Alea – Complex Job Scheduling Simulator

Dalibor Klusáček[1(✉)], Mehmet Soysal[2], and Frédéric Suter[3]

[1] CESNET a.l.e., Brno, Czech Republic
klusacek@cesnet.cz
[2] Steinbuch Centre for Computing, Karlsruhe Institute of Technology, Karlsruhe, Germany
mehmet.soysal@kit.edu
[3] IN2P3 Computing Center/CNRS, Lyon-Villeurbanne, France
frederic.suter@cc.in2p3.fr

Abstract. Using large computer systems such as HPC clusters up to their full potential can be hard. Many problems and inefficiencies relate to the interactions of user workloads and system-level policies. These policies enable various setup choices of the resource management system (RMS) as well as the applied scheduling policy. While expert's assessment and well known best practices do their job when tuning the performance, there is usually plenty of room for further improvements, e.g., by considering more efficient system setups or even radically new scheduling policies. For such potentially damaging modifications it is very suitable to use some form of a simulator first, which allows for repeated evaluations of various setups in a fully controlled manner. This paper presents the latest improvements and advanced simulation capabilities of the *Alea* job scheduling simulator that has been actively developed for over 10 years now. We present both recently added advanced simulation capabilities as well as a set of real-life based case studies where Alea has been used to evaluate major modifications of real HPC and HTC systems.

Keywords: Alea · Simulation · Scheduling · HPC · HTC

1 Introduction

The actual performance of a real RMS depends on many variables that include the type (mix) of users' workloads (e.g., parallel vs. sequential jobs, short vs. long jobs), applied job scheduler and its scheduling algorithm (e.g., trivial First Come First Served (FCFS) or backfilling [14]) and also additional system configuration that typically defines job mapping to queues and their priorities and various operational limits (e.g., max. number of CPUs available to a given user). Therefore, designing a proper configuration is the most important, yet truly daunting process. Due to the complexity of the whole system even straightforward (conservative) changes in the configuration of the production system can have highly unexpected and often counterintuitive side effects that emerge from

R. Wyrzykowski et al. (Eds.): PPAM 2019, LNCS 12044, pp. 217–229, 2020.
https://doi.org/10.1007/978-3-030-43222-5_19

the mutual interplay of various policies and components of the RMS and scheduler [10]. Therefore, simulators that can emulate a particular production system and its configuration represent highly useful tools for both resource owners, system administrators and researchers in general.

Alea jobs scheduling simulator has been first introduced in 2007 as a basic simulator and underwent a major upgrade in 2010 [9] that mainly focused on improving the rather slow simulation speed and also introduced some visualization capabilities. In 2016, Alea was the first mainstream open source simulator to enable the use of so called *dynamically adapted workloads*, where the performance of the simulated scheduler directly influences the submission rates (arrival times) of jobs from the workload [22], providing an important step to mimic the natural user feedback to the system performance [12].

Since then, many new features have been implemented and the simulator has been successfully used for various purposes, both as a purely research tool as well as when testing new setups and new scheduling policies for production HPC and HTC systems. The main contribution of this paper is that (1) we describe recent improvements in the simulator, that allow for truly complex simulations that involve several detailed setups that correspond to typical real-life based scenarios, (2) we describe the recent speedup of the simulator that enables us to run truly large-scale simulations involving millions of jobs and thousands of nodes that complete in just a few hours, (3) we compare the performance of Alea with existing simulators, and (4) we provide several real-life based case studies where Alea has been used to develop and evaluate effects of major modifications of real HPC and HTC systems.

In Sect. 2 we provide a brief overview of existing related work. Next, Sect. 3 shows the current design of Alea and its major features and simulation capabilities. Section 4 presents several real-life examples demonstrating how Alea has been used in practice in order to improve the performance of production systems. Finally, we conclude the paper in Sect. 5.

2 Related Work

Throughout the years, there have been many grid, HPC and cloud simulators. In most cases, each such simulator falls into one of three main groups. The first group represents ad hoc simulators that are built from scratch. Those include, e.g., the recent AccaSim or Qsim. AccaSim is freely available library for Python, thus compatible with any major operating system, and executable on a wide range of computers thanks to its lightweight installation and light memory footprint [5]. Qsim is an event-driven scheduling simulator for Cobalt, which is an HPC job management suite supporting compute clusters of the IBM BlueGene series [21]. It is using exactly the same scheduling and job allocation schemes used (or proposed) for Cobalt and replays the job scheduling behavior using historic workloads analyzing how a new scheduling policy can affect system performance. Still, both simulators are somehow limited. Qsim is aiming

primarily on BlueGene-like architectures, while AccaSim's capabilities (e.g., supported scheduling policies) are still rather limited as of late 2019[1].

Second group of simulators is typically using some underlying simulation toolkit, e.g., SimGrid, GridSim or CloudSim. This group is represented, e.g., by the recent Batsim, Simbatch or GSSIM [3]. Batsim is built on top of SimGrid [4]. It is made such that any event-based scheduling algorithm can be plugged to it and tested. Thus, it allows to compare various scheduling algorithms from different domains. Such schedulers must follow a text-based protocol to communicate with Batsim properly. In the paper on Batsim [4], this is demonstrated by using OAR resource manager's scheduler with the Batsim simulator [4]. This is certainly a very interesting feature adding to the realism of the simulations. Still, it is not very straightforward to use existing schedulers in this way as they are typically tightly coupled with the remaining parts of a given resource manager and cannot be easily used in a standalone fashion. Simbatch and GSSIM [3] were using SimGrid and Gridsim respectively, but their development is currently discontinued for many years.

Finally, the last group typically uses some real-life RMS executed in a simulation mode. For example, the ScSF simulator [16] emulates a real system by using Slurm Workload Manager inside its core to realistically mimic the real RMS. ScSF extends an existing Slurm Simulator [18], improving its internal synchronization to speed up its execution. Also, it adds the capability to generate synthetic workloads. Similar "simulation mode" was supported in Moab in the past[2] but has been discontinued in the recent versions. In all cases, simulators using a real RMS cannot process workload as quickly as the simulators from the first two groups. This is caused by the fact that these simulators must follow the complex timing model of a real RMS (see Sect. 3.4).

Alea simulator, which will be thoroughly described in the next section, represents the second group of simulators using an underlying simulation toolkit. The major weakness of Alea is that it cannot use an existing scheduler and/or RMS. Instead, the RMS/scheduler must be simulated using Alea and GridSim. While this fact can be somehow limiting in certain cases, Alea offers a large set of features that mimic the functionality of real schedulers (see Sect. 3). At the same time, it allows to simulate large workloads and big systems in a very competitive time (see Sect. 3.4) while remaining fully deterministic. This is not the case for simulators using real RMS that are subject to varying "system jitter" from the used RMS [18]. The aforementioned list of existing simulators is not exhaustive and more details can be found in [5,9].

3 Architecture and Major Functionality

Alea is platform-independent event-driven discrete time simulator written in Java. It is built on the top of the GridSim simulation toolkit [20]. GridSim provides the basic functionality to model various entities in a simulated computing

[1] https://accasim.readthedocs.io/.
[2] http://docs.adaptivecomputing.com/mwm/archive/6-0/2.5initialtesting.php.

system, as well as methods to handle the simulation events. The behavior of the simulator is driven by an event-passing protocol. For each simulated event—such as job arrival or completion—one message defining this event is created. It contains the identifier of the message recipient, the type of the event, the time when the event will occur and the message data. Alea extends this basic GridSim's functionality and provides a model allowing for detailed simulation of the whole scheduling process in a typical HPC/HTC system. To do that, Alea either extends existing GridSim classes (e.g., `GridResource` or `AllocationPolicy`) or it provides new classes on its own, especially the core `Scheduler` class and classes responsible for data parsing and collection/visualization of simulation results. Figure 1 shows the overall scheme of Alea simulator, where boxes denote major functional parts and arrows express communication and/or data exchange within the simulator. The blue color denotes recently added or heavily upgraded components of the simulator.

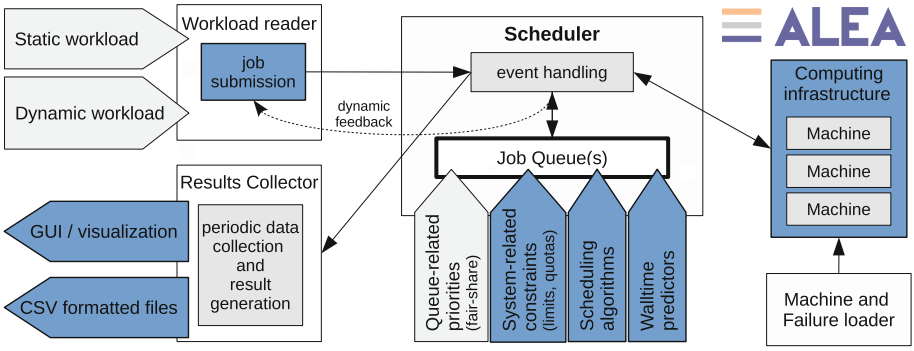

Fig. 1. Main parts of the Alea simulator (blue color denotes new functionality). (Color figure online)

The main part of the simulator is the centralized job scheduler. The scheduler manages the communication with other parts of the simulator. Also, it maintains important data structures such as job queue(s). Job scheduling decisions are performed by scheduling algorithms that can be easily added using existing simple interfaces. Furthermore, scheduling process can be further influenced by using additional system policies, e.g., the fair-sharing policy which dynamically prioritizes job queue(s). Also, various limits that further refine how various job classes are handled are supported by Alea. Additional parts simulate the actual computing infrastructure, including the emulation of machine failures/restarts. Workload readers are used to feed the simulator with input data about jobs being executed and the simulator also provides means for visualization and generation of simulation outputs. Alea is freely available at GitHub [1].

The primary goal of our job scheduling simulator is to allow for realistic evaluation of new scheduling policies or new setups of computing systems. For

this purpose, it is necessary to model all important features that have significant impact on the performance of the system. Our own "hands on" experience from operating production systems have taught us that many promising "theoretical" works based on simulations are not usable in practice, due to the overly simplified nature of performed simulations. Often, researchers focus solely on particular scheduling algorithm while ignoring additional system-related constraints and policies. However, production systems use literally dozens of additional parameters, rules and policies that significantly influence the scheduler's decisions and thus the performance of the system [10,17]. Therefore, following subsections provide an overview of the advanced simulation capabilities that make Alea a very useful tool for detailed simulations of actual systems.

3.1 Detailed System Simulation Capabilities

As we have observed in practice, system performance can be largely affected by the interactions of various components and parameters of an actual RMS. While their nature or scope can be basic and limited, they can easily turn a well functioning system into a troublesome one. Therefore, the simulator should be able to mimic these features within the simulation. These features include:

- queues and their priorities, constraints and various limits
- quotas limiting user CPU usage
- mechanisms to calculate job priorities such as fair-share
- common scheduling algorithms aware of aforementioned features

Queues, Limits and Quotas. First of all, Alea allows to specify the number of job queues, their names, priorities and queue-related constraints such as the maximum number of CPUs that can be used by jobs from that queue at any given moment. Multiple queues are common in systems with heterogeneous workloads. Here, system resources are usually partitioned into several (sometimes overlapping) pools, where each pool has a corresponding queue. Users' workloads (jobs) are then mapped to these queues. Queue limits then avoid potentially dangerous situations such as saturation of the whole system—either with jobs from a single user, or with a single class of jobs [7]. For example, it would be very unwise to fill the whole system with long running jobs as this would cause huge wait times for shorter jobs. Also users and/or groups are often subject to a upper bound on the amount of resources they can use simultaneously. For this purpose, Alea now provides CPU quotas, that guarantee that a user/group will not exceed the corresponding maximum allowed share of resources [2].

Fair-Sharing. Production systems—instead of default job arrival order—often use some priority mechanism to dynamically prioritize system users. This is typically done by fair-sharing. We provide several variants of *fair-sharing mechanisms* that are used to prioritize jobs (users) within queue(s) in order to guarantee user-to-user fairness. Fair-share mechanism dynamically adjusts job/user

priorities such that the use of system resources is fairly balanced among the users [7]. We support both basic fair-sharing mechanisms that only reflect CPU usage as well as more complex multi-resource implementations[3] which also reflect memory consumption.

Scheduling Algorithms. Scheduling algorithms play a critical role in RMS. Alea supports all mainstream algorithms that can be typically observed in practice, starting with trivial FCFS, Shortest Job First and Earliest Deadline First and continuing to more efficient solutions such as EASY backfilling or Conservative backfilling [14]. Alea also supports *schedule optimization methods*, that can be used to further improve initial job schedules as prepared by, e.g., the Conservative backfilling policy. Our optimization methods are based on metaheuristics and can use various objective functions to guide the metaheuristic toward improved schedule [8]. Importantly, in the recent release we provide several job walltime predictors, that can automatically refine (inaccurate) user-provided walltime estimates in order to improve the precision of scheduling decisions.

3.2 Dynamic Workloads

There is one more part which plays a significant role in job scheduling—the workload being processed by the system or the simulator. Alea supports two ways how workload can be fed into the simulator. First, it uses traditional "workload replay" mode, where jobs are submitted based on a historic workload description file (log) and their arrival times are based on the original timestamps as recorded in the log. Alternatively, Alea allows to use so called dynamic workload adaptation, where job arrival times are not fixed but can change throughout the course of the simulation, depending on the scheduler's performance. For this purpose, Alea provides a feedback loop that communicates with the workload reader and informs it upon each job completion. Using this data, the workload reader can either speed up or postpone job submissions for simulated users. This complex behavior mimics real world experience, where users react to the performance of the scheduler. In other words, real-life job arrival times are always correlated to the "user experience", thus it is unrealistic to use plain "workload replay" mode, because the results will be somehow skewed by the "embedded" influence of the original scheduler that is captured in the historic workload log, i.e., in the job arrival time pattern. Alea's implementation is based on the work of Zakay and Feitelson [22], but it also allows to write your own workload adaptation engine, having different job submission adaptation logic.

3.3 Simulation Speed

Since the start of Alea project, simulation speed was our second most important goal right after the capabilities of our simulator. During the years, Alea

[3] For example, we support Dominant Resource Fairness (DRF) inspired fair-share [6].

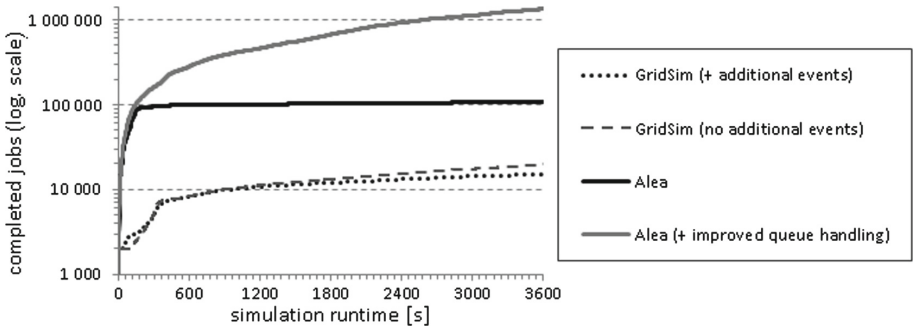

Fig. 2. Number of completed jobs (log. scale) during 1 hour-long simulation using different implementations of `SpaceShared` policy and (un)optimized queue handling.

has introduced several improvements into the GridSim's event-driven simulation model that significantly speed up the simulation. Most changes relate to the way job execution is modeled in the classes that implement job allocation policy on a modeled physical system (see, e.g., GridSim's `SpaceShared` class). As originally designed, this model was not very time-efficient. Upon each start of a job j, an internal event was generated that was scheduled to be delivered at the time $T_{compl}(j)$, which is the time when such job would complete[4]. Although this event at $T_{compl}(j)$ only corresponds to that job j, GridSim would always scan all currently executed jobs to check whether those are completed or not. Obviously, this was not very time efficient way how to proceed with a simulation. Moreover, with each such check GridSim would also generate one additional internal event to trigger a similar check (delayed by a predefined time constant) to further assure that no jobs are "forgotten" by the engine. However, this additional event generator was producing exponential-like increase of events that the GridSim core had to handle, slowing down the simulation extremely. While these inefficiencies are tolerable when dealing with small systems (hundreds of CPUs and few thousands of jobs), they became a real show-stopper for large simulations involving tens of thousands of CPUs and millions of jobs.

Therefore, in this new edition of Alea we have simplified the whole job processing model such that each job now only needs one internal event to be processed correctly. This did not change the behavior of the `SpaceShared` policy, but it introduced a huge speedup of the whole simulator. Also, we have improved the speed of scheduling algorithms. Simulations that struggle with large job queue sizes (plenty of waiting jobs) are often slowed down by the scheduling algorithm which repeatedly traverses long job queues, trying to schedule a new job. With long queues, this may slow down the simulation significantly, especially when the algorithm itself is not a trivial one. Therefore, we have introduced a more efficient *queue handling mechanism* which—based on user specified parameters—limits

[4] $T_{compl}(j) = T_{current} + T_{runtime}(j)$.

the number of jobs that are checked at each scheduling run. This modification brought another huge improvement.

Figure 2 shows an example of the speedup obtained by our techniques. It shows the number of completed jobs (in log. scale) that were simulated during one hour. This experiment involved large system with over 33K CPU cores and many peaks in the job queue that reached up to 5 K of waiting jobs. The results of our optimized event-processing mechanism and the queue handling mechanism are compared to the original GridSim's implementation, with the "exponential" event generator either turned on (denoted as "Gridsim + additional events") or off ("no additional events"). Clearly, there is a huge difference when the optimized event-processing mechanism is introduced (denoted as "Alea"). Even bigger improvement is reached once the more efficient queue handling mechanism is used ("Alea + improved queue handling"). This effect is amplified by the fact that this experiment often experienced very long queue of waiting jobs.

3.4 Simulation Throughput and Speedup Comparison

To give the speed of our simulator into a context, we have studied the reported speeds of different simulators and created a simple comparison of their performance. We have used the recent published data about Slurm Simulator [18], Batsim [13] and ScSF [16]. If possible, we show both the achieved *speedup* as well as the *throughput* of the simulator. The speedup is the ratio of the original makespan[5] to the wall-clock time requested by the simulator to finish the experiment. Throughput is measured as the average number of jobs simulated (completed) in one minute. Since the speedup and throughput also depends on the "size" of the experiment [18], we report the total number of CPU cores and jobs being simulated (if available). The results are shown in Table 1 and show the impressive speed and throughput of Alea. While Batsim reports a very nice speedup, it must be noted that this result was achieved on a very small problem instance (800 jobs and 32 cores) while Alea's results were achieved in a truly large setup (2,7M jobs and 33 K cores). Further comparisons (featuring Alea, AccaSim and Batsim) can be found in the AccaSim report [5].

Table 1. Throughput and speedup of various simulators.

	Jobs	Cores	Makespan (h)	Runtime (s)	Speedup	Throughput (jobs/min)
Slurm Sim	65,000	7,912	571	15,866	130	246
ScSF	N/A	322	168	43,200	14	N/A
Batsim	800	32	4	30	400	1,600
Alea	2,669,401	33,456	744	10,800	248	14,830

[5] Makespan denotes the time needed to process the workload in a real system.

3.5 Visualization

Alea offers `Visualizator` class that provides crucial methods to display graphical outputs during a simulation. Several metrics and outputs that are generally useful, e.g., for debugging purposes are available by default, including the visualization of created job schedule and several popular objectives. An example of such graphical output is captured in Fig. 3, which shows the average system utilization, number of waiting and running jobs, average cluster utilization and the number of used, requested and available CPUs.

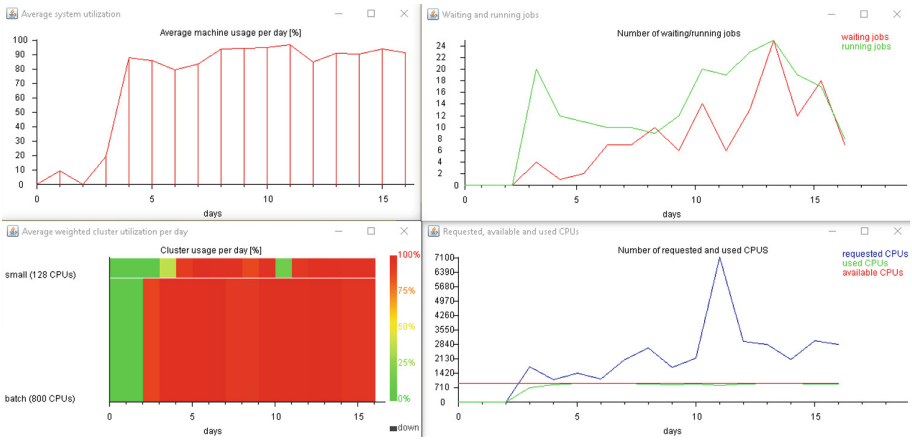

Fig. 3. Alea's visualization output showing various metrics.

Figure 4 shows the newly available visual representation of a job schedule as constructed by the scheduler. This feature is very useful especially for debugging purposes or when tuning a new algorithm. However, for larger systems the schedule cannot be reasonably displayed due to the screen resolution limitation. In this (cropped) example the vertical y-axis shows 112 CPUs of two clusters, and the x-axis denotes the planned start/completion times. The time is not to scale (linear) in order to save space. Instead, the horizontal axis uses fixed lengths between two consecutive events. An event represents either planned job start or job completion. Using this trick, the schedule can typically display rather long schedules (several days) while fitting within the limits of one screen[6].

4 Notable Usages

In this section we present four examples where Alea has been used to model an existing system and analyze the impact of new scheduling approaches. Notably, the two latter examples (Sects. 4.3 and 4.4) were achieved with the recently upgraded Alea described in this paper.

[6] In this case the schedule shows job-to-CPU mapping covering ∼3 days.

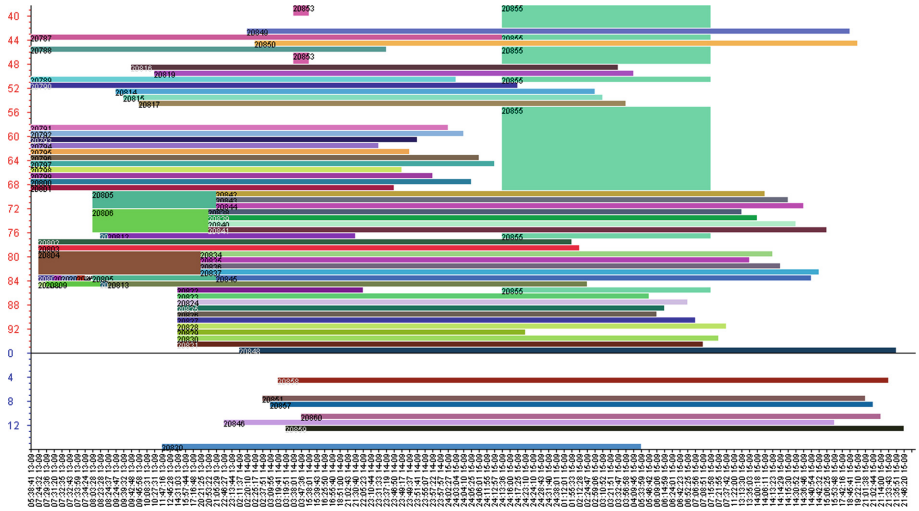

Fig. 4. Alea's new visualization feature showing constructed job schedule.

4.1 MetaCentrum Queue Reconfiguration

The first example is a major queue reconfiguration that took place in MetaCentrum, which is the largest Czech provider of distributed computing facilities for academic and scientific purposes. In this case, Alea has been used to evaluate the impact of new queue setup, where the goal has been to increase fairness, system utilization and wait times across different classes of jobs. Existing conservative setup with 3 major queues (short, normal and long jobs) and rather constraining limits concerning the maximum allowed number of simultaneously running long jobs has been replaced with an improved design introducing new, more fine-grained queues with more generous limits. The promising effect observed in the simulations was then also validated in practice. With the new setup being introduced in January 2014, the overall CPU utilization has increased by 43.2% while the average wait time has decreased by 17.9% (4.4 vs. 3.6 h) [11].

4.2 Plan-Based Scheduler with Metaheuristic Optimization

In July 2014, CERIT Scientific Cloud started to use a unique Torque-compatible scheduler that—instead of queuing—used planning and metaheuristics to build and optimize job schedules. This new planning-based scheduler has been first thoroughly modeled and refined in Alea and then remained in operation until 2017. It was a successful scheduler as it increased the avg. CPU utilization by 9.3% while decreasing the avg. wait time and the avg. bounded slowdown by 36.7% and 79.4%, respectively [8].

4.3 Scheduling with Advance Data Staging

The I/O subsystem is an increasing storage bottleneck on HPC Systems. The ADA-FS project [15] tries to close the bottleneck with deploying an on-demand file system and staging the data in advance to the allocated nodes. In a recent paper [19], Alea has been used to study the suitability of current mainstream scheduling algorithms such as FCFS and backfilling to accurately predict target nodes where a waiting job will be executed. Such a prediction is crucial when data is staged in advance or private file system is deployed prior to actual computation (while a job is still waiting). In this paper, we have demonstrated that current schedulers relying on inaccurate user-provided runtime estimates are unable to make reliable long-term predictions and even short-term predictions (less than 10 min ahead) are not possible for large fractions of jobs (∼50% of jobs).

4.4 Improving Fairness in Large HTC System

In 2019, Alea has been used to model and then reconfigure queue and quota setup in a large HTC system. This system is shared by two different workloads—a local user workload and a grid workload that comes from LHC experiments. The motivation was to increase the fairness toward local users who often have to wait much longer than those grid-originating jobs (roughly twice as long, on average). In this work, the recently improved simulation speed of Alea was mostly important, since the HTC system is rather large (33,456 cores), processing lots of jobs each month (∼2.7 millions). Using Alea, we were able to model the system and evaluate new setups for the system's queues and the per-group CPU quotas. This new setup allowed for improved fairness for local users, by better balancing their wait times with the wait times of grid-originating jobs [2].

5 Conclusion and Future Work

This paper has presented the recently upgraded complex job scheduling simulator *Alea*. We have demonstrated its capabilities and usefulness using real-life examples. Importantly, we have shown that the simulator is capable to simulate large systems and execute large workloads in an acceptable time frame. Alea can be freely obtained at GitHub [1] under the LGPL license.

Acknowledgments. We acknowledge the support and computational resources provided by the MetaCentrum under the program LM2015042, and the support provided by the project Reg. No. CZ.02.1.01/0.0/0.0/16_013/0001797 co-funded by the Ministry of Education, Youth and Sports of the Czech Republic.

References

1. Alea job scheduling simulator, April 2019. https://github.com/aleasimulator
2. Azevedo, F., Klusáček, D., Suter, F.: Improving fairness in a large scale HTC system through workload analysis and simulation. In: Yahyapour, R. (ed.) Euro-Par 2019. LNCS, vol. 11725, pp. 129–141. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29400-7_10
3. Bak, S., Krystek, M., Kurowski, K., Oleksiak, A., Piatek, W., Weglarz, J.: GSSIM - a tool for distributed computing experiments. Sci. Program. **19**(4), 231–251 (2011)
4. Dutot, P.-F., Mercier, M., Poquet, M., Richard, O.: Batsim: a realistic language-independent resources and jobs management systems simulator. In: Desai, N., Cirne, W. (eds.) JSSPP 2015-2016. LNCS, vol. 10353, pp. 178–197. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61756-5_10
5. Galleguillos, C., Kiziltan, Z., Netti, A., Soto, R.: AccaSim: a customizable workload management simulator for job dispatching research in HPC systems. arXiv e-prints arXiv:1806.06728 (2018)
6. Ghodsi, A., Zaharia, M., Hindman, B., Konwinski, A., Shenker, S., Stoica, I.: Dominant resource fairness: fair allocation of multiple resource types. In: 8th USENIX Symposium on Networked Systems Design and Implementation (2011)
7. Jackson, D., Snell, Q., Clement, M.: Core algorithms of the maui scheduler. In: Feitelson, D.G., Rudolph, L. (eds.) JSSPP 2001. LNCS, vol. 2221, pp. 87–102. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45540-X_6
8. Klusáček, D., Chlumský, V.: Planning and metaheuristic optimization in production job scheduler. In: Desai, N., Cirne, W. (eds.) JSSPP 2015-2016. LNCS, vol. 10353, pp. 198–216. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61756-5_11
9. Klusáček, D., Rudová, H.: Alea 2 - job scheduling simulator. In: 3rd International ICST Conference on Simulation Tools and Technique, ICST (2010)
10. Klusáček, D., Tóth, Š.: On interactions among scheduling policies: finding efficient queue setup using high-resolution simulations. In: Silva, F., Dutra, I., Santos Costa, V. (eds.) Euro-Par 2014. LNCS, vol. 8632, pp. 138–149. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-09873-9_12
11. Klusáček, D., Tóth, Š., Podolníková, G.: Real-life experience with major reconfiguration of job scheduling system. In: Desai, N., Cirne, W. (eds.) JSSPP 2015-2016. LNCS, vol. 10353, pp. 83–101. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61756-5_5
12. Klusáček, D., Tóth, Š., Podolníková, G.: Complex job scheduling simulations with Alea 4. In: Ninth EAI International Conference on Simulation Tools and Techniques (SimuTools 2016), pp. 124–129. ACM (2016)
13. Mercier, M.: Batsim JSSPP presentation (2016). https://gitlab.inria.fr/batsim/batsim/blob/master/publications/Batsim_JSSPP_2016.pdf
14. Mu'alem, A.W., Feitelson, D.G.: Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. IEEE Trans. Parallel Distrib. Syst. **12**(6), 529–543 (2001)
15. Oeste, S., Kluge, M., Soysal, M., Streit, A., Vef, M.-A., Brinkmann, A.: Exploring opportunities for job-temporal file systems with ADA-FS. In: 1st Joint International Workshop on Parallel Data Storage and Data Intensive Scalable Computing Systems (2016)

16. Rodrigo, G.P., Elmroth, E., Östberg, P.-O., Ramakrishnan, L.: ScSF: a scheduling simulation framework. In: Klusáček, D., Cirne, W., Desai, N. (eds.) JSSPP 2017. LNCS, vol. 10773, pp. 152–173. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-77398-8_9
17. Schwiegelshohn, U.: How to design a job scheduling algorithm. In: Cirne, W., Desai, N. (eds.) JSSPP 2014. LNCS, vol. 8828, pp. 147–167. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-15789-4_9
18. Simakov, N.A., et al.: A slurm simulator: implementation and parametric analysis. In: Jarvis, S., Wright, S., Hammond, S. (eds.) PMBS 2017. LNCS, vol. 10724, pp. 197–217. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-72971-8_10
19. Soysal, M., Berghoff, M., Klusáček, D., Streit, A.: On the quality of wall time estimates for resource allocation prediction. In: ICPP 2019 Proceedings of the 48th International Conference on Parallel Processing: Workshops. ACM (2019)
20. Sulistio, A., Cibej, U., Venugopal, S., Robic, B., Buyya, R.: A toolkit for modelling and simulating data Grids: an extension to GridSim. Concurr. Comput.: Pract. Exp. **20**(13), 1591–1609 (2008)
21. Tang, W.: Qsim (2019). https://trac.mcs.anl.gov/projects/cobalt/wiki/qsim
22. Zakay, N., Feitelson, D.G.: Preserving user behavior characteristics in trace-based simulation of parallel job scheduling. In: 22nd Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 51–60 (2014)