Developing Accurate Slurm Simulator

Nikolay A Simakov
Robert L. DeLeon
Yuqing Lin
Phillip S. Hoffmann
William R. Mathias
nikolays@buffalo.edu
rldeleon@buffalo.edu
Center for Computational Research, University at Buffalo
Buffalo, NY, USA

ABSTRACT

A new Slurm simulator compatible with the latest Slurm version has been produced. It was constructed by systematically transforming the Slurm code step by step to maintain the proper scheduler output realization while speeding up simulation time. To test this simulator, a container-based Virtual Cluster was generated which fully mimicked a production HPC cluster. As for all Slurm simulators, the realization is a stochastic process dependent on the computational hardware. Under favorable conditions the simulator is able to approximate the actual Slurm scheduling realization. The simulation fidelity is sufficient to use the simulator for its main function, that is, to test Slurm parameter configurations without having to experiment on full production systems.

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, Slurm

ACM Reference Format:
Nikolay A Simakov, Robert L. DeLeon, Yuqing Lin, Phillip S. Hoffmann, and William R. Mathias. 2022. Developing Accurate Slurm Simulator. In *Practice and Experience in Advanced Research Computing (PEARC '22), July 10–14, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3491418.3535178

1 INTRODUCTION

Slurm is an open-source job scheduling system that is widely used in many small and large-scale high-performance computing (HPC) resources. It is highly tunable, with many parametric settings that can significantly influence job throughput, overall system utilization and job wait times. In many cases it is difficult to judge how modification of these parameters will affect the overall performance of the HPC resource. In 2017 we released and utilized our first Slurm Simulator intended to aid HPC personnel tuning Slurm's parameters to optimize throughput or meet specific workload objectives without impacting the HPC system in production [5, 6]. An important discovery was that scheduling is a stochastic process influenced by the execution time uncertainty in the resource allocation and release procedures relative to job submission. The uncertainty originates from significant variations in the time between execution of multiple resource handling routines including the priority-based scheduler and the backfill scheduler and differential detection of jobs exciding the time-limit. The stochasticity results in different scheduling realizations and consequently the need to run multiple simulations to infer the effects of changing the Slurm parameters.

The original version of our simulator was used by us [5, 6] and others to study various aspect of job scheduling on HPC resources. Eleliemy and Ciorba extended the simulator to study multilevel scheduling [1]. Tanash et al [7] and Wang at al [9] used our Simulator to evaluate their ML model for wait time prediction. Villapando and Rubio supplemented their study on how walltime limit correction affect the wait time with simulated results [8]. There have been several Slurm simulators developed over the last decade. RM-Replay [4] uses a slightly modified Slurm and achieves faster than a real time modeling by adjusting the real clock time. Such an approach offers the benefits of an unmodified Slurm but effectively lowers the performance of the control node by scaling the clock speed. This results in a modestly reliable acceleration (x10) and the need for powerful multicore hardware to accommodate the backlogger retiring and starting jobs. It is also not clear the effect on the individual jobs as the clock scaling effects were studied only on workload completion time and not individual jobs. The BSC simulator [2] claims a high time acceleration and high accuracy. However, it was tested only on a homogeneous cluster with a relatively light workload modelled after late 1990s supercomputers which typically would not use the job backfiller and thus would have only moderate utilization. Thus, it is unclear how the simulator would perform for a more realistic, modern workload with 90%+ resource utilization. The earlier version of the BSC Slurm simulator failed to model such workloads and it is unclear that the issues were addressed in the current version of the BSC simulator.

PEARC '22, July 10–14, 2022, Boston, MA, USA

ACM ISBN 978-1-4503-9161-0/22/07…$XX.XX
https://doi.org/10.1145/3491418.3535178

Since 2018, several major Slurm releases occurred, and our previous simulator is now obsolete. Here we report on the progress of a new Slurm simulator based on a recent Slurm release. In this development, major attention was given to the reproduction of real Slurm runs of the same workload. For obvious reasons we can not obtain multiple historical runs of the same workload in a real system and thus some kind of automated workload feeding to Slurm is required. We address this problem by implementing a Virtual Cluster using docker containers where each separate container represents a separate compute or head node. The virtuality here refers to containers being virtual hosts, Slurm is installed on it in a regular manner just like on actual production system. The Virtual Cluster allows us to obtain multiple scheduling realizations as a utility for our Slurm simulator development. Unfortunately, our first attempt to apply the modification used in the earlier simulator version directly to the new version of Slurm failed miserably in the sense that the simulation of the reference workload was very poor. Lengthy debugging didn't localize the problem. We tackled this problem by starting with a new Slurm code and introducing small changes one at a time to implement our simulator. After each change we checked to make sure the simulator was able to reproduce the actual Slurm results. The resulting simulator, under certain conditions, produces a workload statistically indistinguishable from the reference workload.

2 METHODS

Virtual Cluster. The Slurm Simulator can only be useful if it can reproduce the behavior of Slurm on actual HPC resources. From real-life systems, we can only retrieve a single historic workload realization (jobs' submit, start and end time alone with requested and allocated resources). However, in the majority of cases of interest, the job placement occurs in a stochastic manner. Thus, for a valid statistical comparison, we need a multiple runs with the same workload. To do that on a real full-scale system is complicated and impractical. Thus, we developed a Docker-based Slurm virtual cluster with an automatic batch job feeder. The created cluster uses a normal Slurm environment, and essentially is only different from the real-life installation in HPC centers since it is deployed in Docker containers rather than on actual hardware. Besides using it to obtain critical metrics for the Slurm simulator, the developed virtual cluster can be used for other purposes related to Slurm HPC resource management, such as training system administrators and testing Slurm settings in real-time. It was also used to measure the effect of the Slurm built-in job profiler on a filesystem.

A Slurm virtual cluster was created using docker containers where each node of the HPC resource is represented by a container (one for the head node and one for each compute node). A bridge network connection allows standard network communication between nodes. We created an infrastructure for the automatic building of head and compute node docker images with all required services and configurations, including users' accounts. Docker-composer is used to launch the whole cluster. Upon the launch, the cluster represents a complete functioning HPC resource. It is possible to login to the head node as a regular or administrative user and launch batch or interactive jobs, manage Slurm accounts and other operations regularly executed on HPC resources.

To automate job submission, we created an automated job feeder that uses a text-based human-readable format for job request specification. The format extends Slurm's request specification with new options for submission time and actual job duration. This will decrease the learning time as it uses known arguments from Slurm utilities. The format is extensible and later we can include other events like job cancellation and taking down a node. Same format was adopted for the simulator.

To account for the scheduling uncertainties systematically, we add a configurable delay between the start of the Slurm controller and submission of the job stream. The delay is set randomly before the simulation. The state of repeatedly executed resource managing routines are unknown a priori and can be different in different realizations. The randomized initial delay simulates it.

For the batch jobs, we created a set of mini-applications that can be used for different scenarios. The one which will be used the most is the sleep job: the application will simply sleep for a specified time and exit with the requested status. Due to the sleep app's low compute requirement, it is possible to squeeze more virtual compute nodes on the same physical host and is most useful to study the scheduling aspects of Slurm. Another application, called miniapp, cycles over three stages: matrix-matrix multiplication, memory growth and sleep. This app is more helpful when we want a compute job to do something more substantial than simply sleeping. It was used for measuring the effect of the Slurm build-in job profiler. We recycle it on Ookami, an ARM-SVE cluster, to ensure that the profiler works as expected on this novel machine.

For production runs, a virtual cluster was deployed in our center OpenStack cloud at CCR. We have found that 200 compute nodes can be simulated by a single 8 core virtual machine; we reserve the first 4 cores for the head-node and the other 4 cores for all other nodes.

Slurm Simulator. In our previous simulator version, we opted for complete serialization and rather intensive Slurm code modification for better performance and control. Such an approach turns out to be difficult to maintain and makes it impossible for automatic version update for new versions of Slurm. Thus instead of complete serialization, we decided to go a different route and to minimize core Slurm code changes to have a clean separation between a simulator and the actual Slurm code. This should enable better migration to a newer version and should facilitate adoption by Slurm developers. The new version of the simulator is based on version 21.08 of Slurm.

To minimize Slurm core code modifications, we used several approaches to alter the Slurm execution: 1) use the GCC compiler option to wrap standard library functions, 2) use the constructor method to initialize the simulator and thus avoid modifications to the main functions and 3) to access static variables and functions we created a wrapper for the source code with additional functionality to access variables and functions of interest. The only major part of the core Slurm code modified is thread creation, which is very conservative and unlikely to change.

A key feature of the simulator is time acceleration done by opportunistic time skipping in case of no evens. It is implemented by wrapping (see GCC compiler option `--wrap`) all time related function (`gettimeofday` and various `sleep` functions) as well as some `pthread` functions (`pthread_cond_timedwait` and some others).

Possible time skipping is identified by calculating the time to the next event, if the time is within a threshold (more than 10 ms) the clock is incremented by that time. Because the main slurm background loop sleeps for a maximum of one second, the time skipping never exceeds one second. Scheduling events include: repeated priority and backfill scheduling, the main slurm background loop, job submission and completion.

As for usability improvement, we implemented a new text-based specification for batch jobs, using the same format as for the virtual cluster. In the previous version, the jobs specification is set using a raw-C data structure following remapping to the Slurm internal format. Now, we use the same text format as used for request specification by users and invoke functions from the Slurm sbatch utility to convert it to Slurm internal format. This way we do not handle the conversion manually and specify the jobs similar to the way they are input by Slurm users.

Reference Test System and Workload. Two systems were used as a reference test system: a Micro cluster, a small 10-node resource and UB-HPC, and a 217-node cluster, corresponding to a newer portion of the academic cluster at our center. Micro cluster is used as a reference test system. It was used previously, but here it is implemented using a Docker-based virtual cluster instead of a Slurm frontend mode, making it more similar to a real Slurm installation. This ten node cluster consists of different node types: two CPU generations, a large memory and a GPU nodes. This selection allows us to validate job placement based on the resource requests within the batch job. The 500 jobs workload was generated requesting a mixture of non-specific nodes and specific nodes. Jobs were distributed between five users grouped into two accounts. Some jobs were allowed to run indefinitely to trigger wall limit exceed routine. This setup was executed over 20 times with an average running time of 13.7 hours. To improve turn-around, a shrunken version of the previous was added. It takes 48 minutes to run. Finally, for very fast checks a small 10 jobs workload were also develop. UB-HPC cluster (217 nodes) was used as a more realistically sized system. It corresponds to a newer portion of the our center cluster. It consists of 32 cores per node nodes (87 regular and 2 GPU nodes) and 40 cores per node nodes (96 regular, 8 GPU and 24 large memory nodes). The used historic workload contains 29,678 jobs from 95 user among 65 groups. The workload was run on Virtual Cluster 8 times and took 29.4 days per run.

Comparison of Different Scheduling Realizations. During our incremental simulator development route, we needed an effective way to determine that new changes to the code still produced the same expected results. As with many other scientific programs, the difficulties arise from the fact that scheduling is a stochastic process (as opposed to a deterministic process) and thus a simple comparison with a minimalistic threshold to account for minor rounding error would not suffice. Because our reference contains multiple realizations of same workload, we can make a statistically valid comparison. The major outcome of the run is waiting time and that is what we essentially want to compare, however it will vary with the configuration. The job's waiting times are dependent variables within a single run due to shared resources (compute nodes) they compete for. Hence, comparison of averaged wait times can be misleading, because if some jobs start running earlier then some jobs will start later and this would have effect on the mean wait time. So, we decided to treat each job's wait time as a dependent random variable within each run and each run is independent of each other. This way we can compare runs instead of individual jobs and use multivariate analysis. To compare the distance between runs, the Euclidian distance was used where each vector component corresponds to the individual job's wait time. A heat map of distances between individual normal and simulated runs is used for visual comparison. If both reference and simulated runs are drawn from the same distribution the distance between the same type of runs should be same as between different types of runs. A statistical method which essentially capitalizes on this the is the multivariate Wilcoxon test [3], which was used for a more formal judgment of similarities between reference and simulated results.

Compute Resources and Code Repositories. The Micro cluster reference workload realization was calculated using a desktop with Intel i7-9800X, simulation was performed on the same desktop or a laptop with Intel i7-11800H 2.3GHz. The UB-HPC cluster reference workload realization was calculated using our SUNY University at Buffalo center's OpenStack Cloud resource, the simulation was performed on the same desktop or on the Purdue University Anvil Cluster (AMD EPYC 7763 2.54GHz CPU). The Slurm simulator code, tools, virtual cluster, cluster configurations, reference and simulated workloads are available at *https://github.com/ubccr-slurm-simulator/*.

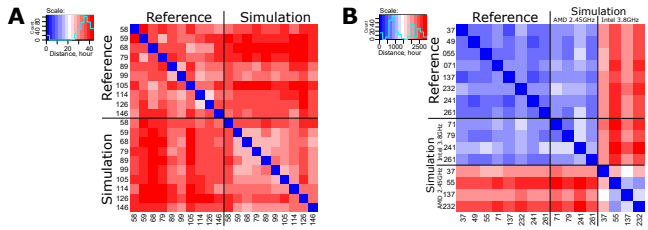

Figure 1: Distance heatmap comparing reference and simulated workload realizations as measured by Euclidian distance on the job's wait time. Each label on the x- and y- axes corresponds to an individual run and each number corresponds to the delay in seconds between the Slurm controller start and the first job submission. As the scale legend indicates, the blue color is the closest, white further and red the greatest distance apart. A Micro cluster, 500 job workload. B. UB-HPC cluster.

3 RESULTS AND DISCUSSIONS

Our initial attempt to apply changes we made earlier to an old version of Slurm to the new version of Slurm produced a working code, however, the resulting job schedule was very different from the reference. A lengthy attempt to debug didn't lead to a solution, there were too many significant changes and so we changed the approach and started from the beginning. We started from the new version of Slurm and incrementally applied changes to make it a Slurm Simulator. At each step we ran a validation test to ensure that we did not introduce any bugs that affected the simulator performance. In the early steps the code was close to the actual

Slurm and we used our Virtual Cluster setup to validate updates. Along with modifications of Slurm, the Virtual Cluster was also evolving to a single Slurm Simulator running docker container. The opportunistic time stepping was introduced last and thus running a test on our full-scale workload would be prohibitively long and thus we largely used small 10 jobs workload and shrunk the 500 jobs workload for validation. Only after time skipping was enabled were we able to switch to the full-scale 500 jobs workload for validation. The UBHPC workload was used for the final validation. Such a scheme proved to be more efficient as it allowed us to catch problems earlier.

The distance maps between reference and simulated workload realizations are shown in Figure 1. The distance is calculated as the Euclidian distance on job wait times. In case of the Micro cluster the reference and simulated look very similar and the multivariate Wilcoxon test indicates that they are not statistically different. For the UB-HPC cluster (Figure 1.B), the result is hardware dependent, the faster computer provides results that are closer to reference. The average within reference data distance is 742±135 hours, while the distance of simulated data to reference data are 906±180 hours and 2290±262 for execution on fast and slower hardware respectively. If all realization are pulled from same distribution these three ranges should be the same, clearly the simulation executed on slower hardware stands out. However, if we recalculate the average distance per job it is only about 3 minutes, which might be close enough in practice. For a better understanding of the hardware dependance and stochasticity we run a number of experimentations on the Micro system with shrunken workloads. The shrunken workload appears to be further apart from reference than a full-scale workload (though not far enough to be statistically significant). If time stepping were off, or with significant padding around events, the resulting job scheduling is much closer to reference. We speculate that the larger deviation from reference on slower hardware is due to the longer times needed for thread synchronization. We might or might not be able to mitigate that in the future. In the latter case we need to study the predictability power of the simulator. In many cases the prediction of relative changes is more reliable due to offsetting differences in the simulation.

The time acceleration was somewhere between 20 and 40 for the Micro and the UB-HPC cluster. While this is not as impressive as earlier versions of the simulator, it is still practicable since it allows us to generate a month-long workload in about a one computational day. Unlike in the Virtual Cluster, in simulation mode compute nodes are not modeled explicitly, but their responses are mimicked within the controller itself without starting additional threads. This significantly decreases the compute demand and dependency on cluster size.

4 CONCLUSIONS AND FUTURE PLANS

Using high performing CPUs running the newly developed version of the Slurm simulator produced a schedule statistically indistinguishable from the reference realization. Modest time acceleration allows us to simulate a month long workload in one-two days. The simulation shows a dependency on the hardware performance and further study is needed to analyze the accuracy of the simulator. Like many other simulators of real physical phenomena, it is very likely that even though the simulator may not produce perfectly accurate absolute results it can be much more accurate in predicting relative changes.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation under awards OAC 2004954. This work used compute resources at UB CCR and the XSEDE through allocation CCR120014.

REFERENCES

[1] Ahmed Eleliemy and Florina M Ciorba. 2021. A Resourceful Coordination Approach for Multilevel Scheduling. *arXiv preprint arXiv:2103.05809* (2021).

[2] Ana Jokanovic, Marco D'Amico, and Julita Corbalan. 2018. Evaluating SLURM simulator with real-machine SLURM and vice versa. In *2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. IEEE, 72–82.

[3] Jana Jurečková and Jan Kalina. 2012. Nonparametric multivariate rank tests and their unbiasedness. *Bernoulli* 18, 1 (2012), 229–251.

[4] Maxime Martinasso, Miguel Gila, Mauro Bianco, Sadaf R. Alam, Colin McMurtrie, and Thomas C. Schulthess. 2018. RM-Replay: A High-Fidelity Tuning, Optimization and Exploration Tool for Resource Management. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. 320–332. https://doi.org/10.1109/SC.2018.00028

[5] 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 *Proceedings of the Practice and Experience on Advanced Research Computing* (Pittsburgh, PA, USA) *(PEARC '18)*. Association for Computing Machinery, New York, NY, USA, Article 25, 8 pages. https://doi.org/10.1145/3219104.3219111

[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. 2017. A slurm simulator: Implementation and parametric analysis. In *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*. Springer International Publishing, Cham, 197–217. https://doi.org/10.1007/978-3-319-72971-8_10

[7] Mohammed Tanash, Daniel Andresen, and William Hsu. 2021. AMPRO-HPCC: A Machine-Learning Tool for Predicting Resources on Slurm HPC Clusters. In *The Fifteenth International Conference on Advanced Engineering Computing and Applications in Sciences ADVCOMP*. 20–27.

[8] Aira Villapando and Jessi Christa Rubio. 2021. Simulation vs Actual Walltime Correction in a Real Production Resource-Constrained HPC. In *Practice and Experience in Advanced Research Computing*. 1–7.

[9] Hao Wang, Yi-Qin Dai, Jie Yu, and Yong Dong. 2021. Predicting running time of aerodynamic jobs in HPC system by combining supervised and unsupervised learning method. *Advances in Aerodynamics* 3, 1 (2021), 1–18.