Evaluating SLURM simulator with real-machine SLURM and vice versa

Ana Jokanovic
Barcelona Supercomputing Center (BSC)
Barcelona, Spain
ana.jokanovic@bsc.es

Marco D'Amico
Barcelona Supercomputing Center (BSC)
Barcelona, Spain
marco.damico@bsc.es

Julita Corbalan
Universitat Politecnica de Catalunya
Barcelona, Spain
julita.corbalan@bsc.es

***Abstract*—Having a precise and a fast job scheduler model that resembles the real-machine job scheduling software behavior is extremely important in the field of job scheduling. The idea behind SLURM simulator is preserving the original code of the core SLURM functions while allowing for all the advantages of a simulator. Since 2011, SLURM simulator has passed through several iterations of improvements in different research centers. In this work, we present our latest improvements of SLURM simulator and perform the first-ever validation of the simulator on the real machine. In particular, we improved the simulator's performance for about 2.6 times, made the simulator deterministic across several same set-up runs, and improved the simulator's accuracy; its deviation from the real-machine is lowered from previous 12% to at most 1.7%. Finally, we illustrate with several use cases the value of the simulator for job scheduling researchers, SLURM-system administrators, and SLURM developers.**

***Index Terms*—SLURM, simulator, job scheduling, workload, simulation, HPC**

I. Introduction

Optimizing a job scheduler for the best HPC system performance and user experience is a complex, multi-dimensional problem. In today's HPC systems, SLURM [1] is a widely used plugin-based job scheduler providing multiple optimization options, either through tuning the configuration parameters or by implementing new plugins and new scheduling and select policies. Setting up all these options in SLURM for the optimal performance would require a detailed parametric analysis to understand the effect of multiple, sometimes interdependent factors. Performing trial and error approach on the real machine might be slow, impractical or most probably, it would negatively impact the performance of the system. On the other side, theoretical models may not include sufficient details, can be imprecise and lead to wrong decisions.

Several years ago, the first version of SLURM simulator was created by a SLURM system administrator from Barcelona Supercomputing Center, A. Lucero [2], with the idea to allow SLURM administrators to do their parametric analysis in the SLURM code itself without affecting the system performance. While the idea was promptly accepted by SLURM community, none of the existing versions of the simulator until today was brought to the level of precision and speed for correct decision and practical use.

Our team, as well, enthusiastically accepted the SLURM simulator, i.e., the latest version that was available at the moment [3], with the intention of implementing new job scheduling policies. However, very soon we realized it had many flaws which made it inaccurate for any serious study of scheduling policies. Our methodology, that allows us to compare the simulator with the real-machine SLURM, enabled us to quantify this imprecision, and it was significant. In this work, we present the experimental results and the methodology used. It demonstrates that when running the simulator with the same input and set-up multiple times the average variation of a job's start time due to the simulator's inconsistency was up to 22 minutes. This variation can significantly influence the scheduling policy evaluation. For example, in one of the experiments, at a specific job we detected that the variation in the job's wait time among 10 simulation runs was from 92 minutes to 372 minutes for a job of 1 minute duration, resulting in the variation of job's slowdown from 93 to 373, i.e., 301%. Also, our experiments demonstrate that the system metrics deviation from the real-machine was up to 12%. More importantly, we show that the error was clearly related to the simulated system load.

Further, we worked on identifying the causes of the inaccuracy and low speed in the simulator's code and introduced multiple fixes and improvements in the inherited version of the simulator. The evaluation of these improvements and the comparison with the previous version is presented in this work, as well. Our main contributions are:

- We removed the random variation from the simulator and made our simulator deterministic across multiple runs for the same input and set-up.
- We improved the accuracy, i.e., lowered the deviation of the simulator's system metrics values from the real-machine ones from previous 12% to at most 1.7%.
- We improved the simulator's performance by 2.6 times.
- We presented our methodology for the evaluation of the SLURM simulator on the real machine.
- We ported the simulator to the latest SLURM version, 17.11.
- We implemented converters between Standard Workload Format (SWF) [4], [5] and the simulator's input trace and between SLURM's completion log and SWF.

Once validated and evaluated regarding accuracy and performance by comparing SLURM simulator results with real-

machine SLURM executions, we have used SLURM simulator to *evaluate* SLURM. We have not evaluated SLURM as a workload manager, but rather its specific configurations or parts. For instance, we measure the impact of changing the backfilling interval on system performance metrics, average slowdown, average wait time and average response time, and on the scheduler itself in terms of scheduling time. Four different use cases for the SLURM simulator have been selected, and in all of them, the impact on system performance metrics and, on specific scheduler metrics was evaluated in the simulator.

The paper is organized as follows. Section II gives a brief description of the SLURM simulator's environment and its internals, as well as the description of the problems in the previous version and the corresponding fixes we did. Section III explains in detail our validation methodology and presents the results from the validation experiments divided in the subsections on *consistency*, i.e., variations across different simulation's runs, *accuracy*, i.e., simulated results vs. real-machine results, and *performance*, i.e., simulator's execution time vs. total simulated time. Section IV presents the reverse analysis. Namely, we present several example use cases of SLURM simulator for evaluating the real SLURM that can be of value for anyone dealing with the optimization of the SLURM performance. Section V gives an overview of the previous works on SLURM simulator and some discussion of theoretical models. Section VI concludes the paper and gives our plans for the future work.

II. SLURM Simulator

SLURM simulator was developed initially at Barcelona Supercomputing Center [2] and passed several iterations of improvements at CSCS [6], and at Umea University/Berkley Lab [3]. Here, we briefly explain the main inputs and outputs, and main components and characteristics of the SLURM simulator that have been present in either all of the versions or in most of them.

A. General description

As shown in Figure 1, SLURM simulator receives as an input the standard SLURM configuration file, *slurm.conf*, that allows for the specification of the system architecture, as well as job scheduler details such as scheduling and selection policies, etc., and a trace file in a binary simulator's format. We have provided a converter from Standard Workload Format (SWF) [4], [5] to simulator's trace format to enable simulator to use a vast amount of existing real-machine and modeled logs in the online repositories such as Feitelson's [7]. The SLURM simulator generates standard SLURM outputs, such as SLURM controller daemon's and SLURM daemon's logs, job completion log, SLURM database files, etc.

Figure 1 shows three main components, i.e., three processes of the SLURM simulator, SLURM simulator's manager, SLURM controller daemon and SLURM daemon.

- *sim_mgr* is in charge of controlling simulated time, i.e., it increments it by one second in each simulator's iteration. Also, it reads the input trace and submits the jobs to the SLURM controller when its arrival time is reached. The job submission is done using SLURM's API *sbatch*.
- *slurmctld* is a standard SLURM controller daemon, and all the core functions of the controller, such as job scheduling and selection policies are the original SLURM code.
- *slurmd* is a simplified SLURM daemon since the jobs' execution is not simulated. It receives the job duration from SLURM controller, sets the job's end time as a future event, and notifies the controller when the job's end time is reached. One *slurmd* process is in charge of all the nodes.

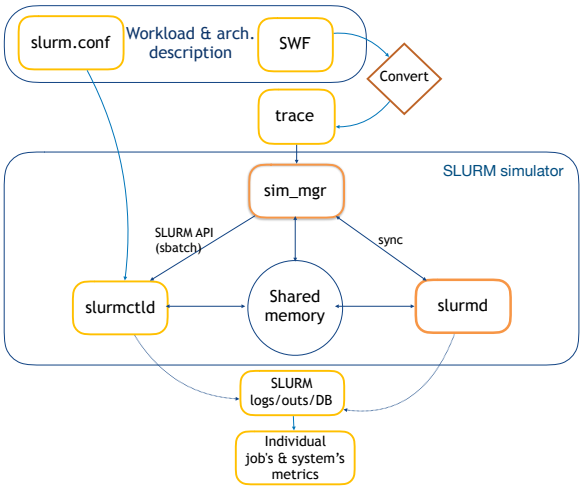


Fig. 1. SLURM simulator's processes.

All the time functions are redefined in order to return simulated time instead of real time.

B. Improvements

Our starting point was the version of Rodrigo et al. [3]. The latest available version at that time was on SLURM version 14.02. We encountered a set of errors when comparing to real machine execution. In particular, we found unexpected delays happening at job arrival, job end, and in the job duration. The causes of these errors were due to problems in processes synchronization, RPC related delays and schedulers calls. Here we give details of the problems found and the solutions we provided.

1) Synchronization of the simulator's processes: Previous versions of SLURM simulator implemented poor synchronization between processes. In Rodrigo's simulator, the synchronization is implemented with a semaphore and a shared variable. Three simulator's processes, all *slurmctld*'s RPCs and the backfill thread use the semaphore to get exclusive access for editing the shared variable. To coordinate the simulation, all the processes keep checking a specific value for the variable to start processing. If the value is not the right one, they sleep and re-check. Reading the variable is not protected by the semaphore so, in the case of concurrent reading and writing, the simulation would get into undefined

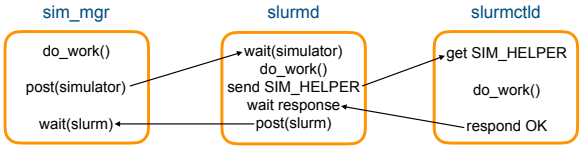


Fig. 2. New synchronization of the SLURM simulator's processes.

behaviors, i.e., a race condition occurs. This incorrect synchronization produces the loss of simulated seconds and job events happening at the wrong time. We implemented a a multi-semaphore synchronization approach (Figure 2). *Slurmd* unlocks the first semaphore after it finishes all the message interchanges with *slurmctld* and it is consumed by the *sim_mgr* to pass to the next simulated second. There is no need for a semaphore between *slurmd* and *slurmctld* since it can be done via *RPC*s, as sender thread is always blocked until it receives an OK response from the receiver. The second semaphore is incremented by *sim_mgr* to unlock *slurmd* after one simulated second has passed and it completed the processing of all events for that second, i.e., all new job requests are sent. The *slurmd*, once unlocked, sends a message called *SIM_HELPER* to the controller, that will respond with an OK message when it terminates all its pending activities. By using this synchronization, we were able to coordinate the simulation processes correctly without losing simulated seconds, and at the same time we sped up the simulator.

2) RPC exchange related delays: The second main point of improvement is related to *RPC* exchange. As we said, RPCs are blocking, but a special behavior occurs at the job end. In this scenario *slurmd* sends a request for terminating a job, *slurmctld* responds and it also sends a request for executing an epilog for the finished job. The *slurmd* immediately responds with OK, closing the RPC, thus unlocking *slurmctld* that was waiting for a response and, after executing the epilog, it sends a new message to the controller marking the real job end. If *slurmctld* is not aware of the remaining epilogs pending to arrive it ends its simulated section, unlocking the simulation to proceed. Epilog messages can arrive with one or more seconds of delay, increasing the duration of the jobs. SCSF simulator only partially addressed this problem using *sleep*s, which caused a slowdown in the simulation and altered job duration, especially on high load. Shared counter variables used by SLURM daemon's threads allow eliminating delays caused by job's end. In synthesis, we wait for the number of arrived epilogs messages to be equal to the number of ending jobs in that second. Since the counters are shared by threads, we implemented read and write locks to control access to the counters.

3) Delay in scheduler calls: Rodrigo's version was using a time-triggered FIFO scheduler, more similar to how the backfill scheduler works. We implemented an event triggered FIFO scheduler, simulating the original SLURM scheduler, that is triggered at job's arrival and at job's end. Starting the scheduler at job's end increases the efficiency of the scheduler, that does not leave resources unused, delaying the start of new jobs. Similar behavior happens at job's arrival, in the case the job can be run immediately.

4) Other improvements: An important effort was porting our improved SLURM simulator version to the latest SLURM version 17.11. Also, we enabled simulation to end when the last simulated job ends, previously, it required passing specific end-time. Backfill interval was hardcoded in the simulator's code, we made it editable from the *slurm.conf* as in regular SLURM. We implemented a set of new *slurm.conf* parameters for different parametric analysis. Also, we did a high number of fixes in the code regarding compatibility with operating systems, execution out of Virtual Machines environment, and multiple parallel executions in supercomputers nodes. We created various tools and scripts for launching and controlling the simulation, for conversion of input log from SWF to simulation input, and from output log to SWF, scripts for generating different SLURM configurations, automatic data extraction, and analysis.

III. SLURM Simulator validation

In this section, we intend to show how our improvements reflect in important points such as accuracy, consistency, and performance of the simulator and to do the first real-machine validation of the simulator that has not been done for any of the previous versions.

- *Consistency experiments*: we evaluate the variability of the simulator across multiple runs for the same input and configuration set up. We compare our improved version with the inherited version of the simulator.
- *Accuracy experiments*: we evaluate the precision of the simulator comparing it to the real-machine SLURM execution for the same input and configuration set up. We compare both - our improved and inherited version - to the real-machine SLURM using typical /system performance metrics.
- *Performance experiments*: we evaluate how fast is the simulator's code comparing it to the inherited version's code executions on the same machine and for the same input and configuration set up. We report metrics execution time and speedup.

We will compare our improved version with the inherited version from Rodrigo et al. [3]. We will use the following notations to distinguish among the versions:

- *SIM_V17* [8]: our improved version ported to the SLURM 17.11
- *SIM_V14* [8]: our improved version ported to the SLURM 14.02, the same version of SLURM as inherited simulator's version, necessary for comparison of simulator's performance.
- *SIM_SCSF(_V14)* [3]: inherited version, i.e., our starting point which was on the SLURM 14.02 version when we started improving it.

A. Workloads

We will use in total eight workloads for our validation experiments - four big workloads of 5000 jobs for the experiments on simulator's consistency and simulator's performance,

Log	Arr. pattern	#jobs	System size	Max job size	Avg wait time (s)	Avg response time (s)	Avg slowdown	Simulated time
big log 1	ANL	5k	3456 nodes	128 nodes	31457	40067	932	4,4 days
big log 2	CTC	5k	3456 nodes	128 nodes	596	9046	8,83	4,5 days
big log 3	KTH	5k	3456 nodes	128 nodes	268	8958	7,47	5,2 days
big log 4	SDSC	5k	3456 nodes	128 nodes	17757	26431	567	4,1 days

TABLE I
WORKLOAD LOGS' CONFIGURATION PARAMETERS AND PERFORMANCE METRICS VALUES

their details are in Table I, and four small workloads of 200 jobs for the experiments on accuracy. In this section, we describe how we generated these logs, and present a set of configuration parameters and metrics for each of the logs.

1) Big logs for consistency and performance experiments: We have generated workload trace files, i.e., logs using the model proposed by Cirne and Berman in [9] based on the analysis of many workloads coming from real workload traces. This model includes user behavior concerning job submission, i.e., job arrival time patterns, job sizes according to system size, system load, etc. We have generated a workload with 5000 jobs in a system with 3456 nodes with 48 CPUs per node for four different arrival patterns as given in Table I. In the Table I, to give an idea of the system load created by each of the workload logs, we also include a set of performance metrics values that we obtained from running these logs in the simulator.

- *Average wait time*: average time passed between job's submission and job's start
- *Average response time*: average time passed between job's submission and job's end
- *Average slowdown*: slowdown is the ratio between response time and the duration, i.e., the execution time of the job.

2) Small logs for accuracy experiments on the real machine: The intention in this set of experiments is to compare simulation results with the real-machine results. Therefore, we had to limit the system size, i.e., the number of nodes we reserve on the real machine and the log size, i.e., wall clock time we request for this reservation, to make the experiments on the real machine feasible. We choose the log size of 200 jobs as being the maximum number of jobs from our set of applications that can be executed within 2h. The wall clock time of 2h was the upper limit of the fast job queue on our chosen production machine. We had to relay on this queue to get big enough number of experiments done on the portion of 10 real-machine nodes in a reasonable time.

The complete step-by-step process of creating CIRNE model [9] based real-applications workload for the execution on the real-machine is given in the flow-chart in Figure 3. First, we generate the logs using a CIRNE model for the system size, i.e., the number of nodes that we plan to reserve on the real machine, and maximum job size being the maximum number of nodes required by the real applications in the workload that we are going to use. As in the case of big logs, we create four logs, for four different arrival patterns: ANL, CTC, KTH, and SDSC. On the other side, we create a pool of NAS benchmarks with different input and job sizes, such that maximum job size matches the maximum job size in the

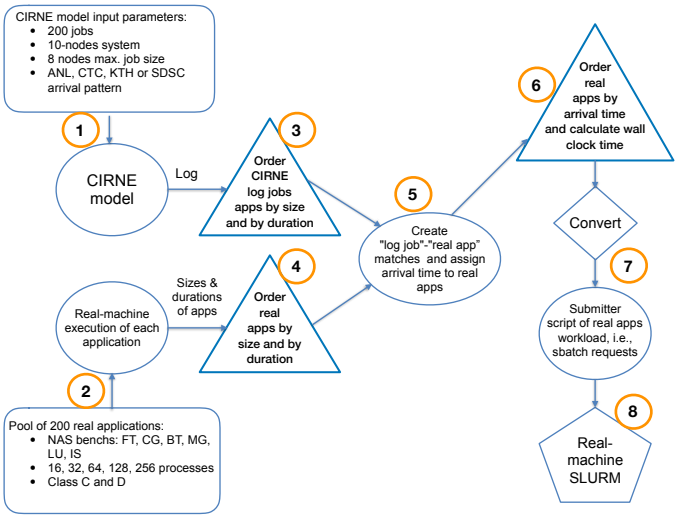


Fig. 3. Creation of real-applications workload based on CIRNE model flow chart.

CIRNE log and execute each of them on the real machine to collect their execution times. Then we map each benchmark to the similar job from CIRNE log based on its size and duration and assign to benchmark its arrival time. Since the durations of the benchmarks will not be exactly the same as the ones in the log's jobs, we make sure the wall clock times of the jobs in the newly generated workload is calculated with a good enough approximation. First, the ratio between the duration and the wall clock time for each job in the CIRNE log is calculated and then, it is used to multiply the duration of its real-benchmark match and get the real benchmark's wall clock time. Finally, we sort the real benchmarks by their newly assigned arrival time and convert this list to a submitter script. This submitter script is a list of *sbatch* requests with the corresponding number of seconds between them to match the arrival times of the applications. These *sbatch* requests are submitted to our SLURM over SLURM environment explained later on in Section III-C.

B. System and job scheduler configuration

Here we explain the set up of SLURM configuration file, *slurm.conf*, for each type of the experiments. For our consistency and performance experiments, we use the system configuration that corresponds to one of the machines to which we have access. The system size of the machine is 3456 computing nodes; each node has 2 CPUs and each CPU 24 cores. This set-up will be the exact configuration of the system in our experiments.

For the accuracy experiments, we use the system size of 10 nodes and the same node architecture configuration since the real-machine experiments will be executed on a 10-nodes

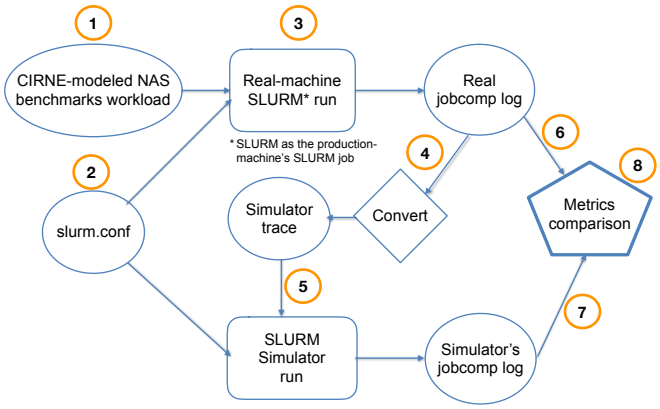


Fig. 4. The accuracy experiments flow chart.

portion of the mentioned reference machine. The SLURM and the SLURM simulator are configured to use linear select policy and FIFO&backfill job scheduling policies All the parameters regarding job scheduling are the same for all the experiments. The summary of the relevant system and job scheduler configuration parameters is given in Table II.

Configuration parameter	Consistency&Performance	Accuracy
System size	3456	10
Number of CPUs per node	48	48
Select policy	linear	linear
Scheduling policy	backfill	backfill
Backfill interval	30s	30s

TABLE II
RELEVANT SLURM CONFIGURATION FILE PARAMETERS

C. Real machine experiments and SLURM over SLURM environment

Since changing configuration of the production-machine's SLURM by a regular researcher is typically not permitted, we created an environment where we can launch the real SLURM as a job on a portion of N nodes of the production-machine. Thus, our SLURM is executed as a production-machine's job, in an exclusive mode, i.e., requested nodes are fully allocated for SLURM, and its daemons do not interfere with other jobs running in the system at the node level. It acts as a regular job scheduler for a sequence of real-applications' jobs submitted to it and being freely configurable by us. This environment allows unhindered configuration of set up parameters, and execution of predefined workloads on the real-machine, i.e., using the same set-up as for the simulators' versions that we want to compare to. The scheme of the steps we follow in our real-machine experiments intended for accuracy evaluation is shown in Figure 4. First, we create the real-applications' workload, as explained in III-A2. Then we execute this workload on the real machine submitting the jobs to our SLURM and collect job completion log. This log is converted to the input trace of the simulator and replayed in the simulator. From this simulation, we obtain simulator's job completion log. From the real machine log and simulator's log, we calculate system performance metrics and perform the comparison.

D. Consistency results

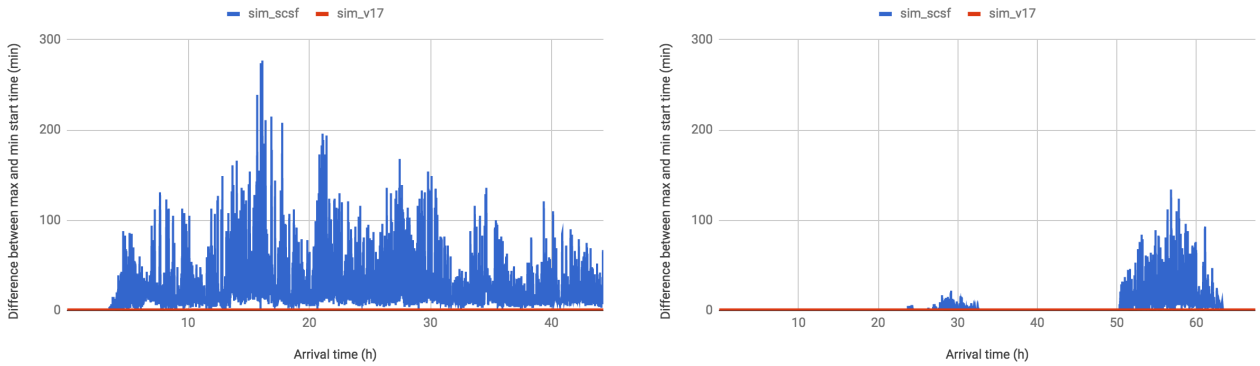
As we explained in Section II-B, we observed a significant variation in *SIM_SCSF* results for the same set up across different runs. Here we quantify this variation performing the simulations of four big logs on the system size of 3456 nodes. Figure 5 shows the variation range of start time for each job across ten different runs for four big workloads in two simulator's versions. *SIM_V17* has been improved and gives deterministic results, i.e., 0 variations across multiple runs for the same input, whereas, the inherited version, *SIM_SCSF*, varies significantly, from up to 108 minutes to up to 277 minutes depending on the workload. In the Figure 6 we give the characterization of the load for each of the workloads in terms of the total number of requested and busy nodes in time until the last, i.e., the 5000th job has been submitted. Figure 7 shows the correlation between the average slope of the load in time and the average error derived from the results in Figure 5 for each workload when running *SIM_SCSF*. We conclude that the higher variation in *SIM_SCSF* is due to higher system load, i.e., there is a correlation between the simulated system load and the *SIM_SCSF* simulator's variation which is consistent with our understanding of the problems found in the *SIM_SCSF* code.

E. Accuracy results

The same small workload is run on the real machine and in two versions of the simulator, *SIM_V17* and *SIM_SCSF* as explained in Section III-C and in Figure 4. In Figure 8 we compare the system performance metrics obtained from simulation runs to the system performance metrics obtained from real-machine runs. Simulator's version *SIM_SCSF* was run ten times, and the average results are presented, since, as we saw in the Section III-D, this version experiences significant variation across runs. Figure 8 gives the deviation of the simulator's system performance metrics from the real-machine case. It shows that *SIM_V17* version is quite close to the real-machine SLURM version. It deviates at most 1.7% in any of the system metrics, and in most of the cases, it is below 1%. On the other side, version *SIM_SCSF* deviates from 6% to up to 12% from the real machine SLURM. Since in Section III-D we already found the correlation between the system load and the error of the *SIM_SCSF*, we expect this deviation to be even higher when simulating bigger systems and workloads.

F. Performance results

We compare the speed of our simulator with the speed of *SIM_SCSF* simulator. Since, *SIM_SCSF* is on SLURM 14.02 version, for the comparison we use our improved version on the same SLURM version, 14.02, *SIM_V14*, to compare the same versions of the code. Also, it is important to mention that we run all the simulators on the same machine and in the same environment. Thus our comparison is valid. We also include the results for the performance of the *SIM_V17*, since this is our latest contribution to SLURM simulator's code. Simulator's speed is calculated as the ratio of total



(a) ANL log, 5k jobs

(b) CTC log, 5k jobs

(c) KTH log, 5k jobs

(d) SDSC log, 5k jobs

Fig. 5. Difference between maximum and minimum start time value for each job in the workload across 10 simulations of the same log and the same configuration set-up. Arrival time of a job in the workload (x-axis), the job's maximum variation in start time across 10 simulations (y-axis). Two simulator's versions are compared, *SIM_V17* (red line) and *SIM_SCSF* (blue line). Big logs of 5000 jobs are used on the system of 3456 nodes, as explained in Section III-A1. Each graph shows the workload for one of the arrival patterns, ANL, CTC, KTH and SDSC. There are 5000 jobs in each case, but they arrive in different time spans for different logs, thus, x-axes are not at the same scale. Note, y-axes are at the same scale.

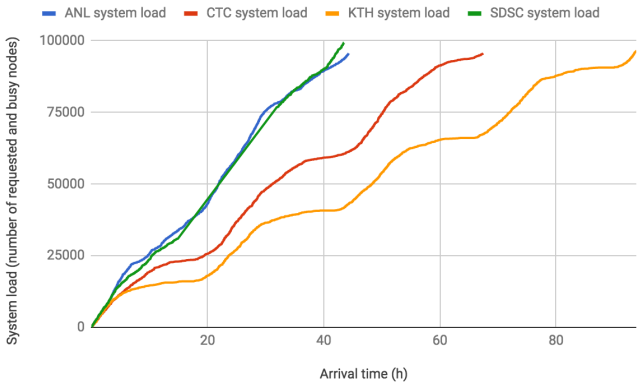


Fig. 6. System load. On the x-axis there is arrival time of the jobs and on the y-axis is the total number of requested and busy nodes. Note that the system load is presented until the last job in the workload arrives to the scheduler queue and not until the end of the simulation.

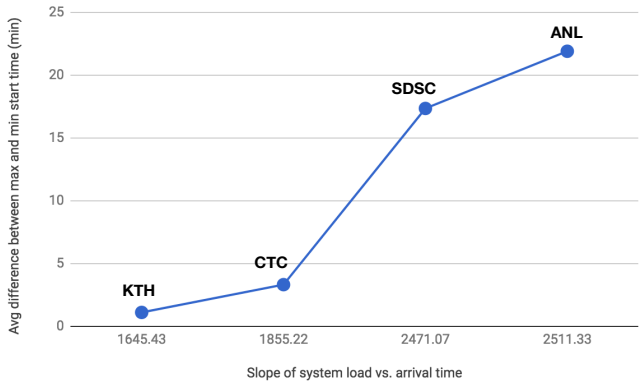


Fig. 7. Correlation between the average slope of system load vs. arrival time, derived from Figure 6 and the average difference between maximum and minimum start time across 10 runs of *SIM_SCSF*, derived from Figure 5.

simulated time and the simulation execution time. Figure 9a and Figure 9b show the execution time and the speedup, of each of the simulator's versions for four different big CIRNE logs, respectively. Our version *SIM_V14* is 2.3 to 2.6 times faster than the *SIM_SCSF* version depending on the workload. The speed of current version of the simulator is rather dependent on the total simulated time since *sim_mgr* increments the time one second each iteration and even with a single job simulator will take significant time to execute, i.e.,

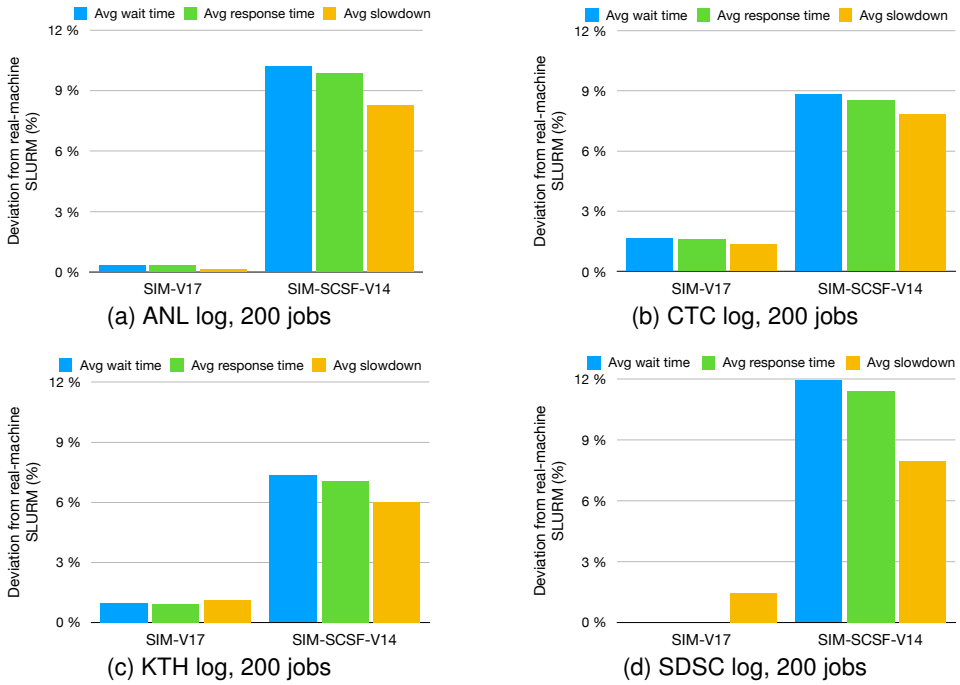


Fig. 8. Deviation of different system performance metrics obtained on SLURM simulator from their respective values obtained on the real-machine SLURM. Two versions of the simulator evaluated, *SIM_V17* and *SIM_SCSF* (x-axis), percent of deviation w.r.t. real machine results for each of the system performance metrics: average wait time, average response time and average slowdown (y-axis). Each graph shows the workload for one of the arrival patterns, ANL, CTC, KTH and SDSC. Small logs, in the case of the simulations experiments, and equivalent small real-application workloads, in the case of real-machine experiments, each of 200 jobs are used on the system of 10 nodes, as explained in Section III-A2 and III-C, and Figure 3 and 4. Note, y-axes are at the same scale.

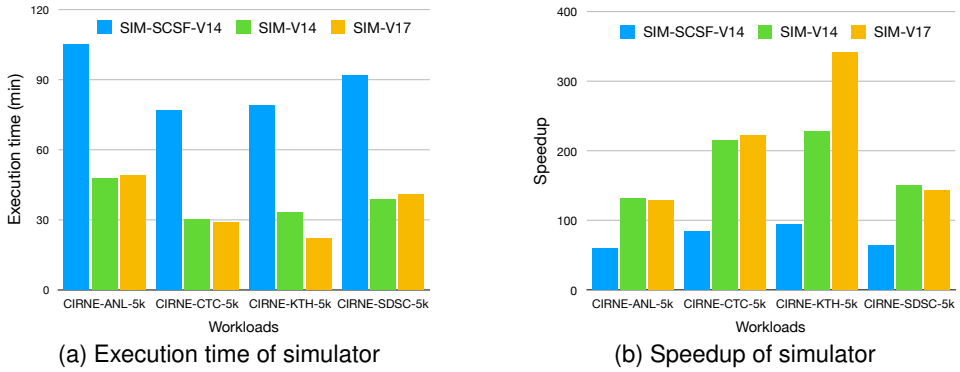


Fig. 9. Comparison of the performance among simulators versions. (a) Execution time, (b) Speedup, i.e., simulated time (see Table I) divided with execution time. Four different big logs of 5000 jobs on the system of 3456 nodes (x-axis). Three different versions of the simulators, *SIM_V17*, *SIM_V14*, *SIM_SCSF*. The version *SIM_V14* is necessary here for comparison purpose, since *SIM_SCSF* is on the SLURM version 14.02. All the simulators are executed in the same environment on the same machine.

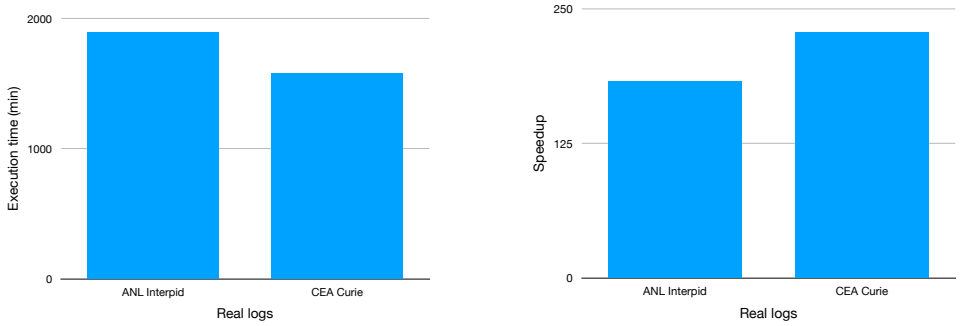
more than necessary. We believe there is a room for significant additional improvement in the simulator's speed by allowing *sim_mgr* to change the number of seconds incremented in each iteration. However, this additional improvement is planned for future work. We include the results for two real logs from the production machines regular executions of over eight months period. Two logs, ANL Interpid and CEA Curie from Feitelson's repository [7], contain much higher number of jobs, 68936 and 198509, respectively. Also, they are simulated on their original system sizes of 40960 and 5040 nodes. We show that these big traces can be executed in order of a day and speedup is comparable to the one we reported for 5000-jobs logs.

IV. SLURM Simulator Use Cases

We present several use cases that show the value of the simulator for various parametric studies.

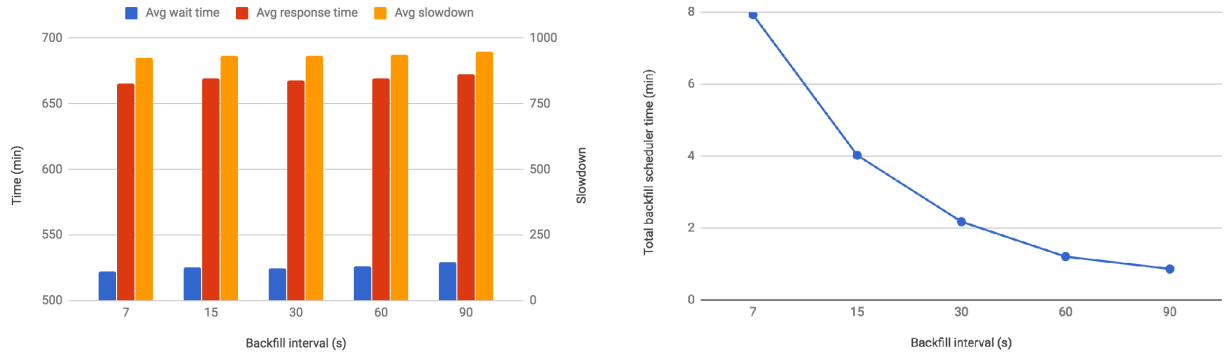
A. Use case 1: Impact of backfill interval on scheduler performance

The configuration of job scheduler parameters, such as backfill interval, may have an important impact on the system performance and scheduling time. Figure 11a shows the system performance metrics for various backfill interval values. We can see that lowering backfill interval below 30 s may bring less than 1% of improvement. However, as shown in Figure 11b this can cause a significant and unnecessary increase



(a) Execution time of simulator (b) Speedup of simulator

Fig. 10. Performance of the *SIM_V17* simulator for big real logs as an input. (a) Execution time, (b) Speedup, i.e., simulated time divided with execution time. Two different real big logs ANL Interpid and CEA Curie from Feitelson's repository [7] (x-axis). ANL Interpid log contains 68936 jobs and it is executed on the original system size of 40960 quad-core nodes. CEA Curie log contains 198509 jobs from the partition of 5040 nodes, each with 2 sockets and 8 CPUs per socket.



(a) System performance metrics (b) Backfill scheduler execution time

Fig. 11. Impact of backfill interval value on (a) system performance metrics and (b) backfill scheduler time. Backfill intervals from 7 to 90 s (x-axis). (a) average wait time and average response time (left y-axis), slowdown (right y-axis). (b) Total time spent in executing backfill scheduler during the entire simulation (y-axis). Big log of 5000 jobs with ANL arrival pattern on the system of 3456 nodes simulated on *SIM_V17* simulator.

in total job scheduling time. Thus, this simple experiment on the simulator may allow system administrators to choose good enough backfill interval for their system.

B. Use case 2: Impact of job queue length on scheduler performance

Similarly, a system administrator or a researcher can think of implementing new job scheduler parameters and testing the system and scheduler performance for different values of these parameters. As an example, we implemented backfill queue limit, a parameter that limits the number of jobs in the queue tested by backfill scheduler. Analysis of the impact of different values of this parameter on system performance and backfill total time is given in Figure 12a and Figure 12b, respectively.

C. Use case 3: Impact of system size on system performance metrics

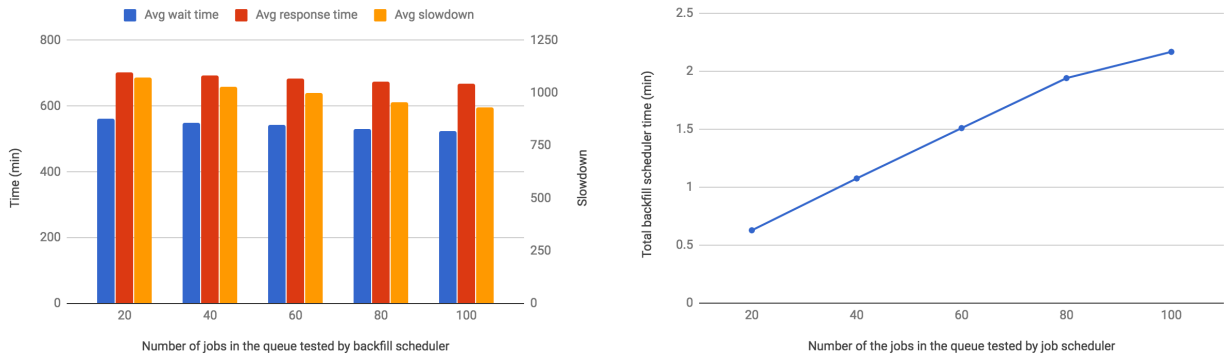
This set of experiments aims to show how a potential increase or decrease in system size may impact system performance metrics. Running the typical system's workload in the simulator may help the system administrators evaluate the overall cost of the system size change. In our example experiment in Figure 13 decrease of the system size by 12,5% or 25% degrades the system performance metrics by around 40-60% and 100-140%, respectively. On the other side, an increase of the system size by 12,5% or 25% improves the system performance metrics by around 30-40% and 50-70%, respectively.

D. Use case 4: Evaluation of the job scheduler scalability

The simulator may help to evaluate the scalability of the SLURM scheduler itself. Namely, we estimate the time spent in backfill scheduling for different system loads. We achieve different system loads by running the same workload on the different system sizes. When system size decreases, the system becomes more loaded and vice versa. Figure 14 shows that the total time spent in backfill scheduler for a workload of more than four days is around 2 minutes, even with the extreme case of the system being twice smaller, the total backfill time reaches 5.5 minutes. This simple example proves SLURM to be rather scalable, but also, it can be used by job scheduling researchers and SLURM developers to estimate the scalability of the new scheduling algorithm implementations.

V. Related work

Job scheduling evaluation is an important topic since small changes in the scheduling and resource management can significantly affect system performance. There are different



(a) System performance metrics

(b) Backfill scheduler execution time

Fig. 12. Impact of backfill queue size on (a) system performance metrics and (b) backfill scheduler time. Number of jobs in the queue checked by backfill scheduler from 20 to 100 (x-axis). (a) average wait time and average response time (left y-axis), slowdown (right y-axis). (b) Total time spent in executing backfill scheduler during the entire simulation. Big log of 5000 jobs with ANL arrival pattern on the system of 3456 nodes simulated on *SIM_V17* simulator.

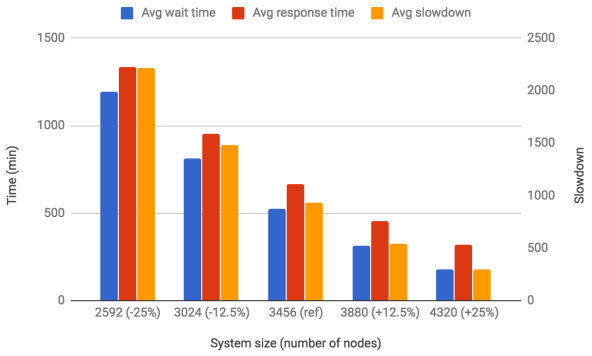


Fig. 13. Impact of system system size on system performance metrics. System size in number of nodes (x-axis). The middle point is the reference system, two points on the left are system sizes reduced by 12,5% and 25% w.r.t. reference system, respectively, two points on the right are system sizes increased by 12,5% and 25% w.r.t. reference system, respectively. Average wait time and average response time (left y-axis), slowdown (right y-axis). Big log of 5000 jobs with ANL arrival pattern on the system of 3456 nodes simulated on *SIM_V17* simulator.

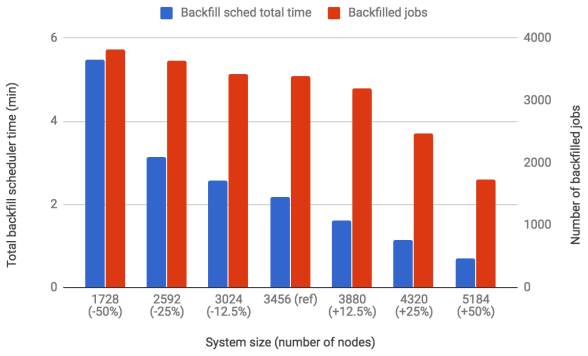


Fig. 14. Impact of system load on the backfill scheduler scalability and number of backfilled jobs. System size in the number of nodes (x-axis). The middle point is the reference system, three points on the left are system sizes reduced by 12,5%, 25%, 50% w.r.t. reference system, respectively, three points on the right are system sizes increased by 12,5%, 25%, 50% w.r.t. reference system, respectively. Total time spent in executing backfill scheduler during the entire simulation (left y-axis). The total number of jobs being scheduled by backfill scheduler (right y-axis). Big log of 5000 jobs with ANL arrival pattern on the system of 3456 nodes simulated on the *SIM_V17* simulator.

methodologies for evaluating the efficiency and effectiveness of a job scheduler which we can separate in *benchmarks* and *simulations*. Benchmarks assume a real run of workloads in a cluster, with the purpose of evaluating well-known system metrics [10] or specific aspects of the system that administrator needs to optimize [11], such as the effect of dynamic job scheduling in the context of malleable jobs. However, it is not always possible to stop a production-machine to perform this type of evaluation, so usually, simulations are more convenient and practical to perform.

We can further divide job scheduling simulators into general job scheduling simulators and implementation-specific job scheduling simulators simulators.

There are plenty of traditional job scheduler simulators [12] [13] [14] [15]. These simulators are suitable for a theoretical evaluation of scheduling algorithms, and they usually include configurations for modeling the different platforms, partitions, hardware, energy, and networks. General job scheduler simulators lack enough details, characterization of parameters included in production software and the software architecture that a system administrator might want to optimize to get the most out of a particular machine. Batsim presents some accuracy tests, done after developing an adaptor between Batsim and OAR [16] to allow using OAR schedulers into the simulator, and a submission system that reads and sends 800-jobs requests to OAR operating on 161 nodes. This methodology is hard to reproduce since few workload managers permit decoupling the scheduler code from the rest, and simulation is limited to the scheduler itself, not the whole software infrastructure, including other code parts and implementation related parameters. Also, Simbatch presents accuracy tests, by implementing models for simulating batch schedulers, and running 100 tasks on 5 nodes tests with OAR. In this case, it is not clear up to which detail the models are representing real job schedulers. In conclusion, standard job scheduling

simulators are a good start for the evaluation of a scheduling algorithm, but they give only little hints about how they will perform on a real machine, not representing an extensive set of tools for system administrators.

On the other side, implementation-specific simulators keep all the details of a specific job scheduler, maintain their architecture, reusing their source code, and giving the possibility to system administrators to try different configurations and algorithms with the objective of tuning system performance. To the best of our knowledge, we found three simulators in this category; the first is Qsim[17], an event-based simulator for Cobalt[18], specific job scheduler for Blue Gene systems. A second example is Moab[19] scheduler, that implements a simulator mode, in which the user can interact and control simulated time, but it is a proprietary software. Flux [20] is a Resource Management framework that includes a simulator in its code, but either publications and documentation lack of information about it. There is no published evaluation of the consistency and the accuracy for any of these simulators. The last one is SLURM simulator that, before our work, experienced several previous improvements. The first version was based on SLURM version 2, developed at Barcelona Supercomputing Center by A. Lucero [2]. In the second one, Trofinoff and Benini [6] from CSCS updated the simulator to SLURM version 14 and brought a series of improvements over it. Our work is based on the top of G. Rodrigo [3] effort, which improved the synchronization and the simulator speedup, together with a set of tools for workload generation, scheduler configuration and output analysis. None of these simulators satisfied our needs regarding accuracy and consistency.

Finally, Simakov et al. [21] attempted to validate their own SLURM simulator version. Simakov heavily simplified simulator structure, serializing the code on a single process, the SLURM controller, that can be compiled as a simulator. While he significantly reduced the amount of executed code and complexity, he lost some of the features that SLURM can offer, e.g., plugins that are used inside SLURM node daemons and SLURM original architecture. Moreover, the paper is poor in the validation part, in which SLURM code is not validated with real-machine runs, but it uses a SLURM compiled in front-end mode. This mode is typically used by SLURM developers for testing, and debugging purposes and it is the base mechanism used by all SLURM simulators. It allows simulating multiple nodes by routing all messages to the same *slurmd*, that acts as a front-end. This methodology alters the SLURM architecture and communications, and it overloads a single daemon that is in charge of simulating a high number of nodes, affecting the final results. On our side, we validated the simulator in a real machine, by creating and running real workloads into a supercomputer. In our validation, we reported more accuracy, in both real runs and simulator run. Our simulator runs are completely deterministic, while it is not clear where the variability in Simakov's simulator comes from, as controlling simulated time and simplifying the SLURM code remove all the sources of outliers, and the real-machine variability model was not reported as the work done in this version. Regarding the simulator code, we reported better speedup while keeping standard SLURM architecture, that opens more possibilities for system administrators.

VI. Conclusions and Future work

In this paper, we presented our latest improvements of SLURM simulator, together with the methodology and the results of the first-ever validation of the simulator on the real machine. The validation experiments show improvements in the performance of 2.6 times comparing to the previous version, deterministic results from the multiple same-input executions of the simulator, and accuracy at the level of the real-machine, with at most 1.7% of deviation. Our effort also includes porting of the simulator to the latest version of SLURM, 17.11. Besides, we present a selected number of use cases to illustrate the usefulness of the simulator for SLURM administrators, researchers, and developers. The parametric studies of the impact of backfill interval and system size on system performance might be a use case for the administrators. Adding new parameters and testing their influence on the system performance or studying the execution time of existing and new job scheduling algorithm's as a function of system load might be a use case for the researchers and developers.

Our next efforts will go in two main directions: introducing new models into the simulator and enabling support for heterogeneous jobs.

Since the variability encountered in the previous versions of the simulator was due to poor implementation of the synchronization, it cannot be justified by the variability of a real-machine scheduler. Modeling this variability and introducing a parametrizable real-machine variability model in the simulator will be one of our future efforts.

Currently, the simulator receives as an input a job duration that is fixed during the entire simulation. Implementing a performance model that will enable changing a job's execution time depending on the architecture or other factors, such as sharing resources, is an important task for the future. Similarly, we plan to integrate into the simulator energy models.

Since the simulator is ported to the SLURM 17.11 that provides support for heterogeneous jobs, we plan to make adaptations in the simulator internals and simulator's inputs to accept and process heterogeneous jobs' requests.

Acknowledgment

This work is partially supported by the Spanish Government through Programa Severo Ochoa (SEV-2015-0493), by the Spanish Ministry of Science and Technology through TIN2015-65316-P project, by the Generalitat de Catalunya (contract 2017-SGR-1414) and from the European Commission's Horizon 2020 Programme for research, technological development and demonstration under Grant Agreement No 754304.

The authors would like to thank previous contributors, Alejandro Lucero, Massimo Benini and Gonzalo Rodrigo, for their work and their timely response to our questions.

REFERENCES

[1] M. A. Jette, A. B. You, and M. Grondona, "Slurm: Simple linux utility for resource management," in *Proceedings of the 9th International Workshop Job Scheduling Strategies for Parallel Processing (JSSPP)*. Springer, Lecture Notes in Computer Science (LNCS), volume 2862, 2003, pp. 44–60.

[2] A. Lucero, "Simulation of batch scheduling using real production-ready software tools," in *Proceedings of the 5th IBERGRID*, 2011.

[3] G. P. Rodrigo, E. Elmroth, P.-O. Ostberg, and L. Ramakrishnan, "Scsf: A scheduling simulation framework," in *Proceedings of the 21st International Workshop Job Scheduling Strategies for Parallel Processing (JSSPP)*, 2017.

[4] S. J. Chapin, W. Cirne, D. G. Feitelson, J. P. Jones, S. T. Leutenegger, U. Schwiegelshohn, W. Smith, and D. Talby, "Benchmarks and standards for the evaluation of parallel job schedulers," in *Proceedings of the 13th International Workshop Job Scheduling Strategies for Parallel Processing (JSSPP)*. Springer-Verlag, 1999, pp. 66–89.

[5] *The Standard Workload Format*. [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload/

[6] S. Trofinoff and M. Benini, *Using and Modifying the BSC Slurm Workload Simulator*, Slurm User Group Meeting 2015. [Online]. Available: https://slurm.schedmd.com/SLUG15/

[7] *Logs of Real Parallel Workloads from Production Systems*. [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload/

[8] *BSC Slurm Simulator*. [Online]. Available: https://github.com/BSC-RM/slurm_simulator/

[9] W. Cirne and F. Berman, "A comprehensive model of the supercomputer workload," in *Proceedings of the 4th Annual Workshop on Workload Characterization*, 2001.

[10] A. T. Wong, L. Oliker, W. T. Kramer, T. L. Kaltz, and D. H. Bailey, "ESP: A system utilization benchmark," in *Supercomputing, ACM/IEEE 2000 Conference*. IEEE, 2000, pp. 15–15.

[11] V. Lopez, A. Jokanovic, M. DAmico, M. Garcia, R. Sirvent, and J. Corbalan, "Djsb: Dynamic job scheduling benchmark," in *Job Scheduling Strategies for Parallel Processing: 21st International Workshop, JSSPP 2017, Orlando, FL, USA, June 2, 2017, Revised Selected Papers*. Springer, 2017.

[12] P.-F. Dutot, M. Mercier, M. Poquet, and O. Richard, "Batsim: a Realistic Language-Independent Resources and Jobs Management Systems Simulator," in *20th Workshop on Job Scheduling Strategies for Parallel Processing*, Chicago, United States, May 2016. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01333471

[13] D. Klusáček and H. Rudová, "Alea 2: Job scheduling simulator," in *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, ser. SIMUTools '10. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2010, pp. 61:1–61:10. [Online]. Available: https://doi.org/10.4108/ICST.SIMUTOOLS2010.8722

[14] Y. Caniou and J. S. Gay, "Simbatch: An api for simulating and predicting the performance of parallel resources managed by batch systems," in *Euro-Par 2008 Workshops - Parallel Processing*, E. César, M. Alexander, A. Streit, J. L. Träff, C. Cérin, A. Knüpfer, D. Kranzlmüller, and S. Jha, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 223–234.

[15] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2899–2917, Jun. 2014. [Online]. Available: http://hal.inria.fr/hal-01017319

[16] "Oar resource manager." [Online]. Available: http://oar.imag.fr/documentation

[17] W. Tang, Z. Lan, N. Desai, and D. Buettner, "Fault-aware, utility-based job scheduling on blue, gene/p systems," in *2009 IEEE International Conference on Cluster Computing and Workshops*, Aug 2009, pp. 1–10.

[18] *Cobalt website*. [Online]. Available: https://www.alcf.anl.gov/cobalt-scheduler

[19] *Maui simulator*. [Online]. Available: http://docs.adaptivecomputing.com/maui/16.0simulations.php

[20] D. H. Ahn, J. Garlick, M. Grondona, D. Lipari, B. Springmeyer, and M. Schulz, "Flux: A next-generation resource management framework for large hpc centers," in *2014 43rd International Conference on Parallel Processing Workshops*, Sept 2014, pp. 9–17.

[21] N. A. Simakov, M. D. Innus, M. D. Jones, R. L. DeLeon, J. P. White, S. M. Gallo, A. K. Patra, and T. R. Furlani, "A slurm simulator: Implementation and parametric analysis," in *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*, S. Jarvis, S. Wright, and S. Hammond, Eds. Cham: Springer International Publishing, 2018, pp. 197–217.