A Fast Simulator to Enable HPC Scheduling Strategy Comparisons

Alex Wilkinson[1,2], Jess Jones[1], Harvey Richardson[1], Tim Dykes[1], and Utz-Uwe Haus[1]

[1] HPE HPC/AI EMEA Research Lab, Bristol, United Kingdom
[2] University College London, London, United Kingdom
alexander.wilkinson.20@ucl.ac.uk

Abstract. Accurate and fast simulation of HPC job scheduling is an important tool for exploring the effect of different scheduling strategies on production systems and for providing insight into future HPC design. Current realistic simulations are computationally intensive and cannot provide a rapid feedback loop to facilitate the development of novel scheduling strategies. This work presents a lightweight simulation of the workload manager Slurm that is able to accurately reproduce the performance of the UK's national supercomputer ARCHER2 using historical workload accounting data. The simulation achieves a speed up of ~ 400 over a period of full system utilisation, allowing for months of activity to be simulated in hours, while maintaining wait times accurate to 7%. The simulator design supports incorporating external factors into scheduling to enable comparison of power-aware strategies. By using the simulation to evaluate the effect of multiple possible scheduling changes to ARCHER2, focusing on improving power management, the potential to provide insight into configuration changes and extensions to the scheduling logic is demonstrated.

Keywords: HPC Scheduling · Simulation · Slurm

1 Introduction

A well configured workload manager is a critical part of running an efficient HPC system. It is the mechanism through which the system is controlled, allowing for a balance to be struck between maximising utilisation, minimising wait times, ensuring fair allocation among users, and meeting any specific system requirements. Current workload managers, notably the popular open-source workload manager Slurm [1], are capable of striking this balance but have a large configurable parameter space that needs to be tuned and explored to do so. In addition to this, as workloads grow larger and more complex, extensions to current workload managers will be desirable to allow for delicate control of system power usage and integration of new advanced scheduling algorithms.

The important work of exploring configurations and new algorithms is difficult on a production system due to the risk of compromising system efficiency.

This can lead to conservative parameter choices and a reluctance to employ new scheduling strategies. Simulation can be used to provide the insight necessary to find more optimal positions in the parameter space without risk. It allows for different combinations of parameters and the integration of novel scheduling algorithms to be evaluated without risk. Although this principle applies to all workload managers, this work attempts to simulate Slurm specifically. Many features of Slurm are common to all workload managers, but focussing on a single piece of software is necessary to ensure a uniform format for job traces and configuration data, and to reproduce any particularities for validation purposes.

There are many grid scheduling simulations that can be used to study scheduling strategies, such as GridSim [2] and SimGrid [3]. Although these frameworks can be configured to study HPC systems, they are not well suited to suggesting specific scheduling changes. Simulation of Slurm specifically was initially developed in [8] and has since been iterated on numerous times with attempts to improve accuracy and performance [6,11,12]. Its potential to improve system performance has been demonstrated in [9] and [10] which build frameworks for using the simulation to test configuration changes on production systems. The common approach of these works is to submit jobs to Slurm from a trace and emulate job allocation and termination messages from the daemons running on compute nodes to the controller daemon. Speed up is achieved by skipping through time between events. These simulators have generally been able to reach high accuracies and successfully scale to large systems and workloads. However, by starting from the Slurm source code they struggle to achieve the speed up necessary for any significant parameter exploration on large systems where they are most needed. In addition, exploration of new functionality requires significant time investment as it requires implementation as a plugin to Slurm. This may be unfeasible if the functionality is not yet proven to be beneficial to a wide number of systems.

Current research into HPC scheduling, especially those taking machine learning approaches, tend to create custom scheduling simulations to evaluate algorithms [4,7]. These often capture some level of job priority calculation and backfilling but do not include many features present in production workloads such as resource limits, node failures, and fair share algorithms. This means that they are not suitable for studying the integration of these new algorithms into production HPC systems. A simulation that can sufficiently model a production system while remaining fast and easily extendable would benefit HPC scheduling research.

This work proposes a fast simulation that includes only the features of Slurm directly relevant to scheduling. Even with Slurm's implementation recreated in full, stochasticity in routine execution guarantees that the precise order of job submission will diverge for a constant stream of job submissions. For this reason, the simulation should be able to accurately reproduce the dynamics of a real system without trying to reproduce the design of Slurm. The simulation is evaluated through how well it can recreate aggregate behaviours and respond to changes in characteristics of the workload over time. The simulation is built

from scratch and validated using a job trace from the production HPC system ARCHER2. A less thorough validation is performed with other systems. Its use is then demonstrated by studying the effect of changes to configuration and scheduling logic on system performance over historical workload data.

2 Simulation

2.1 Overview

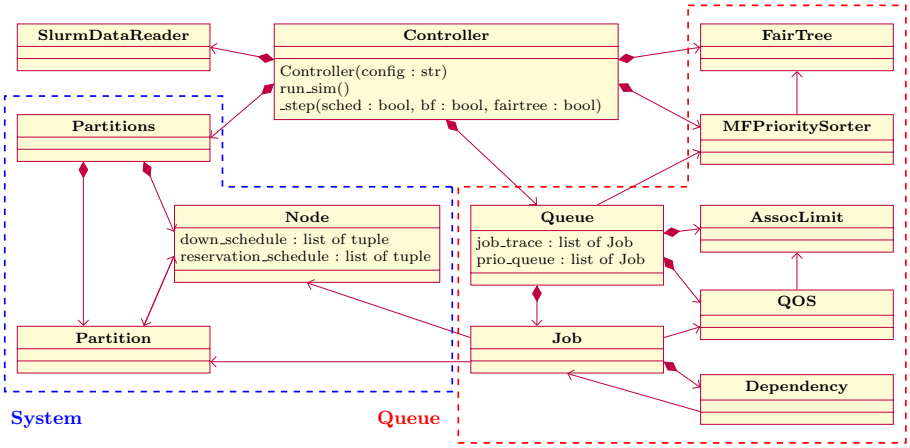

Fig. 1. Class diagram for scheduling simulation. Only a handful of important methods and attributes are shown for clarity. Classes are grouped into system and queue to indicate the general component of scheduling each collection is responsible for.

The simplified class diagram in Fig. 1 gives an overview of the simulator's design. The class *Controller* steps through time and controls which operations are executed at each time step. It is also where the main scheduling and backfilling loops are implemented. The collection of classes labelled queue are responsible for providing a priority sorted queue of jobs waiting to be scheduled and for determining which jobs are allowed to be started at any given time. Job submission from the historical data is simulated here. The collection labelled system is responsible for tracking the current state of the system i.e. the compute nodes. The controller decides which jobs to schedule based on data from these two collections.

The simulation is implemented in standard library Python with some use of libraries for CSV file manipulation. The system is configured using a YAML file that contains the locations of the Slurm configuration file and Slurm accounting data dumps for job data, node events, quality of service (QoS), active reservations, and association (4-tuple of user, cluster, partition, and account used

in Slurm accounting) data. Scheduling parameters set by the configuration file can be overridden by explicitly setting them in the YAML file. More substantial changes and extensions to the scheduling can be achieved by directly modifying class methods. This design is intended to be used to realise different scheduling strategies for a production system over part or all of its historical job trace.

2.2 Features

The simulation includes numerous features both general to scheduling and specific to Slurm that are either implemented in their entirety or partially. The three features most relevant to scheduling are discussed in this section.

Backfilling A conservative backfilling algorithm [5] is implemented to run alongside the main scheduling loop at configured intervals. Backfilling is a computationally expensive operation that can take a long time to complete. Large HPC sites will typically need to configure Slurm's backfilling thread to periodically yield locks so that other operations can be allowed to complete. The backfilling will then continue operation, ignoring any changes to system state that occurred during the lock release interval. To capture this in the simulation, the backfilling executes in steps of the lock yield interval parameter. Between these steps, the main scheduling loop is allowed to run if triggered.

Resource Limits The *QOS* class is used to track resource use and report if a job should be considered for scheduling at a given time. The hierarchy between limits set in the QoS and limits set at the user association level is implemented also. Slurm has an additional partition level hierarchy which is not implemented at this time. Any resource limits that will prevent a job from running indefinitely e.g. MaxWallDurationPerJob or MaxTRESPerJob, do not need to be implemented since they result in the job being absent from the trace. Of the subset that are relevant for the simulation, the majority are implemented.

Fairshare A ubiquitous feature of Slurm is its multifactor priority plugin. It is used to sort the queue according to a hierarchy of factors that take into account advanced reservations, partition, and multiple job properties including wait time, size, and a fairshare factor. The fairshare calculation is fully implemented with the fair tree implementation [1]. A rooted ordered tree of associations is constructed and used to sort users by their usage relative to allocated system shares.

2.3 Limitations

A key limitation in any simulation making use of historical Slurm data comes from missing information in the Slurm accounting database. Some job information such as dependencies and requested nodes are not explicitly recorded. They

can sometimes be extracted from the submission line but if set inside the submission script or using environment variables it may not be possible to recover them. In addition to this, records of completed reservations are not stored. This means that a reservation history would need to be kept externally or a synthetic set of reservations generated. Lastly, there can be discrepancies coming from human interaction with the system e.g. holding jobs and changing user resource limits, that cannot be replicated.

The simulation was developed by implementing features of Slurm in order of their assumed importance in making scheduling decisions. In this way, the simulation was incremented until the subset of Slurm required to reproduce scheduling behaviour on the HPC systems considered for validation was implemented. This subset encompasses the majority of the desirable features. Many of the missing features are options that slightly alter already implemented parts of the simulation and would require minor changes to incorporate. Notably missing is the ability to specify consumable resources other than compute nodes. The scheduler already chooses nodes from all valid nodes based on the configured node weight, so checking, for example, CPU count would not require much change to the structure of the simulation. Features that are expected to require a more substantial effort to implement are job preemption and heterogeneous job support.

Due to the aforementioned limitations, this work is presented as a demonstration of the ability of a lightweight scheduling simulation to provide useful insight into scheduling strategies. The simulation has the potential to be expanded to be able to produce accurate simulation of any HPC system regardless of its configuration and workload characteristics. At this time however, the simulation does not have sufficient coverage of Slurm's features to guarantee this.

3 Validation

3.1 ARCHER2

ARCHER2 is the UK's national supercomputer consisting of 23 HPE Cray EX cabinets forming a network of 5,860 compute nodes. The workload manager configuration as of the time of writing is described here. The system uses Slurm version 21.08 with a configuration refined from multiple years of operation. Nodes are assigned to overlapping standard and high memory partitions and jobs to a QoS depending on their size, length, and importance. The workload comes from a wide range of academic fields and includes job sizes ranging from a one to a few thousand nodes. System utilisation is consistently over 90% with a queue of at least a few hundred jobs. This means that even minor changes to job scheduling can create significantly different submission orders.

As input to the simulation job data was collected from the Slurm accounting database for four months of operation from October 2022, totalling approximately 600,000 jobs. Where possible, job dependencies and reservations were parsed from the submission line stored in the record. The system state was captured at the end of the four month period and used to configure the simulation.

3.2 Results and Discussion

As discussed previously, the simulation aims to accurately reproduce aggregate features of the scheduling while allowing the precise order of job submission to deviate. To evaluate this, the two week moving average wait time for the results of the simulation and for the true submission times taken from the data are compared. This is shown in Fig. 2 over the four month period. Over this time period the simulation differs from the data by a mean absolute percentage error of 7.3% The simulation closely matches the high frequency trends of the data, suggesting that as groups of jobs enter the moving average they are treated in the same way relative to current jobs in the window for both the simulation and data. Low frequency behaviour is generally captured well but for the wait time it is consistently lower. This is likely due to imperfect historical reservations which were partially recovered for ARCHER2.

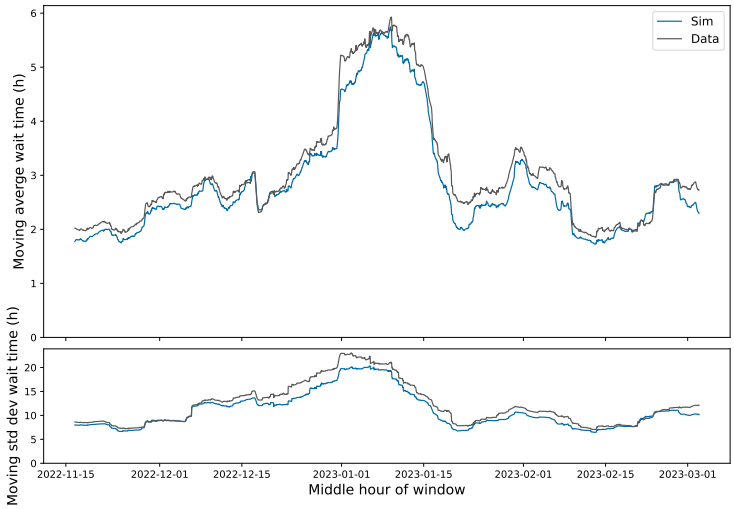

Fig. 2. Moving average wait time and standard deviation for jobs in a two week submission window.

To examine this behaviour over time closer, Fig. 3 shows the two week moving average wait time for jobs from a selection of QoS. Jobs from these QoS differ not only by their resource limits but also by their typical usage, for example *taskfarm* jobs will tend to be from a small group of users submitting a large number of jobs. This means that accurate simulation of their wait times depends on the implementation of many scheduling components in addition to resource limits. The wait times for each QoS are generally well matched between data and simulation. The *lowpriority* jobs are configured to be considered for scheduling only after all jobs from other QoS have been. The resulting long wait

times are correctly reproduced in the simulation. Jobs from the *largescale* QoS have significantly shorter simulation wait times. The simulation usually spends much less time attempting to schedule these large jobs, a pronounced example of this is in Fig. 4. The reduced time spent at lower system utilisation is also a factor in the slightly shorter wait times seen in Fig. 2. It is not likely that the shorter *largescale* wait times are caused by the priority calculation or backfilling algorithm as all other metrics suggest these are implemented correctly. The cause may be a constraint, such as a maximum number of leaf switches, set outside the submission line or perhaps more complicated behaviour related to Slurm's design that is not captured by the simulation. Further investigation into the system state during the scheduling of such jobs is required to understand the cause.

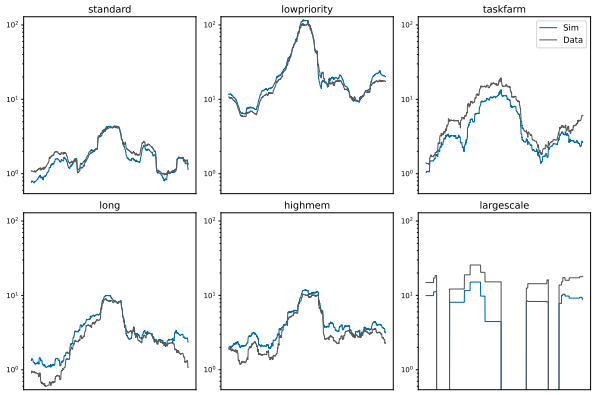

Fig. 3. Moving average wait time over a two week window for jobs from different QoS.

In Fig. 5, the simulated average wait time is compared with the average wait time from data for users with the highest overall system usage over the four month time period. The simulation matches the data reasonably well but has wait times that are generally shorter for most users for the previously stated reasons. This shows that the simulation can be used to provide insight into the effect of scheduling changes down to the user level.

To study the behaviour of the simulator, the relationship between wait time and certain job properties can be examined. Two-dimensional histograms of wait time against both requested nodes and requested time are shown in Fig. 6. Ignoring any correlations between job properties, the distributions are primarily a result of how difficult a job of a given size or length is to backfill. The relationship of wait time to these job properties matches reasonably well between simulation and data, but with wait times for the simulation shifted to earlier times. This suggests that the backfill algorithm is correctly implemented but the simulation operates with a higher throughput.

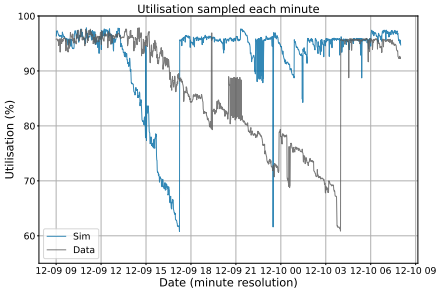

Fig. 4. System utilisation during scheduling of a large scale job.

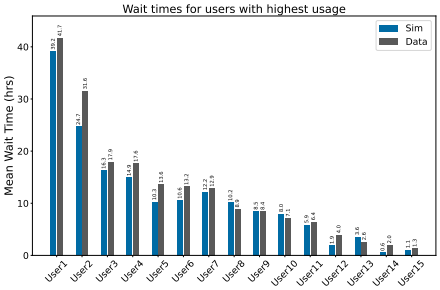

Fig. 5. Mean wait times for top fifteen users by usage.

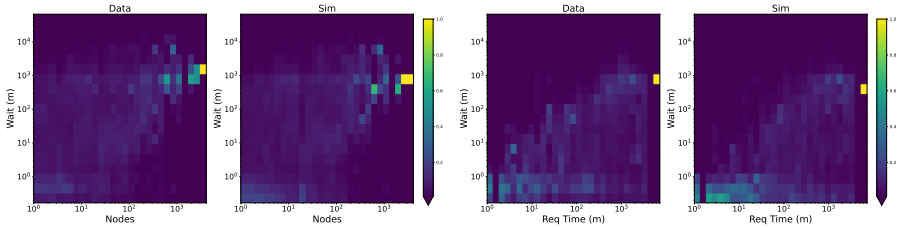

Fig. 6. 2D histograms of job size in nodes (left) and job requested time (right) against wait time. Bin counts in each column, corresponding to a range of job sizes or requested times, are normalised to unity.

Running on a 64-core AMD EPYC 7742 process at 2.25 GHz, the simulation completes in approximately 7 hours and 20 minutes. This represents a speed up by a factor of 400. The simulation time is dominated by the backfilling algorithm which requires frequent I/O opertations to maintain a map of current and future job placement. For ARCHER2 during the simulated time period, the queue contains 541 jobs on average and the backfilling loop is configured to run for 30 s over a maximum of 1,000 jobs and then sleep for 30 s. Simulation time will scale with queue length up to this configured maximum. For larger maxima, the backfill thread will typically need more time to complete and so will run less frequently. This means that simulation times for other systems should not increase significantly. In the current implementation simulation time also scales poorly with queue size due to liberal use of sort operations. Refactoring to use heap data structures would help mitigate this.

3.3 Other Systems

LUMI LUMI is a HPE Cray EX system that is currently the third fastest globally. It is made up of 4,096 compute nodes in both GPU and CPU partitions.

The component of the system partitions allocatable by node was loaded into the simulator along with a 3 month job trace. The simulation successfully ran over this time period after implementing some additional Slurm features. The simulated wait times matched the truth with reasonable accuracy, but a lack of historical reservation data lead to significant deviations for some time periods of the simulation. The details of the simulation results cannot be shared at this time as permission is yet to be received from administrators.

Peta4 Peta4 is a 1,468 node supercomputer forming part of the Cambridge Service for Data-Driven Discovery. Anonymised Slurm data was collected over a two month period and the system state and job trace was successfully loaded into the simulator. However, the simulation does not currently have the features necessary to sufficiently reproduce the scheduling behaviour. Extending it with these features is an important direction for future work since its workload has a much higher throughput than ARCHER2 with approximately four million jobs over the two month period. This would make it a valuable test of the simulator's performance.

4 Experiments

4.1 High Priority Jobs

It may be desirable for a HPC system to offer a high priority QoS for producing time sensitive outputs. Before implementing this, the simulation can be used to provide insight into its impact on system performance. An example of this is shown in Fig. 7, where the effect of putting different proportions of *standard* QoS jobs into a *highpriority* QoS at random is tested. High priority jobs have their wait times reduced by a factor of 8 with little effect on other QoS when 5% of standard jobs are submitted as high priority. As this proportion grows, its impact on other QoS grows also until at 30% they begin to significantly impact scheduling of large jobs and starve high memory jobs of resources. The average system performance across all jobs is negatively impacted when there is a small number of high priority jobs but for larger numbers it is not. This is likely due to reaching a threshold where high priority jobs form a substantial portion of the queue rather than acting as sporadic disruptions to the backfilling. The simulation can be used to study the impact introducing this QoS at different granularities and experiment with the different possible implementations and uses of a high priority QoS, highlighting its value to production HPC systems.

4.2 Large Scale Jobs at Peak Times

Slurm can be configured to associate energy counters collected from compute nodes with running jobs. This is used to provide a consumed energy for each job, allowing for average job power to be inferred from the accounting data. For ARCHER2, energy accounting is configured. To use this data to estimate total

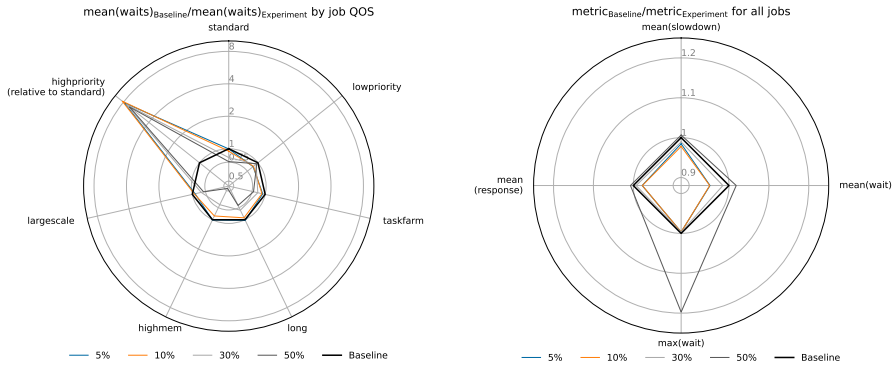

Fig. 7. Effect of different usages of a high priority QoS on wait times by QoS (left) and system-wide performance metrics (right).

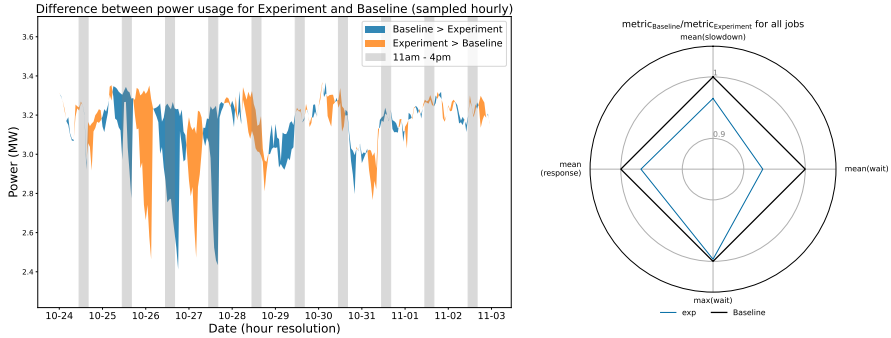

Fig. 8. Effect of holding scheduling of *largescale* jobs until morning on power (left) and system-wide performance metrics (right)

system power draw at any given time, the power reported by nodes with running jobs is fitted to power reported at the cabinet level with an occupancy term to account for idle nodes. This allows the simulator to be used to analyse the total power usage over time for different scheduling strategies.

ARCHER2 uses a large scale QoS to manage jobs over one thousand nodes. Although these represent a very small fraction of the workload, they have a significant effect on the system when being scheduled. When being scheduled, the backfilling algorithm will struggle to find jobs with requested times short enough that they will not interfere with the start time of the large job. This results in a significant drop in system utilisation over the hours leading up to the large job being submitted. This effect is inevitable for priority based schedulers. In the context of power management it would be desirable to ensure that this period of low utilisation overlaps with the peak hours of the day when electricity

is in high demand and cooling is under the most stress. The simulator can be used to evaluate strategies that attempt to do this.

The *largescale* QoS is adapted to hold all jobs until a specified time the next morning. The simulator determines the time they are allowed to be scheduled from a small set of switch cases dependent on job size and with optimal submit times estimated from average behaviour. The effect of this modification on system power and performance can be seen in Fig. 8. The scheduling of large jobs result in reductions in power usage of up to 0.8 MW depending on their size. Even using the rudimentary method for choosing the release time, the utilisation drops are fairly well aligned with peak times. It would be interesting to explore more advanced algorithms that identify jobs expected to be hard to backfill and hold them until a predicted optimal time, thus trading low utilisation during peak times for higher utilisation at off-peak times. This could be explored by extending the simulator with elements from machine learning libraries. Also shown in the figure is the impact of this scheduling modification on performance metrics. The drop in performance is due to the scheduling of large jobs coinciding with the peak submission time in the day, rather than a drop in throughput. Further examination of the effect on particular users or jobs with certain characteristics can be extracted from the simulation results as required.

5 Summary

This work presents a lightweight scheduling simulation to fill the gap between simple custom schedulers and full simulation of workload managers. The simulator's design incorporates the majority of Slurm's features relevant to scheduling, allowing for simulation of some HPC systems. It's design is easily extendable and well suited for rapid testing of scheduling strategies. The simulator was validated using a recent historical workload from the production system ARCHER2. The aggregated performance of the simulator closely matches the data down to the granularity of a single user despite limitations in recovering complete job properties from Slurm accounting data. Further investigation is required to understand the significantly shorter simulated wait times of a small number of large jobs. The potential of the simulation to provide insight into the effects of scheduling strategies is demonstrated with two experiments. The latter experiment shows how Slurm's energy accounting enables the exploration of scheduling strategies focused on power management using the simulator.

An important avenue for future work is the further development of the simulation to achieve coverage of Slurm's features sufficient to simulate the majority of HPC systems. This would permit validation with a wide range of systems. Using the simulation to demonstrate significantly more advanced scheduling strategies than those presented previously is another valuable direction for future efforts.

Acknowledgements. This work used the ARCHER2 UK National Supercomputing Service (https://www.archer2.ac.uk). We would like to thank Andrew Turner from EPCC for valuable discussions during development and validation

with ARCHER2. Thanks to the Cambridge Service for Data Driven Discovery (CSD3) at the University of Cambridge for providing access to workload data. We acknowledge support of CSC who provided access to scheduler data from the LUMI system, owned by the EuroHPC Joint Undertaking.

References

1. Slurm workload manager. https://slurm.schedmd.com/documentation.html, accessed: 2023-03-31
2. Buyya, R., Murshed, M.: Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. Concurrency and Computation: Practice and Experience **14**(13-15), 1175–1220 (2002), https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.710
3. Casanova, H., Giersch, A., Legrand, A., Quinson, M., Suter, F.: Versatile, scalable, and accurate simulation of distributed applications and platforms. Journal of Parallel and Distributed Computing **74**(10), 2899–2917 (Jun 2014), http://hal.inria.fr/hal-01017319
4. Fan, Y., Lan, Z., Childers, T., Rich, P., Allcock, W., Papka, M.E.: Deep reinforcement agent for scheduling in hpc. In: 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS). pp. 807–816. IEEE (2021)
5. Feitelson, D., Weil, A.: Utilization and predictability in scheduling the ibm sp2 with backfilling. In: Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing. pp. 542–546 (1998)
6. Jokanovic, A., D'Amico, M., Corbalan, J.: 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). pp. 72–82 (2018)
7. Kassab, A., Nicod, J.M., Philippe, L., Rehn-Sonigo, V.: Assessing the use of genetic algorithms to schedule independent tasks under power constraints. In: 2018 International Conference on High Performance Computing & Simulation (HPCS). pp. 252–259 (2018)
8. Lucero, A.: Simulation of batch scheduling using real production-ready software tools. In: Proceedings of the 5th IBERGRID (2011)
9. Martinasso, M., Gila, M., Bianco, M., Alam, S.R., McMurtrie, C., Schulthess, T.C.: 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. pp. 320–332 (2018)
10. Rodrigo, G.P., Elmroth, E., Östberg, P.O., Ramakrishnan, L.: Scsf: A scheduling simulation framework. In: Job Scheduling Strategies for Parallel Processing. pp. 152–173. Springer International Publishing (2018)
11. Simakov, N., Innus, M., Jones, M., Deleon, R., White, J., Gallo, S., Patra, A., Furlani, T.: A Slurm Simulator: Implementation and Parametric Analysis, pp. 197–217 (2018)
12. Trofinoff, S., Benini, M.: Using and modifying the bsc slurm workload simulator. Slurm User Group Meeting 2015 (2015), https://slurm.schedmd.com/SLUG15/BSC_Slurm_Workload_Simulator_Enhancements.pdf