, e.g. one day of simulated time would take one real day to run. This is not very useful. To accelerate the simulation, the Slurm simulator uses two approaches: time stepping and real-time scaling. In the case that there are no events on the system, the simulated time can be incremented by small time steps (usually 0.5 to 1.0 seconds). On a system with few job submissions, this could result in a significant acceleration. Unfortunately, on bigger systems, it would not help much because something is always happening, like job retirement, new job submission, and job placement. The backfill scheduler takes a significant amount of time to run. On a real Slurm installation, a single backfill scheduling cycle can often reach one minute or even more. Fortunately, the simulator can perform the same calculation about ten times faster due to several reasons: serial execution in simulator mode (to avoid thread locks), compilation with optimization flags and without assertion functions. Using these characteristics the backfill scheduler execution time can be scaled resulting in a much faster simulation. This is implemented in following way: during the backfill scheduling cycle, a simulated time is adjusted for each job scheduling attempt so that the simulated execution time of this attempt is equal to the real execution time multiplied by a scaling factor. The scaling factor is defined based on a comparison between the backfill scheduler performance in real and in simulated modes and is around ten. Such a real-time scaling results in a simulation about ten times faster, i.e. now 10 days of simulation time can be done in one day.

The result of using simulation time scaling with opportunistic time stepping is the dependency on a particular hardware where the simulation is run and on the background processes on the execution host. On one hand, it is an obvious drawback of the simulator, on the other hand, it is an essential feature of the Slurm scheduler itself. That is, faster hardware can result in better scheduling. Because of this feature, all calculations should be done on the same hardware otherwise the results are not directly compatible.

2.2 Simulations Design

To study the effect of multiple controller configurations, we performed a simulation with a single controller for the whole cluster and one where the KNL and SLX subclusters had their own dedicated controller. For each controller layout, we tested different configurations of node sharing on the SLX portion of the cluster. Specifically, no node sharing, sharing by sockets and sharing by

cores. Sharing by sockets means that jobs can share the same node but all cores from the same socket (physical CPU) can be allocated to only a single job at a time. Sharing by cores means that jobs can run on the same node but have their own dedicated cores.

Because the Slurm workload simulation is a stochastic process (Figure 1) the simulation results are not identical from run to run. Therefore to improve the confidence in the simulation and to study the variability of a large system on long workloads we performed multiple simulations for the various configurations, that is for a single controller with no node sharing and for separate controllers with node sharing enabled on SLX nodes (shared by cores). For each individual simulation, we randomize the time difference between the controller boot time and initiation of the submission of the job stream in order to determine the dependence of the job scheduling on this variable. Due to the significant amount of computational work required by these simulations, they were performed on an HPC cluster. Because the HPC cluster has different processors than the one used for the calculations described in the previous paragraph, this difference should be taken into consideration when these two sets are compared.

2.3 Slurm Configuration

The Slurm Configuration used was very similar to the one used on the actual TACC Stampede2 cluster. The following parameters were set differently. The time window for future job placement (`bf_window`) was set to 4.5 days with granularity set to 1 minute`bf_resolution`. The number of jobs from the same user considered for scheduling by the backfill scheduler, `bf_max_job_user`, was increased to 10 from 3. The sleeping interval between different backfill scheduling cycles (`bf_interval`) was set 30 seconds while the time limit for a single backfill scheduling cycle was set to 120 seconds (Currently Slurm has only a single parameter for these two values).

2.4 Workload Generation

At the time that this article was written, Stampede2 was still in the early production stage and thus there was little historical workload available. Furthermore, the available workload would not be representative of the workload during the full production stage. Therefore for the simulations, the workload was generated from the historical workload of the TACC Stampede1 supercomputer. Stampede1 has a similar total number of nodes as Stampede2 - 6400 but it has Sandy Bridge nodes with 16 CPUs per node.

The workload was generated as follows. First, a job bank was created from stampede 1 historical jobs running after 2015-05-16 and submitted before 2015-08-08 (a 12 week period). All single node jobs were converted to cores-requested jobs using average CPU utilization rounded to the closest biggest core count of 1,2,4,6,8 and 12. Jobs with CPU utilization higher than 12 was considered as requesting an entire node. The CPU utilization data was obtained from XDMoD [4] using TACC-Stats data [3]. Jobs were randomly selected from the job bank (without replacement). The number of selected jobs was proportional to the node counts of Stampede 1 and 2. A portion of jobs was set to be executed on KNL nodes (the fraction of jobs is proportional to the fraction of KNL nodes). The final workload consists of 295 thousand jobs which would require 11.3 million node hours. The distribution of the generated jobs over the node counts are shown in Table 1 and Figure 2. Figures 3.A and 3.B show the number of submitted jobs and requested node hours. Both of them exhibit weekly periodicity as original Stampede 1 workload.

In the SLX partition, 29% of all jobs or 2.4 % of the total node-hours are sub-node jobs requesting 12 or fewer cores. The assumption was that all jobs which can fully utilize an entire Stampede 1 node would also be able to utilize all cores on an SLX node. However, there are 48 cores on SLX nodes which is far more than the 16 cores of a stampede 1 node and therefore it is likely that in reality there are going to be more sub-nodal jobs than used in our test workload.

Table 1: Distribution of resulting workload over node count groups.

Node Count	Number of Jobs	Node Hours
Jobs for Execution on KNL Nodes		
Single-node jobs	91,004	380,159
Low Node Count, [2,64)	113,224	4,646,202
Mid Node Count,[64,512)	4,772	2,633,302
High Node Count, ⩾ 512	308	231,126
Jobs for Execution on SLX Nodes		
Subnode jobs	25,243	81,247
Single-node jobs	12,117	77,524
Low Node Count, [2,64)	46,995	1,935,412
Mid Node Count,[64,512)	2,044	1,224,495
High Node Count, ⩾ 512	114	86,850
Total	295,821	11,296,316

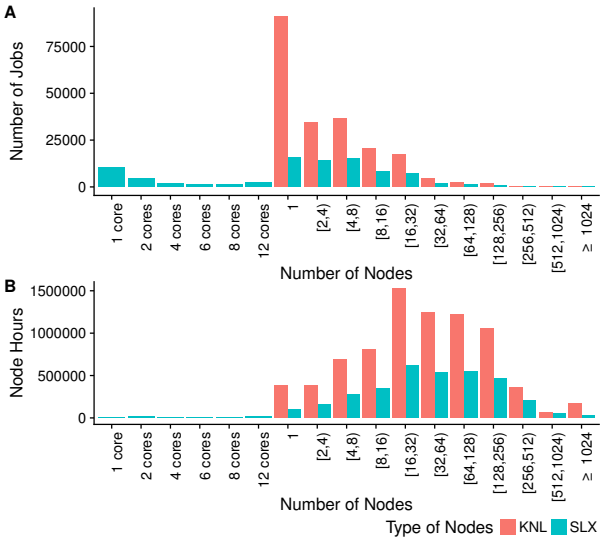

Figure 2: Distribution of jobs from workload over the requested nodes.

2.5 Execution Hardware

One of the simulator features is that its results depend on the performance of the hardware on which it is computed, therefore it is important to use the same hardware and software stack for the calculations. In this work, we used compute nodes from the UB-HPC academic cluster at Center for Computational Research, SUNY University at Buffalo which were dual socket Intel Xeon L5520 (2.27 GHz, totaling 8 cores) nodes with 24 GiB RAM and 2x160 GB RAID0 HDD storage. Because the simulation results are sensitive to the performance of used hardware only one simulation was done on a single node at a time.

2.6 Slurm Simulator Code Availability

Slurm simulator source code was forked from official Slurm repository and is available at https://github.com/nsimakov/slurm_simulator. To simplify further merges only the changes made to Slurm code itself are stored in that repository. The documentation, developed tools and examples are available at https://github.com/nsimakov/slurm_sim_tools.

3 RESULTS AND DISCUSSION

In this work a series of Slurm simulations was done to study two aspects of HPC resource management: utilization of multiple controllers and node sharing using the TACC Stampede 2 system as an example. Most modern HPC resources are heterogeneous, and if the sub-clusters are sufficiently large, dedicating a separate controller to each cluster can lead to better scheduling at the cost of maintaining multiple Slurm controllers. Node sharing allows placing serial and sub-node parallel jobs on the same node. This can increase the throughput and produce more efficient scheduling, but only if the number of additional resources to track (cores, memory, GPUs, etc.) would not overwhelm the scheduler. One of the biggest benefits of the Slurm simulator is that it uses same scheduling routings as actual Slurm and thus was used here to estimate actual Slurm performance.

The length of simulated workload is 12 weeks. Because simulation of this workload on Stampede 2 system containing 5936 nodes takes about 3-4 days in real time, first a single simulation was done for each configuration of interest, followed by series of simulations for the initial and final configuration of the system to study the stochastic variability. In the following, results are often grouped by node types, that is, into KNL and SLX jobs. Totally 19 separate Slurm simulation was performed.

3.1 Job Waiting Time

From the user's point of view job waiting time is one of the most important characteristics of the scheduler and HPC resource. Shorter wait time would benefit the end users. To monitor change in this important characteristic we used three different metrics for job wait time. The first one is arithmetic mean of all job waiting time. The second one is the mean of the average user waiting time, that is first the average wait time for each user was calculated and then these values were averaged. These two means characterize the users befits, and show the average change in their waiting time under different Slurm configurations. The third one is the node-hour weighted mean of the job waiting time. This metric shows the average time needed to wait in the queue in order to produce one node-hour of computation. This essentially characterizes the efficiency of the job packing in the system.

Table 2: Mean wait time in different multi-controller and node sharing. The results are grouped by node types. The averaging was done over jobs longer than half an hour.

Controller	Node Sharing	Mean Wait Hours	Mean User Average Wait Hours	Wait Hours, Weighted Mean by Node Hours
Jobs Executed on KNL Nodes				
Single	No Sharing	9.8 (0%)	9.9 (0%)	9.3 (0%)
	By Sockets	9.2 (-5%)	9.3 (-6%)	9.4 (0%)
	By Cores	9.2 (-5%)	9.3 (-6%)	9.3 (0%)
Separate	No Sharing	9.7 (-0%)	9.6 (-3%)	9.4 (+0%)
Jobs Executed on SLX Nodes				
Single	No Sharing	14.7 (+0%)	13.6 (+0%)	17.6 (0%)
	By Sockets	12.3 (-16%)	11.5 (-16%)	16.2 (-8%)
	By Cores	12.8 (-13%)	11.9 (-12%)	16.7 (-5%)
Separate	No Sharing	9.5 (-35%)	9.1 (-33%)	14.7 (-16%)
	By Sockets	8.8 (-40%)	8.5 (-38%)	14.2 (-19%)
	By Cores	9.1 (-38%)	8.7 (-36%)	14.7 (-16%)

Table 3: Mean of mean wait time is averaged over 5 independent simulations with same Slurm configuration. The number after plus-minus sign reports a population standard deviation of mean wait time

Controller	Node Sharing	Mean of Mean Wait Hours	Mean of Mean User Average Wait Hours	Mean of Wait Hours, Weighted Mean by Node Hours
Jobs Executed on KNL Nodes				
Single	No Sharing	9.76 ± 0.01	9.97 ± 0.02	9.30 ± 0.04
Separate	No Sharing	9.78 ± 0.05	9.73 ± 0.07	9.37 ± 0.04
Jobs Executed on SLX Nodes				
Single	No Sharing	14.87 ± 0.27	13.79 ± 0.27	17.77 ± 0.27
Separate	By Cores	8.89 ± 0.16	8.56 ± 0.10	14.32 ± 0.26

The mean wait times for single-run simulations are shown in Table 2. For the case of KNL nodes, the mean wait time was nearly the same in all cases. This is not very surprising since the node sharing only affects the SLX nodes and there is only a moderate decrease in the number of KNK nodes handled when two separate controllers are used. For the SLX nodes, the mean waiting time decreased by 35% by switching to a separate controller and the mean node-hours weighted wait time decreased by 16%. Turning on node sharing has a smaller effect; it reduces the mean wait time by 13-16% and the node-hours weighted mean by 5-8% for a single controller. The effect of adding node sharing when separate controllers are used is even smaller. The single simulation run of

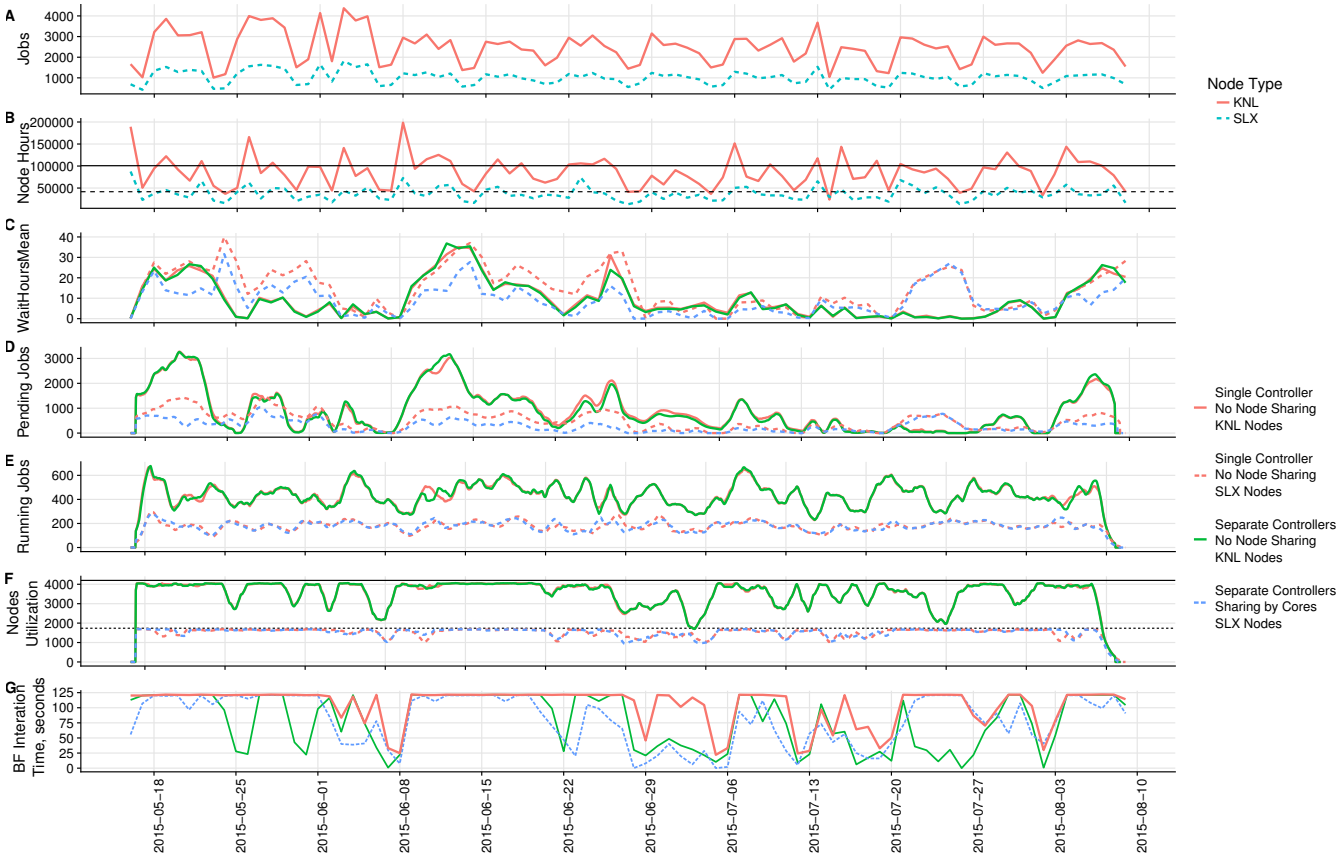

Figure 3: The distribution of various properties along the time axis. The major ticks along the time axis are spaced by one week and each tick corresponds to a Monday. A - Number of jobs submitted per day, note the weekly pattern. B - Total number of node hours of jobs submitted each day, black horizontal line show maximum node hours deliverable by the system. C - Daily mean wait time. D - Number of pending jobs along simulated time, node types are shown by line style: solid lines for KNL nodes and dashed line for SLX, colors show controller and node sharing configuration. E -Number of running jobs along simulated time. F - Nodes utilization along simulated time. G - daily averaged backfill scheduler iteration time.

Table 4: Distribution of mean wait time over node counts.

Cont-roller	Node Sharing	Subnode Jobs	Single-node Jobs	Low Node Count, [2,64)	Mid Node Count, [64,512)	High Node Count, ⩾ 512
Jobs Executed on KNL Nodes						
	No Sharing		9.4 (+0%)	10.1 (+0%)	9.1 (+0%)	7.6 (+0%)
Single	By Sockets		8.8 (-7%)	9.6 (-5%)	9.0 (-1%)	8.8 (+16%)
	By Cores		8.8 (-7%)	9.6 (-5%)	8.9 (-2%)	8.8 (+16%)
Separate	No Sharing		9.3 (-2%)	10.1 (+1%)	8.8 (-3%)	4.1 (-46%)
Jobs Executed on SLX Nodes						
	No Sharing	13.6 (+0%)	14.9 (+0%)	15.0 (+0%)	16.7 (+0%)	19.3 (+0%)
Single	By Sockets	11.3 (-17%)	12.5 (-16%)	12.6 (-16%)	15.2 (-9%)	23.2 (+20%)
	By Cores	11.6 (-15%)	13.0 (-13%)	13.1 (-12%)	15.6 (-7%)	21.3 (+10%)
	No Sharing	8.3 (-39%)	9.3 (-38%)	10.0 (-33%)	12.9 (-23%)	17.1 (-11%)
Separate	By Sockets	7.7 (-44%)	8.6 (-42%)	9.2 (-39%)	12.3 (-26%)	20.9 (+8%)
	By Cores	7.8 (-43%)	9.0 (-40%)	9.5 (-37%)	12.8 (-24%)	19.9 (+3%)

Table 5: Distribution of jobs between schedulers. Only jobs longer than half an hour were used.

Controller	Node Sharing	Main Sceduler, %	Backfill Scheduler, %
Jobs Executed on KNL Nodes			
Single	No Sharing	82	18
	By Sockets	73	27
	By Cores	73	27
Separate	No Sharing	66	34
Jobs Executed on SLX Nodes			
Single	No Sharing	56	44
	By Sockets	44	56
	By Cores	42	58
Separate	No Sharing	49	51
	By Sockets	42	58
	By Cores	42	58

Table 6: Backfill Scheduler Characteristics

Controller	Node Sharing	Mean Execution Time, seconds	Wall Time Limit Hit Rate, %	Mean Jobs Attempted to Schedule
Single	No Sharing	101	77	134
	By Sockets	88	64	176
	By Cores	89	65	175
Separate, KNL	No Sharing	74	54	148
Separate, SLX	No Sharing	78	51	131
	By Sockets	81	50	126
	By Cores	80	47	128

node sharing by socket is slightly better than of sharing by cores. However, the value of sharing by socket is within one standard deviation of the core sharing value averaged over 5 different runs (Table 3). Switching to separate controllers improves waiting time across all job sizes with a greater improvement at lower node counts (Table 4). It is similar to node sharing, with an exception for jobs with high node count.

Mean daily waiting time alone time is shown in Figure 3.C for a subset of configuration. The wait time on KNL nodes is mostly the same while on SLX it is smaller for separate controllers and node sharing.

Because of the scheduler stochasticity, we perform five separate calculations for the initial and final configurations of the system. Namely, the initial configuration was single Slurm controller without node sharing and the final configuration was separate Slurm controllers for KNL and SLX nodes with node sharing enabled on SLX nodes. For the node sharing a single core was a smallest allocatable unit. The results of these simulations are shown in Table 3. The standard deviation of mean wait time is small: it is less than 1% of mean value for KNL nodes and around 2% for SLX nodes. Using results of these multiple runs, the overall effect of switching to multiple controllers and enabling node sharing can be 40% reduction in mean wait time and 19% reduction in node-hours weighted wait time mean.

3.2 Schedulers

The previous section shows that separate controllers and node sharing improves waiting time on the system. The origin of this performance improvement is based upon how Slurm functions. Most Slurm installations utilize two schedulers: a main priority based scheduler, which schedules only high priority jobs, and a backfill scheduler, which attempts to place jobs in vacant intervals between high priority jobs. The latter is more complicated and significantly more computationally demanding. The backfill scheduler computation on large systems is often a performance bottleneck. In this section, we examine metrics which characterize the Slurm scheduler's performance.

Table 5 shows the percentage of jobs which were placed by each scheduler type. Although the workload for KNL nodes and SLX nodes is very similar and mainly differ by the number of jobs, on the KNL partition most jobs were placed by the main scheduler while on the SLX partition each scheduler placed roughly half of jobs. One factor which helps to explain this difference is that while the mix of job sizes is similar for the two clusters the SLX cluster is substantially smaller. Hence, on the SLX partition, the ratio of max job size node count to the total number of SLX nodes is 0.6 making such a job a capability type job. For the KNL partition, this ratio is only 0.2. Therefore, all of the jobs on the KNL nodes sub-cluster are capacity jobs and a good portion of the SLX jobs are capability jobs. As indicated by the simulation results and explained in further detail below, it is more difficult and more computationally stressing to the schedule such capability jobs. The SLX partition, therefore, benefits more from increasing the computational capacity by implementing separate controllers for each partition.

The configurations with shorter wait times have a higher portion of jobs scheduled by backfill scheduler (Tables 2 and 5). This makes sense because the main agenda of the backfill scheduler is to place lower priority jobs without affecting the launch time of higher priority jobs. This manner a more efficient backfill scheduling results in more jobs started earlier and thus smaller wait times.

In Slurm, the backfill scheduler has a time-limit on its execution. During each scheduling cycle, the backfill scheduler goes through a priority ordered job list and attempts to schedule each job. If it reaches a time-limit, it stops execution and after a sleeping period starts a new cycle from the beginning. Getting through the whole list is important to execute efficiently the "fill the gaps" type scheduling which the backfill scheduler does. Since typically the backfill scheduler is not able to consider all of the potential jobs on every cycle, see Figure 3, any scheduler modification which allows increased consideration of queued jobs should be beneficial.

Table 6 shows the mean run time of the backfill scheduler, percent of cycles hitting the run time limit and a mean number of jobs that the backfill scheduler attempted to schedule. The single controller without sharing hits the run time limit most often 77% of all runs, while using separate controllers with node sharing hits it on only between 47% and 54% of the runs. Figure 3.G) shows the

daily average backfill cycle execution time along the time simulation axis for a subset of configurations. As can be seen, although separate controllers reach the time-limit less often it still is not an uncommon occurrence. However, implementing separate controllers reduces the number of managed resources and jobs per controller thus resulting in better performance of the backfill scheduler. The combined mean attempts to schedule a job is about two times larger than for a single controller.

A more difficult result to explain is that, even for a single controller, enabling node sharing leads to a lower time limit hit rate and a higher number of jobs attempted to be scheduled (Table 6). Most likely this is because enabling node sharing results in more jobs running on the system which in turn leads to a reduction of jobs in the queue. Another possible contribution is that, because of the smaller number of available spots in the system, job rejection can occur more quickly. In any case, it is important that with node sharing enabled backfill scheduler is not overwhelmed by extra allocatable resources and work more efficiently.

Different workload and scheduling properties over the time are shown in Figure 3 for the initial (single controller, no node sharing) and final (separate controllers, node sharing by cores on SLX nodes) configurations. The results of separate controllers and node sharing are minimal on KNL nodes. But it is a different situation for SLX nodes. The simulations show significant reduction in wait time (Figure 3.C) and number of jobs in queue (Figure 3.D). Although node utilization (Figure 3.F) and the number of jobs running ((Figure 3.E)) look very similar for the various configurations, the subtle differences translate into significant differences in the waiting times.

4 CONCLUSIONS

A Slurm simulator was used to study the applicability of implementing multiple Slurm controllers and node sharing on large systems and TACC Stampede 2 in particular. The objectives were to estimate the potential benefits of multiple Slurm controllers and node sharing, to check the performance of the backfill scheduler in node sharing mode and to evaluate the combined benefits of multiple Slurm controllers and node sharing.

It was found that separate Slurm controllers lead to 35% smaller wait times on SLX jobs and nearly the same performance of KNL nodes.

Node sharing did improve the scheduling performance even when only a single controller was used. It should be noted that node sharing was on only on the SLX nodes, which constitute only 1,736 nodes out of total 5,936 nodes. If the node sharing would be on all 5,936 nodes, the results might be totally different.

The combination of node sharing and multiple controllers offer the best performance, namely waiting time is reduced by 40%. In this configuration, for the Stampede job mix, the backfill schedulers have a greatly reduced rate of hitting the time limit and double the number of attempts to do job placement. Therefore in the case of a natural separation into sub-clusters, multiple Slurm controllers and node-sharing utilization can offer substantially smaller wait-times and should be considered by the HPC resource operator.

Slurm offers high configurability that can meet the needs of a large range of HPC resources and the Slurm simulator which we developed can help in the search for optimal parameters to ensure high HPC resource utilization.

ACKNOWLEDGMENTS

This work is supported by the National Science Foundation under grant number ACI 1445806 (XD Net Metrics Service for High Performance Computing).

REFERENCES

[1] Susanne M. Balle and Daniel J. Palermo. 2008. Enhancing an Open Source Resource Manager with Multi-core/Multi-threaded Support. In *Job Scheduling Strategies for Parallel Processing*, Eitan Frachtenberg and Uwe Schwiegelshohn (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 37–50. https://doi.org/10.1007/978-3-540-78699-3_3

[2] Alex D. Breslow, Leo Porter, Ananta Tiwari, Michael Laurenzano, Laura Carrington, Dean M. Tullsen, and Allan E. Snavely. 2013. The case for colocation of high performance computing workloads. *Concurrency and Computation: Practice and Experience* 28, 2 (2013), 232–251. https://doi.org/10.1002/cpe.3187 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.3187

[3] T. Evans, W. L. Barth, J. C. Browne, R. L. DeLeon, T. R. Furlani, S. M. Gallo, M. D. Jones, and A. K. Patra. 2014. Comprehensive Resource Use Monitoring for HPC Systems with TACC Stats. In *2014 First International Workshop on HPC User Support Tools*. 13–21. https://doi.org/10.1109/HUST.2014.7

[4] J. T. Palmer, S. M. Gallo, T. R. Furlani, M. D. Jones, R. L. DeLeon, J. P. White, N. Simakov, A. K. Patra, J. Sperhac, T. Yearke, R. Rathsam, M. Innus, C. D. Cornelius, J. C. Browne, W. L. Barth, and R. T. Evans. 2015. Open XDMoD: A Tool for the Comprehensive Management of High-Performance Computing Resources. *Computing in Science Engineering* 17, 4 (July 2015), 52–62. https://doi.org/10.1109/MCSE.2015.68

[5] Nikolay A. Simakov, Robert L. DeLeon, Joseph P. White, Thomas R. Furlani, Martins Innus, Steven M. Gallo, Matthew D. Jones, Abani Patra, Benjamin D. Plessinger, Jeanette Sperhac, Thomas Yearke, Ryan Rathsam, and Jeffrey T. Palmer. 2016. A Quantitative Analysis of Node Sharing on HPC Clusters Using XDMoD Application Kernels. In *Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale (XSEDE16)*. ACM, New York, NY, USA, Article 32, 8 pages. https://doi.org/10.1145/2949550.2949553

[6] Nikolay A. Simakov, Martins D. Innus, Matthew D. Jones, Robert L. DeLeon, Joseph P. White, Steven M. Gallo, Abani K. Patra, and Thomas R. Furlani. 2018. A Slurm Simulator: Implementation and Parametric Analysis. In *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*, Stephen Jarvis, Steven Wright, and Simon Hammond (Eds.). Springer International Publishing, Cham, 197–217. https://doi.org/10.1007/978-3-319-72971-8_10

[7] Slurm Developers Team. 2018. Slurm: Large Cluster Administration Guide. Retrieved March 8, 2018 from https://slurm.schedmd.com/big_sys.html

[8] Slurm Developers Team. 2018. Slurm Workload Manager. Retrieved March 8, 2018 from https://slurm.schedmd.com/

[9] Joseph P. White, Robert L. DeLeon, Thomas R. Furlani, Steven M. Gallo, Matthew D. Jones, Amin Ghadersohi, Cynthia D. Cornelius, Abani K. Patra, James C. Browne, William L. Barth, and John Hammond. 2014. An Analysis of Node Sharing on HPC Clusters Using XDMoD/TACC_Stats. In *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment (XSEDE '14)*. ACM, New York, NY, USA, Article 31, 8 pages. https://doi.org/10.1145/2616498.2616533

[10] Andy B. Yoo, Morris A. Jette, and Mark Grondona. 2003. SLURM: Simple Linux Utility for Resource Management. In *Job Scheduling Strategies for Parallel Processing*, Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 44–60. https://doi.org/10.1007/10968987_3