HPC Scheduling Simulator

Daniel Rauch

HPC Scheduling Simulator

Daniel Rauch

***Abstract*—Scheduling jobs on HPC systems can impact performance, efficiency, and utilization greatly. It can be costly to try out experimental job scheduling algorithms on an actual HPC system because of the unknown performance implications. Because of this it is important to be able to test out the implications of a job scheduling algorithm before implementing it into the system. Introduced here is an event-driven job scheduling simulator called Schedulus. It uses the Simulus Python library to implement a first come first serve job scheduling algorithm. Some comparisons are made to other job scheduling simulators.**

***Index Terms*—Event-driven simulator; Job scheduling simulator; Simulus**

I. Introduction

The motivations for this project are as follow:

- Design an Event-Driven Job Scheduling Simulator
- Implement a FCFS with EASY Backfilling job scheduling algorithm
- Utilize the Simulus discrete-event simulation library
- Analyze our simulator

The discussion will begin with an introduction into what scheduling is and why scheduling simulators are important. Discrete-event simulation is discussed and several contemporary job scheduling simulators are reviewed. After that there is a description of what the Simulus Python library is and how to use it as well as a description of what job scheduling algorithm was tested with the Schedulus job simulator. In section two some technical information is given about Schedulus and how it works. Section three will be the results followed by the conclusion in section four.

II. Background

Scheduling and simulation are proven tools for increased productivity. Scheduling is defined as, "the process of allocation of scarce resources over time" [1]. The desired outcome of scheduling is to optimize one or more objectives in a decision-making process [2]. Simulation is a tool that is used to describe a certain process in a time-dependent manner [1]. For computer systems, system resources are allocated to processes for some period of time based on the systems job scheduling policies [3]. The scheduling policies for any computer system greatly impact the performance of that system [4]. It is often impractical to test experimental scheduling policies on the hardware they are intended to run on. This is especially the case for distributed systems and supercomputers. For this reason, scheduling simulators for computer systems are invaluable tools for increasing and fine tuning the performance of a system [5], [6]. In the context of a computer system, a scheduling simulator is used to test different scheduling policies independent of the physical computer hardware. A scheduling simulator is a virtual model of a system used to test and optimize the distribution of resources and the decision making process behind those distributions in that system [7].

A. Discrete-Event Simulation

There are several different paradigms that could be applied to scheduling simulation. The discrete-event driven paradigm was used for the Schedulus simulator. A discrete-event simulator "models the operation of a system as a discrete sequence of events in time. Each event occurs at a particular instant in time and marks a change of state in the system" [8]. The state of the system is assumed unchanged between consecutive events which allows the simulation time to directly jump ahead to the next occurrence of an event, called the next- event time progression. This is important because it allows the simulation to run much faster than it would under real time constraints, continuous-time simulation, ect [8], [9].

Fig. 1: Discrete-event simulator.

B. Scheduling Simulators

The following are some examples of currently available job scheduling simulators:

- CQsim: An event driven job scheduling simulator [10]–[17]. Can take user parameters via a command line interface or can be taken via a standard workload format (SWF) file [18].
- Alea 2: A Grid and cluster event driven job scheduling simulator. Supports several common scheduling algorithms working either on the queue or the schedule based principle. Includes a visualization interface for data representation [19].
- Slurm Simulator: This simulator is actually a modification of the Slurm resource manager for HPC [20]–[22]. This simulator allows the modeling of a system using Slurm, then the ability to run workload information through that model with the option of different system settings. This case is good because it allows experimentation on the system without actually impacting performance on the actual system.
- GridSim: A Java-based discrete-event grid simulation toolkit. This toolkit supports modeling and simulation of heterogeneous grid resources, users and their application models [23].

C. Simulus

Simulus is an open-source discrete-event simulator that is both event-driven and process-oriented. It offers high-level modeling abstractions to ease simulation and synchronized groups supporting model composition and parallel simulation [24]. Simulus is a programming tool written in the Python programming language and can be easily obtained via the command line using pip. There are two primary ways to use Simulus. One way is through events. The user can schedule events and simulus makes sure all events are sorted in timestamp order. When an event occurs, Simulus advances the simulation time to the event and calls the event handler, which is a user-defined function. While an event is being processed, the user can schedule new events into the simulated future which is called "direct event scheduling" in Simulus [24].

The second way to use Simulus is through processes. The user can create and run processes that can interact with each other. Each process is a seperate thread of control. The approach of "process scheduling" is implemented during a process's execution where a process may be suspended by either sleeping for some time or becoming blocked while waiting for a currently unavailable resource. Both direct event scheduling and process scheduling are to be used in Simulus together to achieve modeling tasks.

D. First Come First Serve w/ EASY Backfilling

The first come first serve (FCFS) job scheduling algorithm is pretty self explanatory. The system runs each job in the order that they were received and if there are enough resources to do so. If there are not enough available resources then the jobs are placed in a queue where they wait their turn to run. This is a very simple algorithm to implement but can cause fragmentation [25]. An example of fragmentation would be if a large job is next up in the queue but is unable to run because there are not enough resources available for that large job. If there are enough resources available to run smaller jobs that are further down the queue then this would result in fragmentation. It has a negative impact on resource utilization and performance.

In order to avoid the fragmentation and poor utilization of FCFS one could also implement the EASY backfilling algorithm. EASY stands for the "Extensible Argonne Scheduling sYstem" [25]. This algorithm allows smaller jobs to skip ahead of the line if it is able to run given the available resources if a larger job earlier in the queue can not. There are some rules to this however. To avoid starvation of resources from larger jobs, the smaller job has to be sure not to delay the start of the larger job. The run time estimates of the jobs are known beforehand [26].

The two schenarios where EASY is applicable are represented by the two populated schedule spaces in Figure 3. Suppose the job labeled "first" in both depictions in Figure 3 is the next in like job to run given our FCFS schedule, and call that job J. The top depiction in Figure 3 shows the importance of shadow space. Shadow space is the region in the schedule space that is not currently occupied by another job and begins

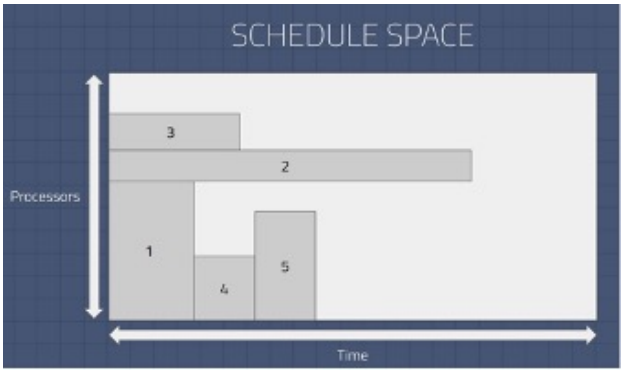

Fig. 2: Example Scheduling Space.

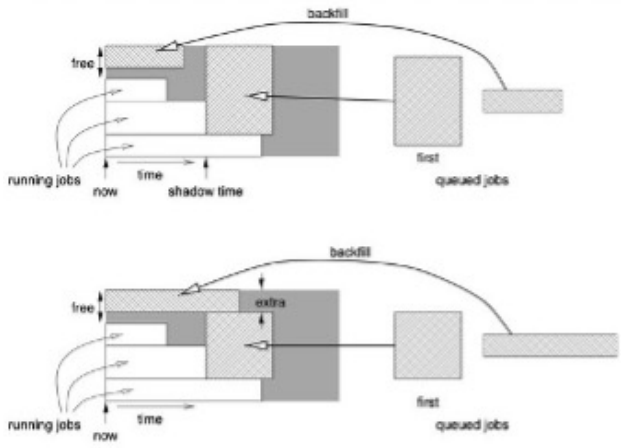

Fig. 3: EASY Backfilling [27]

now and ends at the shadow time, or the time at which job J is scheduled to run. The bottom depiction in Figure 3 represents the importance of calculating the number of extra processors available while job J is being executed. The region in schedule space including these extra processors while J is running will be called the extra processor space. It is always permissible for a scheduler operating under EASY backfilling semantics to fill the shadow space and extra processor space with jobs while backfilling. The implementation and rationalization for EASY backfilling in Schedulus was taken from [27].

III. Scheduling Conceptual Overview

Schedulus operates by maintaining a schedule space, as respresented visually by Figure 1. Schedulus does this by maintining object types representing fundamental concepts in HPC, where objects types are defined as classes in object oriented programming (OOP) [28]. Schedulus orchestrates these objects by responding to discrete-events generated by Simulus. Schedulus defines event semantics that determine how jobs are processed. At any given time during execution, Schedulus maintains a list of scheduled jobs and a list of running jobs.

A. Jobs

A job object represents a computational activity that has not yet been run, is running, or has already been run by the scheduler. It stores data as specified by the Parallel Workloads Archive in the standard workload format. Examples of this data are the jobs's time of submission, the time it was started, and the time it ended. It also has methods for handling the submission, start, and finish of the job during its lifecycle in Schedulus.

A job in Schedulus maintains state regarding its current position in its lifecycle. A jobs lifecycle is defined as the time interval from when it is scheduled for submission until it is ended and removed from the Schedulus data structures. At any given moment a job's ID can be found residing in the schedule's queue waiting to run, in the set of running jobs, or in neither.

B. Cluster

A cluster object represents the state of the computational machinery Schedulus is simulating. Possibilities for computational machinery can range from actual hardware such as Summit located at Oak Ridge National Laboratory to theoretical designs for hardware. Currently the specification for a cluster object contains the number of total available nodes and the number of currently idle nodes. Included are methods for safely allocating and deallocating these nodes to and from jobs. In the future this specification will be expanded to capture more sophisticated hardware characteristics.

C. Event Semantics

The scarce resource in most HPC systems is the number of available nodes, or more granularly processors [29]. Schedulers like Schedulus aim to maximize the use of these resources to maximize performance. Schedulus uses events to simplify the process of changing job state. By not having to worry about orchestrating job state transition events, more attention can be given to the evaluation of performance of different scheduling strategies. Schedulus responds to two types of discrete-events: job submission and job termination. Using these events Schedulus can treat each job independently from one another; it only has to sychronize the allocation and deallocation of the scarce resources defined by the cluster object.

D. Job Submission Events

The workload log define a submission time for all jobs given in the log. Schedulus uses this time, usually given in seconds, to schedule a discrete-event at that time. Simulus then fires an event at that time that takes the form of a submission event handler, that is just a Python function. This function takes care of determining if the job can be run immediately, or if it should be appended to the schedule.

E. Job Termination Events

When a job ends its execution phase, it is removed from the running list and the number of computing units it was using is returned to the cluster. Schedulus then runs as many jobs as is allowed by the schedule space taking one at a time from the schedule queue. Running a job in Schedulus means add it to the list of running jobs and schedule a job termination event at some time t equal to the current simulation time plus the time it takes for the job to run. If at some job termination event time no jobs can be safely run during a job termination event, Schedulus runs a backfilling operation if one is predefined at runtime.

IV. Results

To test the effectiveness and correctness of Schedulus, experiments were run to gather statistics. The worload logs used in the experiments were the HPC2N log, the CTC-SP2 log, and the MetaCentrum log given in the Parallel Workloads Archive. The HPC2N log contains 202,871 jobs. The CTC-SP2 log contains 77,222 jobs. The MetaCentrum log contains 103,656 jobs. Any times reported here are simulation times, but can be interpreted as seconds because the workload logs report event times in seconds. The statistics calculated were the average number of running jobs in the system, the average bounded slowdown of jobs, and the average wait time of jobs. More specifically, the average number of running jobs was calculated by summing the number of running jobs each time an event was fired and dividing by the number of events fired. The average bounded slowdown was calculated by taking the mean of all jobs' bounded slowdown. It reduces the emphasis on short jobs exhibited by the typical slowdown metric. Bounded slowdown is defined as:

$$max(\frac{T_w + T_r}{max(T_r, \tau)}, 1)$$

where T_w and T_r are the waiting time and running time of a job. τ is a discretionary parameter called the "interactive threshold" in the formula and is set to 10 seconds for our experiments [2], [18].

The average wait time was calculated as the mean of the job wait times. Wait time is calculated internally by Schedulus and is equal to the time the job started running minus the time the job was submitted.

TABLE I: Average Number of Running Jobs Result Table

	FCFS	FCFS w/ EASY Backfilling
	Ave. Num. Running Jobs	Ave. Num. Running Jobs
HPC2N	47	47
CTC-SP2	35	35
MetaCentrum	267	267

V. Discussion

The ability to simulate full workloads and collect standard statistics shows the viability of the event-driven approach to HPC scheduling simulation. Some improvements can be made to Schedulus to move it closer to being a contender for

TABLE II: Average Bounded Slowdown Result Table

	FCFS	FCFS w/ EASY Backfilling
	Ave. Bounded Slowdown	Ave. Bounded Slowdown
HPC2N	208	86
CTC-SP2	4276	181
MetaCentrum	7	7

TABLE III: Average Wait Time Result Table

	FCFS	FCFS w/ EASY Backfilling
	Ave. Wait Time	Ave. Wait Time
HPC2N	16189	9570
CTC-SP2	768179	67508
MetaCentrum	1459	1458

conducting real academic research. The granularity and complexity of the cluster representation can be extended to better capture the hardware characteristics. Doing so would allow for more realistic simulation of actual scheduling systems. Also, in the future we will compare performance of Schedulus to other scheduling simulators like the ones mentioned above. This will give us better indications for areas where improvement still needs to take place, both in terms of overall runtime and simulation quality. An area of immediate neccesary improvement for Schedulus is the backfilling algorithm strategy. Establishing a computationally less expensive way to find backfilling candidates would greatly improve the runtime of Schedulus. Currently, research is being conducted to use parallelism to find backfilling candidates.

VI. Conclusion

Scheduling systems in HPC are extremely important. They are principal components of maximizing utilization and therefore performance. It is expensive to test new scheduling algorithms on real hardware. HPC supercomputers are expensive, costing hundreds of millions of dollars annually to operate. There is no room for experimental scheduler testing that could lead to long job delays and even errors. Schedulus is an experimental event-driven HPC scheduling simulator capable of running full HPC system workloads. It simplifies the maintence of job state so more attention can be paid to the research and development of better scheduling algorithms. Promising intial experimental results confirm the viability of event-driven scheduling semantics. Schedulus is being actively developed.

References

[1] J. W. Fowler, L. Monch, and O. Rose. Scheduling and simulation: The role of simulation in scheduling. In *Handbook of Production Scheduling (pp.109-133)*, 2006.

[2] Y. Fan, Z. Lan, P. Rich, W. Allcock, M. Papka, B. Austin, and D. Paul. Scheduling Beyond CPUs for HPC. In *HPDC*, 2019.

[3] P. Qiao, X. Wang, X. Yang, Y. Fan, and Z. Lan. Joint Effects of Application Communication Pattern, Job Placement and Network Routing on Fat-Tree Systems. In *ICPP Workshops*, 2018.

[4] Y. Fan, Z. Lan, T. Childers, P. Rich, W. Allcock, and M. Papka. Deep Reinforcement Agent for Scheduling in HPC. In *IPDPS*, 2021.

[5] L. Yu, Z. Zhou, Y. Fan, M. Papka, and Z. Lan. System-wide Trade-off Modeling of Performance, Power, and Resilience on Petascale Systems. In *The Journal of Supercomputing*, 2018.

[6] Y. Fan. Application Checkpoint and Power Study on Large Scale Systems. In *IIT Tech. Report*, 2021.

[7] P. Qiao, X. Wang, X. Yang, Y. Fan, and Z. Lan. Preliminary Interference Study About Job Placement and Routing Algorithms in the Fat-Tree Topology for HPC Applications. In *CLUSTER*, 2017.

[8] N. Matloff. Introduction to discrete-event simulation and the simpy language. *Davis, CA. Dept of Computer Science*, 2008.

[9] Y. Fan. Job Scheduling in High Performance Computing. In *Horizons in Computer Science Research*, 2021.

[10] W. Allcock, P. Rich, Y. Fan, and Z. Lan. Experience and Practice of Batch Scheduling on Leadership Supercomputers at Argonne. In *JSSPP*, 2017.

[11] Y. Fan and Z. Lan. Exploiting Multi-Resource Scheduling for HPC. In *SC Poster*, 2019.

[12] Y. Fan and Z. Lan. DRAS-CQSim: A Reinforcement Learning based Framework for HPC Cluster Scheduling. In *Software Impacts*, 2021.

[13] B. Li, S. Chunduri, K. Harms, Y. Fan, and Z. Lan. The Effect of System Utilization on Application Performance Variability. In *ROSS*, 2019.

[14] Y. Fan, P. Rich, W. Allcock, M. Papka, and Z. Lan. ROME: A Multi-Resource Job Scheduling Framework for Exascale HPC System. In *IPDPS poster*, 2018.

[15] Y. Fan, P. Rich, W. Allcock, M. Papka, and Z. Lan. Hybrid Workload Scheduling on HPC Systems. In *Advances in Computer and Network Simulation and Modelling*, 2021.

[16] Y. Fan, Z. Lan, and M. Papka. Intelligent Job Scheduling for Next Generation HPC Systems. In *SC Doctoral Showcase*, 2021.

[17] Y. Fan. Intelligent Job Scheduling on High Performance Computing Systems. In *IIT Dissertation*, 2021.

[18] D. Feitelson, D. Tsafrir, and D. Krakov. Experience with using the Parallel Workloads Archive. *JPDC'14*.

[19] D. Klusáček and H. Rudová. Alea 2 - job scheduling simulator. *SIMUTools 2010 - 3rd International ICST Conference on Simulation Tools and Techniques*, 2010.

[20] 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*, 2018.

[21] M.Chadha, J. John, and M. Gerndt. Extending SLURM for Dynamic Resource-Aware Adaptive Batch Scheduling, 2020.

[22] M. Jette, A. Yoo, and M. Grondona. SLURM: Simple Linux Utility for Resource Management. In *JSSPP'03*.

[23] R. Buyya and M. Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience*, 2002.

[24] J. Liu. Simulus: Easy breezy simulation in python. In *2020 Winter Simulation Conference (WSC)*, pages 2329–2340, 2020.

[25] A.W. Mu'alem and D.G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. *IEEE Transactions on Parallel and Distributed Systems*, 2001.

[26] Y. Fan, P. Rich, W. Allcock, M. Papka, and Z. Lan. Trade-Off Between Prediction Accuracy and Underestimation Rate in Job Runtime Estimates. In *CLUSTER*, 2017.

[27] A. Mu'alem and D. Feitelson. Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling. *TPDS'01*.

[28] Y. Fan, P. Rich, W. Allcock, M. Papka, and Z. Lan. Exploring Machine Learning to Adjust Job Runtime Estimate for High-Performance Computing. In *Greater Chicago Area System Research Workshop (GCASR)*, 2017.

[29] Y. Fan, P. Rich, W. Allcock, M. Papka, and Z. Lan. Next generation workload management system for high performance computing systems. In *Greater Chicago Area System Research Workshop (GCASR)*, 2018.