CQSim+: Symbiotic Simulation for Multi-Resource Scheduling in High-Performance Computing

Yash Kurkure
University of Illinois Chicago
Chicago, Illinois, USA
ykurku2@uic.edu

Shambhawi Sharma
University of Illinois Chicago
Chicago, Illinois, USA
sshar102@uic.edu

Xin Wang
Computer Science
University of Illinois Chicago
Chicago, Illinois, USA
xwang823@uic.edu

Michael E. Papka
University of Illinois Chicago
Chicago, Illinois, USA
Argonne National Laboratory
Lemont, Illinois, USA
papka@uic.edu

Zhiling Lan
University of Illinois Chicago
Chicago, Illinois, USA
Argonne National Laboratory
Lemont, Illinois, USA
zlan@uic.edu

Abstract

Efficient job scheduling is crucial in high-performance computing (HPC), balancing user demands for quick job turnaround with facility goals for high resource utilization. Traditional scheduling requires users to specify a system at job submission, which can lead to inefficiencies. A unified scheduling approach, viewing the resources within a computing facility as an integrated pool, promises improved resource use and reduced job wait times. This paper presents CQSim+, an open-source, discrete event-driven simulator tailored for symbiotic multi-resource scheduling. CQSim+ supports dynamic simulation by continuously integrating real-time data from job schedulers, enabling adaptive scheduling based on the system's current state. Through extensive experimentation, we demonstrate CQSim+'s ability to enhance resource utilization and decrease job wait times in both homogeneous and heterogeneous HPC environments. Additionally, we present a case study that coordinates job scheduling between two production systems, illustrating how CQSim+ can effectively optimize job scheduling across distinct systems.

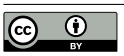
CCS Concepts

• **Computing methodologies** → **Modeling methodologies**; **Discrete-event simulation**; **Real-time simulation**; **Simulation tools**; • **Social and professional topics** → *System management*.

Keywords

Multi-resource scheduling, Simulation tool, Resource management systems, High-performance computing

ACM Reference Format:
Yash Kurkure, Shambhawi Sharma, Xin Wang, Michael E. Papka, and Zhiling Lan. 2025. CQSim+: Symbiotic Simulation for Multi-Resource Scheduling in High-Performance Computing. In *39th ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (SIGSIM-PADS '25), June 23–26, 2025, Santa Fe, NM, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3726301.3728404



SIGSIM-PADS '25, Santa Fe, NM, USA

ACM ISBN 979-8-4007-1591-4/25/06
https://doi.org/10.1145/3726301.3728404

1 Introduction

In high-performance computing (HPC), job scheduling is critical as the intermediary between users and the resources of the computing facility. It manages the influx of user jobs on available resources to achieve rapid turnaround times for its users and high resource utilization in line with the facility's goals. Since resources are often divided among various computing systems, users must specify a particular system when submitting their jobs. This rigid approach to system selection can result in inefficiencies. For instance, users may request the most powerful system within a facility despite a less powerful system with a shorter wait queue, potentially offering much faster application turnaround times.

Transitioning from *siloed single-system scheduling* to *a unified multi-resource scheduling* presents a promising solution to the problem. By viewing the entire computing facility as an integrated pool of diverse resources, multi-resource scheduling can accommodate a broad mix of applications with varying resource requirements. This global and uniform access facilitates a more effective allocation of resources across diverse applications, enabling the balancing and optimization of resource utilization within the facility.

Multi-resource scheduling simulation is an indispensable tool for designing and assessing multi-resource scheduling policies. It allows researchers and practitioners to explore various scheduling algorithms, resource allocation policies, and system configurations in a controlled environment. In particular, the community is in critical need of *symbiotic simulation* to support dynamic decision-making at runtime. Such a symbiotic simulation framework must leverage real-time data the job scheduler generates to update the simulation model continuously. It must allow real-time adjustments and optimization based on the system's current state. By integrating real-time data into the simulation process, symbiotic simulation can bridge the gap between static modeling and the dynamic nature of multi-resource scheduling environments, ultimately improving resource allocation and job scheduling.

Despite its critical importance, the field lacks an open-source, high-fidelity multi-resource scheduling simulation tool designed for symbiotic simulation. It creates a significant gap for the community, limiting innovation in developing advanced multi-resource scheduling techniques.

To bridge the gap, we present ***CQSim+***, a multi-resource scheduling simulator designed for symbiotic simulation. CQSim+ is built on CQSim, an open-source, discrete event-driven multi-resource simulator[27]. Discrete event simulation is renowned for achieving high-fidelity computer system simulations swiftly. CQSim+ extends the capabilities of CQSim to support multi-resource scheduling. In the development of CQSim+, we have addressed several technical challenges. The first is to transition from offline simulation to symbiotic simulation. Although CQSim has been validated and utilized by the community for over ten years, it is primarily designed for offline trace analysis. To facilitate symbiotic simulation, our design incorporates several techniques that enable real-time data acquisition and streaming between CQSim+ and job schedulers. Another challenge is to accelerate the simulation to support real-time decision-making processes. We adopt a multi-process architecture for CQSim+, enabling it to perform rapid simulations necessary for runtime optimizations. Specifically, this work makes the following key contributions:

- We introduce CQSim+, an open-source simulator that supports high-fidelity symbiotic simulation for multi-resource scheduling. Built upon a multi-process, discrete event-driven simulation architecture, CQSim+ enables efficient modeling of complex multi-resource environments, facilitating real-time data acquisition and dynamic scheduling decisions.
- We conduct extensive experiments demonstrating the advantages of using CQSim+ for multi-resource scheduling across homogeneous and heterogeneous systems. These experiments highlight CQSim+'s capability to optimize resource utilization and reduce job wait times.
- We present a case study utilizing CQSim+ to coordinate the scheduling of jobs across the Polaris and Theta supercomputers at Argonne Leadership Computing Facility (ALCF) [10]. This case study illustrates how CQSim+ can effectively manage and optimize workloads between distinct systems, providing valuable insights into the practical use of multi-resource scheduling in real-world HPC environments.

2 Related Work

A job scheduler, or batch scheduler, is a system-level scheduling component that manages the allocation of jobs to computing resources (e.g., compute nodes) based on site policies and current resource availability.

Two widely used batch schedulers in HPC environments are Slurm [32] and PBS [3], each offering distinct capabilities to manage and optimize resource use in large-scale HPC systems. First Come First Serve (FCFS) with EASY backfilling is the most widely used heuristic, which sorts the jobs in the wait queue according to their arrival times and executes jobs from the head of the queue[21]. If the available resources are insufficient for the first job in the queue, the scheduler will reserve the resources for this job. Backfilling is often used in conjunction with job reservation to enhance system utilization. It allows subsequent jobs in the wait queue to move ahead under the condition that they do not delay the existing reservations.

Multi-resource scheduling methods have been extensively explored in various studies. In grid computing, "meta-scheduling" is frequently used for scheduling applications among the distributed resources belonging to different administrative domains [13]. Existing multi-resource scheduling methods can be broadly classified as either heuristic or classical optimization approaches. Heuristic methods, like best-fit and list scheduling, are commonly used to make allocation decisions based on simple rules or strategies [15, 28]. Other studies typically formulate the problem as a mixed-integer linear problem (MILP) and apply dynamic programming or greedy algorithms to solve it [8, 12].

Both Slurm and PBS offer basic interfaces for resource provisioning across multiple systems. In Slurm, a heuristic method called federation scheduling is employed, which assigns jobs to the system that offers the earliest start time subject to its queue of pending and running jobs[32]. On the other hand, PBS adopts a different heuristic method known as peer scheduling, which transfers jobs from a busy system to an idle one[3]. These systems operate on the similar principle of selecting the system for job execution using the greedy method.

Simulation is an essential technique in scheduling and fault tolerance research, providing a controlled environment for testing and evaluation. It is a well-utilized technique used to prove the benefits of multi-resource scheduling in theory as the general k-resource problem [18], as well as in practice such as HPC [19] and Cloud [20]. Over the years, numerous simulators have been developed to support research in grid, HPC, and cloud computing environments. These simulators typically fall into one of three main categories. The first category consists of simulators that employ real-life resource management systems (RMS) in the simulation mode. For example, the ScSF[24] and Slurm simulator [26] emulate a real system by using Slurm Workload Manager inside its core to mimic the real RMS realistically. A similar "simulaton mode" is supported in Moab[1] and PBS. These simulators provide highly realistic scheduling behaviors, closely replicating the RMS configurations used in many large-scale HPC systems, though they are often limited to specific environments. The second category includes simulators that leverage an underlying simulation toolkit, such as SimGrid [6], GridSim [17], and CloudSim [5]. Examples include Batsim [9], ALEA [16], and WRENCH [7]. Batsim is a batch scheduling simulator built on top of SimGrid, designed to support a wide variety of event-based scheduling algorithms. Based on GridSim, ALEA provides flexible environments for simulating job scheduling in grid and HPC systems. WRENCH, an extension of SimGrid, introduces higher-level abstractions to facilitate the simulation of real-world CI (CyberInfrastructure) scenarios, including scientific workflows across multi-resource environments. The third category includes AccaSim [14], and QSim [30], which are built from scratch. AccaSim is a flexible Python-based scheduling simulator designed to test scheduling policies in diverse HPC workloads. Qsim is an event-driven scheduling simulator supporting utility-based policies to balance various scheduling requirements and priorities.

Although existing simulators are valuable for scheduling research, none fully support symbiotic simulation, which integrates

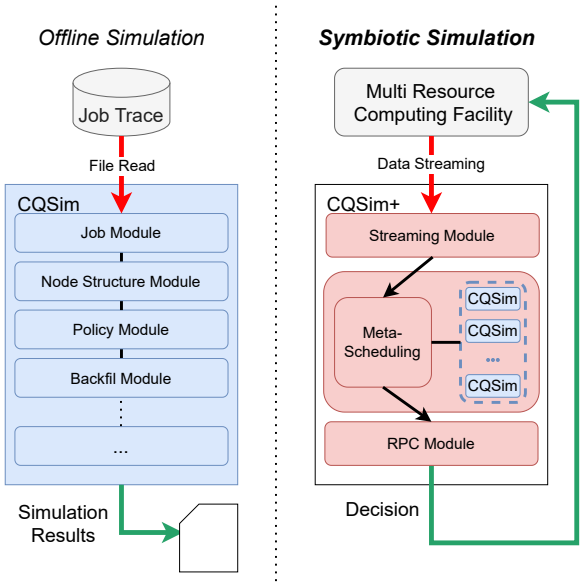


Figure 1: Transitioning from CQSim to CQSim+. (Left) traditional offline simulation CQSim, (Right) symbiotic simulation CQSim+.

system modeling with real-time data analytics to improve operational decision-making. Onggo et al. introduce the symbiotic simulation system (SSS) [22], a digital twin of a physical system that enables continuous interaction through simulation, data analytics, optimization, and machine learning.

However, they highlight several challenges, such as synchronizing real-time data with model execution, managing hybrid models, and adapting to rapidly changing physical conditions. Addressing these issues requires advancements in model validation, real-time decision-making optimization, and adaptive simulation frameworks. This work seeks to fill this gap by extending existing simulators with symbiotic simulation capabilities, allowing for real-time feedback and adaptation during the simulation.

3 Design of CQSim+

CQSim+ is a multi-resource scheduling simulator designed explicitly for symbiotic simulation. It enables the real-time interaction between a physical system and its simulated model.

CQSim+ is built upon CQSim [27]. Originally developed for IBM BlueGene/P systems in 2009, CQSim is an open-source, discrete-event-driven job scheduling simulator with enhanced functionalities[2, 30, 31]. CQSim processes job traces provided in the Standard Workload Format (SWF), a widely used format for job scheduling [29] simulations. It models job scheduling as a sequence of events, and each event (e.g., job arrival, job start, job completion, etc.) occurs at a specific time and marks a state change in the system. Based on these events, CQSim emulates job submission, scheduling, allocation, and execution under a specific scheduling policy. The simulation operates on a single thread, reading jobs sequentially from a trace file, which ensures simplicity and transparency. Its functionalities are distributed among its modules that keep track of the state of specific components in the system as shown in Figure 1 for offline simulation. Using its modular interface, this work extends CQSim to CQSim+, which enables the usage of its offline simulation modules within the symbiotic simulation system.

CQSim+, as a symbiotic simulation system, can interact in real-time with the resource management system of a multi-resource computing facility (e.g., PBS). The physical system drives the simulation using live data, allowing the simulation to provide predictions and assist in decision-making. In this setup, the physical HPC job scheduler manages job scheduling, while the original CQSim+ acts as a digital simulator reflecting scheduling decisions and outcomes. This integrated design allows CQSim+ to respond dynamically to real-time changes within the physical system, optimizing scheduling and resource allocation through a continuous feedback loop.

Figure 1 illustrates the differences between offline simulation with CQSim and symbiotic simulation with CQSim+. CQSim+ consists of three main components in this design: *a streaming module* for data acquisition, *a meta-scheduling module* for simulation-related components, and *an RPC module* to interact with the job scheduler. The Streaming module enables high-speed transactions of real-time job queue events and machine states from the systems of the multi-resource computing facility, passing these data streams to the meta-scheduler for processing. The meta-scheduler module manages multiple instances of CQSim as shown by the blue instances in Figure 1. It also allows the creation of multiple copies of each instance and uses the copy to simulate the future of a particular system. The future predictions from each instance's copy are then gathered as feedback to help the simulated meta-scheduler make decisions for the real system. Finally, an RPC module is developed using the jobs schedulers API that sends back the decision of the CQSim+ to the facility's job scheduler.

3.1 Data Acquisition and Streaming

Typically, the job scheduler is composed of various services running on the system nodes. Consequently, events within the system can originate from multiple locations. For example, PBS consists of four main types of services: the server, scheduler, communication service, and MoM server (Machine-oriented Mini Server)[4], each running on separate nodes. The server is responsible for handling jobs and PBS-related commands, the scheduler contains the logic for scheduling jobs, the MoM handles the execution lifecycle of a job on each compute node, and the communication service manages communication between all PBS-related services.

Acquiring data from the job scheduler in real-time is necessary for symbiotic simulation. Most job schedulers have an interface that allows developers and users to add functionality. In the case of PBS, a Python interface called PBS Hooks[4] is provided. PBS Hooks are Python scripts triggered by particular events, such as job submit, run, end, etc. Similarly, Slurm provides the plugin interface[25] to its users and developers. These interfaces can be used to acquire data in real-time.

As stated earlier, the nature of the events is distributed. In the case of PBS, the Python hook scripts only run on the machine where the event occurred. The manual for PBS [4] defines events that may

```python
# logging_hook.py
import pbs
import sys
import os
import time


e = pbs.event()

# Get the timestamp at which the event occurs
timestamp = int(time.time() * 1000)

# Get the job id
job_id = str(e.job.id)

# Get the event type
event_type = pbs.event().type

try:
    # Check for the event that triggered this hook
    if event_type is pbs.QUEUEJOB:
        event_str = 'q'
    elif event_type is pbs.EXECJOB_BEGIN:
        event_str = 'mom_r'
    elif event_type is pbs.EXECJOB_END:
        event_str = 'mom_e'


    # Create the log entry
    log_string = f'{timestamp}, {job_id}, {event_str}'

    # Write to some file for logging or data streaming
     service
    write_to_some_file(log_string)

    # accept the event
    e.accept()

except SystemExit:
    pass
except:
    e.reject("Failed to set job priority")
```

Figure 2: An example of a PBS Hook. The hook is configured to run at the queuejob, execjob_begin, and execjob_end events within PBS. Each PBS Hook reads the event detail from the event-object acquired on line 8. Then, as shown in lines 20-26, the event can be identified. Line 33 is where the hook writes the event data to a file. The file is created on the machine where the hook runs.

occur at the server and the MoM. But the hooks configured to run at server events will only run on the head node, and those configured to run at MoM events run on the compute node. This requires us to collect data from multiple nodes in the system and not just the node where the scheduler runs. Figure 2 is an example of a simple hook that runs for the queue, execjob_begin, and execjob_end events and writes a log string to a file. The queue job is a server-side event; hence, for the queue event, this script would run on the head node where the PBS server runs. The execjob_begin and exejob_end events are MoM-sided; hence, they will run on some compute node. Due to this, the log files the hook would write would also be written

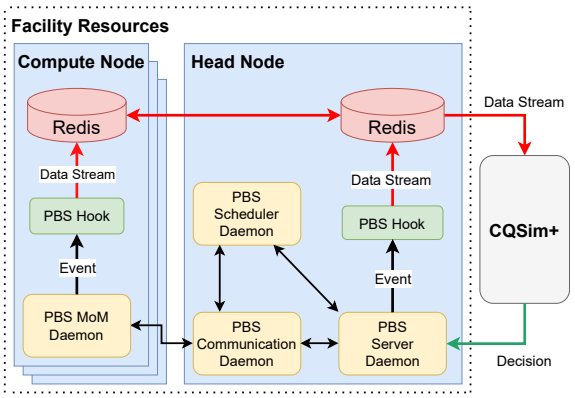


Figure 3: Data streaming between CQSim+ and PBS components.

to different machines in the system. To address this problem, storing these events in a database is necessary.

To address the issue of transferring event data between the scheduler and simulator and the issue of the origin of events being distributed in the system, Redis offers a fast in-memory distributed database and publishes subscribe streams. Figure 3 provides an overview of how Redis works with PBS to stream data to CQSim+. The developer writes a PBS Hook using Redis's Python client that publishes events to a Redis stream. While the stream data is stored in Redis's distributed database, CQSIM+ subscribes to the stream and consumes messages from the stream. Redis guarantees subscribers get events in the order they were published in.

From the perspective of CQSIM+, its interactions with data streaming services are handled by the streaming module. It is responsible for implementing the client interface of the streaming service (Redis), listening to the stream of events, and propagating the events to the meta scheduler.

3.2 Meta-Scheduling

Meta-scheduling involves distributing workloads across multiple systems. CQSim+ extends CQSim by managing multiple instances, each corresponding to an individual system of the facility. This approach enables the running of predictive simulations on each system before determining the optimal placement for a job.

The meta-scheduling module is responsible for maintaining multiple instances of CQSim. Figure 1 shows a list of CQSim instances under the meta-scheduling module. Each CQSim instance represents a single system operating in a facility. It also contains the logic for advancing simulators, keeping each simulated instance in sync with the job scheduler and creating copies of instances to predict the future.

Multi-process Architecture. Future states of individual CQSim instances can be predicted by creating a copy of the instance and then advancing its simulation based on new information. This is achieved using the multi-processing package of Python, which essentially forks the main process in which CQSim+ is running

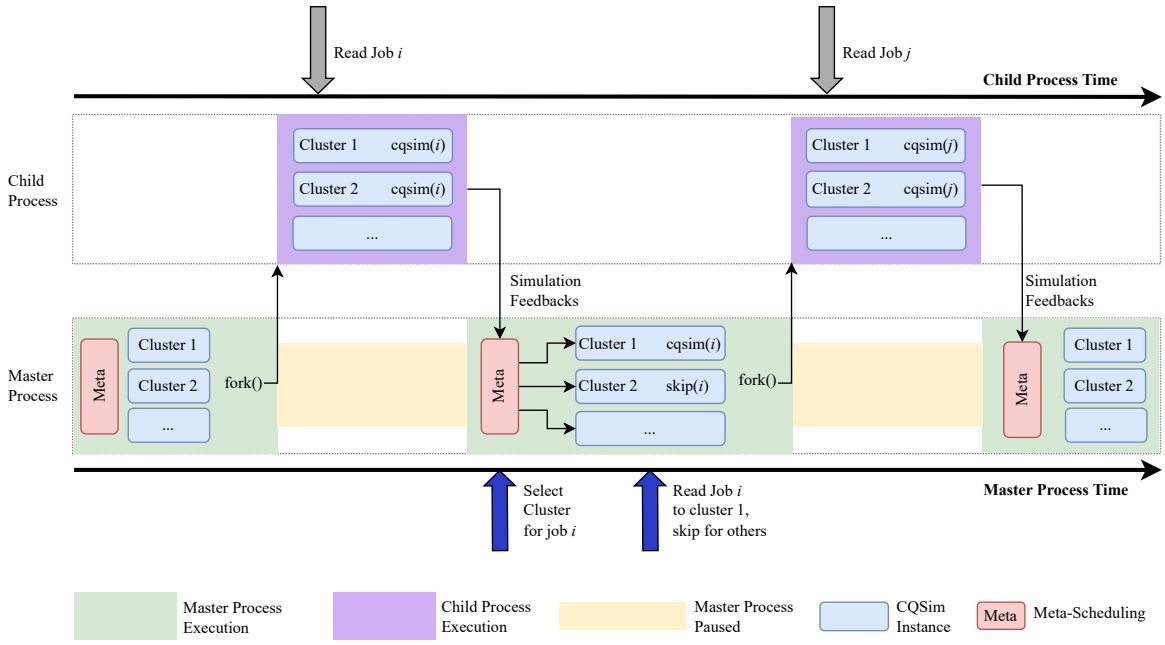


Figure 4: A time diagram of CQSim+ performing meta-scheduling for multiple systems.

and creates a new process with a copy of CQSim+ having the same exact copies of each CQSim instance.

The CQSim instances of the main process are kept in sync with the job scheduler. When a new job arrives, the process is forked, creating a child process with copies of each CQSim instance. Inside the child process, each simulator reads this new job and advances to completion. At the end of the simulation in the child process, each simulator has a prediction. These predictions are sent back to the main process as simulation feedbacks as shown in Figure 4. The meta-scheduling component reads this feedback and selects the optimal system in the main process. Only then is the simulator of the optimal system advanced in the main process with the new job. All other CQSim instances in the main process skip this job since they were not chosen by the meta-scheduling component. Then the main thread waits to read the next job from the stream.

Scalability. Without the help of multiprocessing, every time a new job arrives, a new instance of CQSim would need to be created. Additionally, this new instance would need to read all previous jobs to be in sync with the job scheduler in real-time. This incurs a significant computational overhead. As described before, multiprocessing allows creating copies of CQSim instances that have already processed past events. This saves us the computational time of re-running an instance from the beginning. It is important to note that we use multiprocessing instead of threading in Python for scalability reasons. This is because the global interpreter lock in Python does not allow parallelism between threads while using a multi-core machine [23].

3.3 Decision Making via RPC

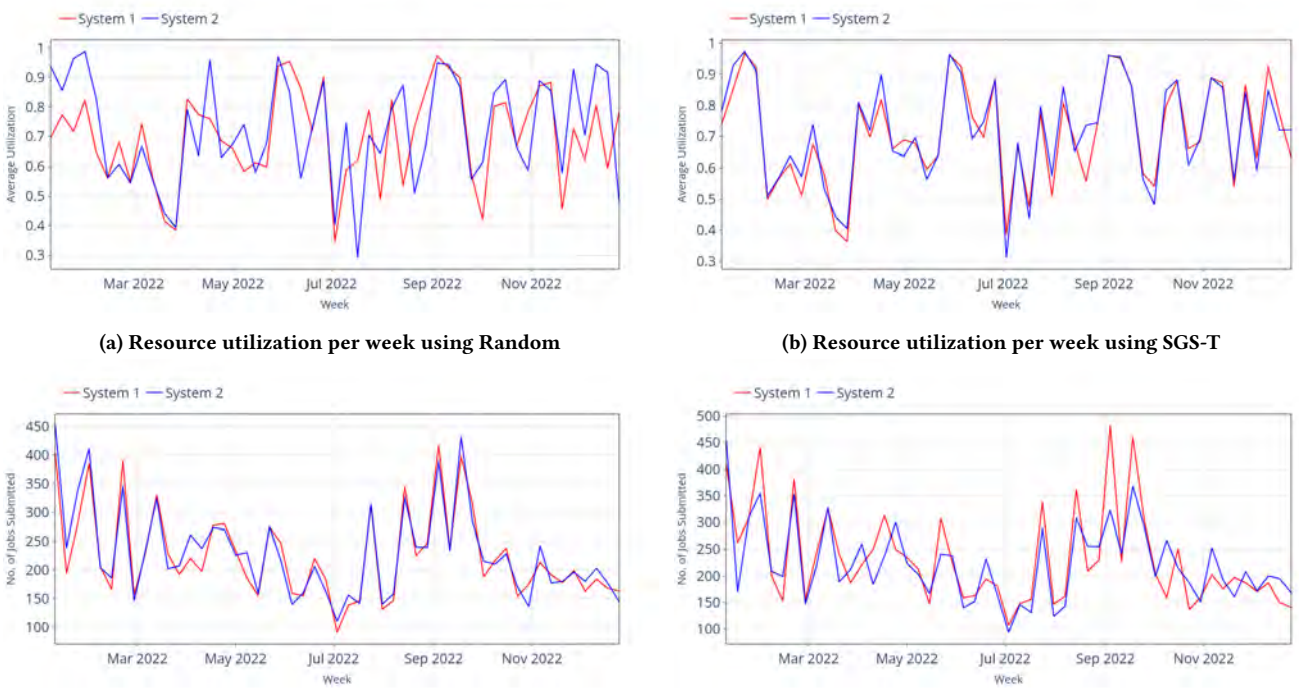
The results from CQSim+ are fed back into PBS to drive and guide multi-resource scheduling in the facility's physical systems. This real-time decision feedback is achieved using Remote Procedure Calls (RPC). In the case of PBS, it provides the Batch Interface Library [4] written in C++. The routines exposed by the library allow connecting to the PBS server daemon and issuing PBS commands, including job-related commands such as submit, alter, run, etc. The RPC module is responsible for providing wrappers for these scheduler-specific routines, creating a layer of compatibility between CQSim+ and any other resource management systems.

4 Experiments

To investigate multi-resource scheduling using CQSim+, we evaluate a scheduling scenario where users submit their jobs to a computing facility containing multiple computing systems. Without a meta-scheduling simulator, a user might randomly select a system that may not be optimal from a scheduling perspective.

Since a facility may contain multiple systems of the same or different types, we study two cases. The first is the *homogeneous* configuration, where a facility has multiple systems employing the same hardware. The second is the *heterogeneous* configuration, where the systems use different hardware.

The experiments are performed using the job logs from the Theta system at ALCF [10], which contain 23,911 jobs after filtering out debug-related jobs. The Theta system comprised 4,392 nodes, of which 32 were reserved exclusively for debugging. For simulation purposes, we removed debugging jobs and the nodes used for debugging. Additionally, for the multi-system setup under simulation, we ensure that the total number of nodes across all systems is equal to the number of nodes in Theta. Using CQSim+, the experiments are designed to meta-schedule jobs among two systems for both the homogeneous and heterogeneous cases. This is done using two instances of CQSim within CQSim+.



(a) Resource utilization per week using Random

(b) Resource utilization per week using SGS-T

(c) Jobs submitted per week using Random

(d) Jobs submitted per week using SGS-T

Figure 5: Resource utilization and the number of jobs submitted per week in the homogeneous case: (left) Random, (right) SGS-T.

Job Size	Job Count and %	Average Wait Time (min)		
		Random	SGS-T	% Improv.
(0-128]	12133 (55.9%)	214.6	131.4	38.8
(128,256]	4841 (22.3%)	558.1	416.3	25.4
(256,512]	1881 (8.7%)	879.6	691.6	21.4
(512,1024]	2438 (11.2%)	1748.9	1465.1	16.2
(1024,2180]	429 (2.0%)	1741.2	1301.6	25.2
Overall	21722	551.1	416.2	24.5

Table 1: Average wait times binned by job size (Homogeneous)

4.1 Homogeneous

To illustrate a scenario where a user must choose between homogeneous systems, the setup consists of two systems, each with 2,180 processes. We selected 2,180 for two reasons:

(1) To test the scenario where the user does not have a clear idea of which system to choose, as they are identical in size and performance. Thus, the selection may be random.
(2) To maintain consistency with the amount of resources used by the original job trace. This is achieved by ensuring that the total number of nodes across both systems equals 4,360, matching the number of nodes of the original system of the job trace.

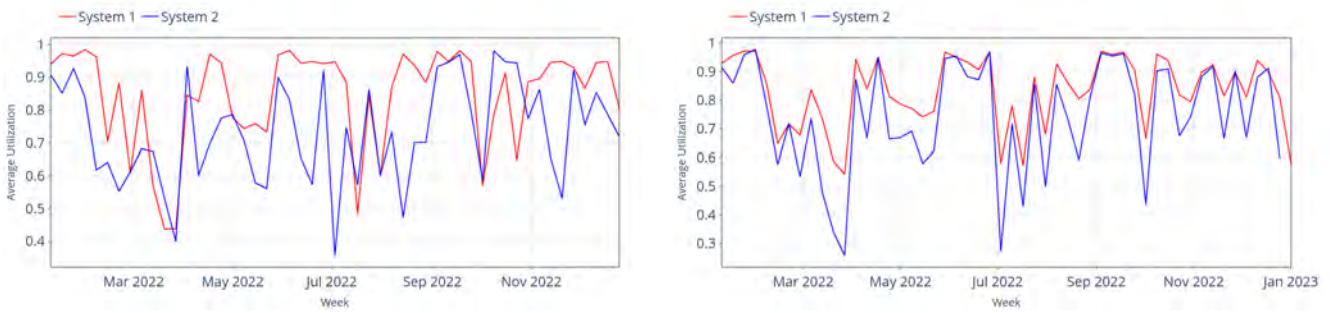
With these considerations in mind, we designed our experiment using CQSim+ with the two systems, evaluating the following meta-scheduling strategies:

(1) The job is assigned to either system with a 50% probability. This approach represents the absence of simulation assistance to the user. As described previously, the user's choice is random since both systems are identical in speed and performance. We label this approach as **Random**.
(2) The job is assigned to the system with the best turnaround time. This makes a case for the presence of simulation guidance from CQSim+ to the user, where the user always schedules the job on the system with the optimal turnaround time. We refer to this strategy as **SGS-T** (Simulation Guided Scheduling based on Turnarounds).

We evaluate the simulation results using two key metrics: *average wait time* and *system utilization*.

Average Wait Time. Table 1 shows the average wait times grouped by the job size under the Random and SGS-T policies. SGS-T performs 24.5% better overall and in each job size category compared to Random. Similarly, Table 2 shows the average wait times grouped by wall time, where each job category benefits from SGS-T.

Average Utilization. The average utilization per day is calculated for all systems under each meta-scheduling strategy. Figure 5(a) shows the average utilization of both systems when Random is used for meta-scheduling decisions, comparing this to figure 5(b), which



(a) Resource utilization per week using User(0.6)

(b) Resource utilization per week using SGS-T

Figure 6: Resource utilization per week in the heterogeneous case: (left) User(0.6), (right) SGS-T.

Walltime (min)	Job Count and %	Average Wait Time (min)		
		Random	SGS-T	% Improv.
(10,30]	2344 (10.8%)	534.1	402.9	24.6
(30,60]	5344 (24.6%)	298.9	218.0	27.0
(60,120]	1662 (7.7%)	403.1	338.5	16.0
(120,250]	7918 (36.5%)	410.3	285.6	30.4
(250,500]	2865 (13.2%)	902.8	715.1	20.8
(500,1000]	992 (4.6%)	1528.4	1155.5	24.4
(1000,1500]	597 (2.7%)	1843.3	1527.6	17.1
Overall	21722	551.1	416.2	24.5

Table 2: Average wait times binned by walltime (Homogeneous)

Job Size	Job Count and %	Average Wait Time (min)		
		User(0.6)	SGS-T	% Improv.
(0-128]	12133 (55.9%)	344.0	218.6	36.4
(128,256]	4841 (22.3%)	952.2	676.2	29.0
(256,512]	1881 (8.7%)	1456.0	1185.8	18.6
(512,1024]	2438 (11.2%)	3006.6	2351.5	21.8
(1024,2180]	429 (2.0%)	3073.3	2142.2	30.3
Overall	21722	928.6	681.7	26.6

Table 3: Average wait times binned by job size (Heterogeneous)

shows the utilization under SGS-T. At a glance, it is observable that the average utilization of both systems under SGS-T is almost the same (load is balanced), whereas for Random there are many instances where the utilization is less for one system than the other.

To investigate further, we look at Figures 5(c) and (d) that show the job submission pattern per system for Random and SGS-T, respectively. In the case of Random, jobs are equally distributed among the systems, whereas for SGS-T, the jobs are not equally distributed. This tells us that SGS-T balances the overall load of the jobs equally among the systems, whereas Random only distributes the jobs equally among the systems.

4.2 Heterogeneous

Heterogeneity breaks the uniformity in size and performance, introducing additional factors for consideration when scheduling user jobs. In such cases, it may be appealing for users to choose the faster and/or bigger system.

To evaluate the performance of simulation guidance in a heterogeneous environment, we make two key changes to the experimental design from the homogeneous case:

(1) *Differing system performance.* One system is slower than the other. This is done by scaling the runtime from the original job trace by a factor of y.
(2) *User selection probability.* Instead of comparing SGS-T to Random, we introduce a new strategy that accounts for the probability of a user selecting the faster system. The strategy is denoted as *User(x)*, where x represents the probability of the user choosing the faster system.

The simulations are conducted with $y = 1.3$ for system 2 and $x = 0.6$, creating a scenario where system 2 runs 1.3 times slower than system 1, and the user selects the faster system (system 1) with a probability of 60%.

Walltime (min)	Job Count and %	Average Wait Time (min)		
		User(0.6)	SGS-T	% Improv.
(10,30]	1263 (5.8%)	1099.9	641.5	41.7
(30,60]	5012 (23.1%)	576.3	443.1	23.1
(60,120]	2512 (11.6%)	570.7	439.2	23.0
(120,250]	8272 (38.1%)	659.8	469.7	28.8
(250,500]	2925 (13.5%)	1446.1	1057.2	26.9
(500,1000]	1137 (5.2%)	2438.3	2021.1	17.1
(1000,1500]	365 (1.7%)	4267.4	2310.0	45.9
Overall	21722	928.6	681.7	26.6

Table 4: Average wait times binned by walltime (Heterogeneous)

Average Wait Time. Table 3 shows the average wait times grouped by job size under the User(0.6) and SGS-T meta-scheduling policies. SGS-T performs 26.6% better overall and in each job size category compared to User(0.6). Similarly, Table 2 shows the average wait times grouped by wall time, where each job category benefits from SGS-T.

Average Utilization. The average utilization per week is calculated for all systems under each meta-scheduling strategy. Figure 6(a) shows the average utilization of both systems when User(0.6) is used for meta-scheduling decisions, comparing this to figure 6(b), which shows the utilization under SGS-T. At a glance, it is observable that the average utilization of both systems under SGS-T is balanced, unlike for User(0.6) as seen earlier for the homogeneous experiments. The job submission patten under User(0.6) shows that more jobs are consistently scheduled on the faster system as expected.

5 Case Study: Polaris and Theta

To further explore the effectiveness of CQSim+ for multi-resource scheduling, we conducted a case study using CQSim+ to examine the multi-resource scheduling of two systems, Theta and Polaris, from ALCF. Originally, resources on Theta and Polaris were managed independently by independent job schedulers. This is referred to as Siloed scheduling, where systems are managed independently. We formulate this simulation-based case study to show the benefits of multi-resource scheduling for the ALCF systems and evaluate CQSim+'s ability. Similar to the experiments performed earlier, we simulate the two systems using CQSim+ and their logs sourced from the ALCF public database [10]. The same time frame of 4 months, from September 1, 2023, at 00:00:00 to January 1, 2024, at 00:00:00, is utilized for the simulation.

Two instances of CQSim are run within CQSim+, the first for the Theta system consisting of 4,360 nodes and the second for Polaris with 552 nodes. Debug nodes are ignored from each system, these were 32 for Theta and 8 for Polaris. A special case for Polaris is considered where a debug job may span more than 8 nodes. Instead of ignoring those jobs from the log, we subtract 8 from their required node count. Other additional considerations include the hardware and performance of the system. The hardware difference being the GPU in Polaris and the performance difference between their CPUs. Since Theta has no GPU, all GPU jobs are always scheduled on Polaris. Where as for the CPU jobs, we scale the runtimes of the jobs by a factor. Using the peak performance metrics of each system (Polaris 44 FLOP/s and Theta 11 FLOP/s) a simple assumption is made that Polaris is 4 times faster than Theta for executing the CPU same job. CQSim+ considers this difference by scaling the runtime of the job by the factor of 4 when a Polaris job is simulated on Theta and a factor of 0.25 for the reverse.

A comparative analysis is conducted of the following resource scheduling strategies:

- *Siloed scheduling.* Under siloed scheduling, Polaris and Theta are simulated as individual instances of CQSim, simulating each system independently. This isolated simulation provides a baseline as being the original setup at ALCF.
- *User(0.6) multi-resource scheduling.* Jobs were scheduled to either Polaris or Theta using the assumption that users pick the faster system (Polaris) 60% of the time.
- *SGS-T multi-resource scheduling.* Under the SGS-T scheduling policy, each job is assigned to the system with the lowest simulated turnaround time. Additionally, SGS-T utilizes CQSim+ to simulate each job's expected performance on both systems prior to allocation, optimizing job placement based on these turnaround estimates.

The final dataset includes a total of 34,574 jobs, comprising 26,266 jobs from Polaris and 8,281 jobs from Theta. Using this data set, a data stream is simulated by feeding the jobs to CQSim+ and then recording the prediction time for each decision to measure the performance of it's design.

5.1 Results

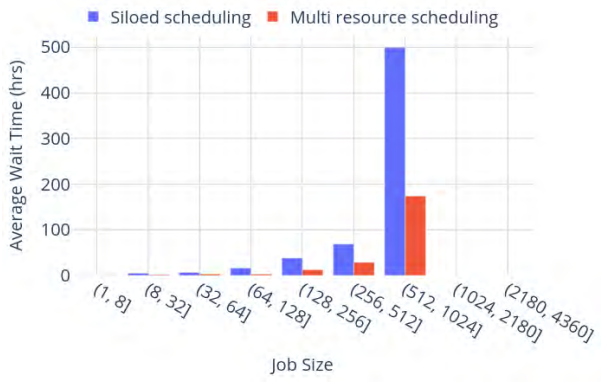
The results are discussed by analyzing key scheduling metrics including wait time, resource utilization and turnaround times. First we look at these metrics at the system level comparing the performance of each system. Then at the job level by considering job characteristics.

5.1.1 System Level Analysis. We examine the average wait times of Polaris and Theta jobs under siloed and multi-resrource scheduling. In the siloed case, each system is simulated individually with only its respective jobs, as gathered from the original logs. In the multi-resource case, CQSim+ is used for simulation-guided scheduling, where all jobs, including those from Polaris and Theta, are processed together. Overall, Polaris jobs showed a 58.7% improvement in average wait time, while Theta jobs showed a 27.6% improvement. The daily resource utilization is also analyzed for each system, showing that SGS-T peforms on par with siloed. On the other hand User(0.6) performs poorly by over utilizing the faster system (Polaris) as seen before for the faster system under the heterogeneous experiments.

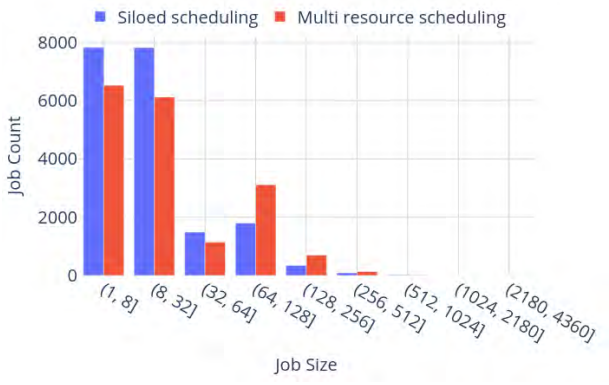
Original System	System Selected by CQSim+	Number of Jobs	Wait Time (s) (siloed)	Wait Time (s) (CQSim+)	% Impv.
Polaris	Polaris	18,439	18,707	16,898	9.7%
Theta	Polaris	4,943	10,363	5,439	47.5%
Polaris	Theta	7,827	44,061	2,107	95.2%
Theta	Theta	3,338	27,441	29,510	-7.5%

Table 5: Average wait time in seconds by original system and system selected by CQSim+.

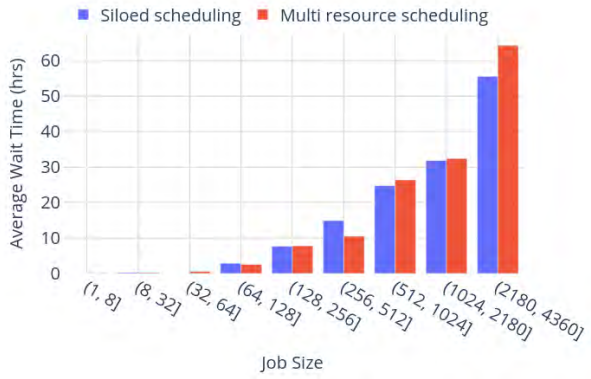
Table 5 compares the average wait time of jobs in the siloed scheduling case to that in the multi-resource scheduling case using CQSim+. It shows that 4,943 jobs originally executed on Theta were processed by Polaris, and 7,827 jobs originally executed on Polaris were processed by Theta. The largest improvement is observed for Polaris jobs placed on Theta, with a 95.2% reduction in average wait time, even with Theta processing them four times slower than Polaris. This demonstrates the clear benefits of facility-wide meta-scheduling. The results also shows that this improvement comes with a 7.5% increase in the average wait time for jobs originally assigned to Theta and processed on Theta by CQSim+.



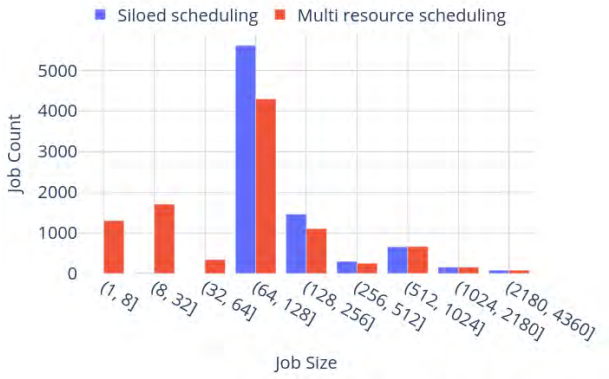
(a) Average wait time (hrs) by job size for Polaris



(b) Number of jobs processed by job size on Polaris

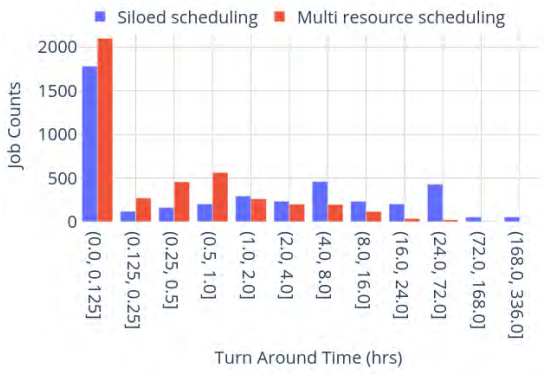


(c) Average wait time (hrs) by job size for Theta



(d) Number of jobs processed by job size on Theta

Figure 7: Comparison of average wait time and number of jobs processed on Polaris (top) and Theta (bottom).



(a) Turnarounds Time (hrs) of Polaris jobs processed by Theta

(b) Average Wait Time (hrs) of Polaris jobs processed by Theta

Figure 8: Comparison of turnaround times (left) and average wait times (right) for Polaris jobs processed by Theta.

5.1.2 Job Level Analysis. We further analyze average wait times by comparing siloed scheduling with multi-resource scheduling using CQSim+. Figure 7(a) and Figure 7(c) show the average wait time grouped by job size, while Figure 7(b) and Figure 7(d) present the number of jobs in each job size category.

By Figure 7(b) it can be seen Polaris processes fewer of the smaller 1-64 node jobs under meta-resource scheduling compared to Figure 7(d) where the amount of those same jobs increase for Theta.

As noted in the system-level analysis, the only jobs that experienced delays were those originally belonged to Theta and processed by Theta under meta-scheduling. This is further illustrated in Figure 7(b), where jobs requiring 512–4360 nodes have a longer average wait time compared to before. However, as shown in Figure 7(d), there are very few such jobs.

Since Polaris jobs run 4 times slower on Theta, analyzing the turnaround times for this set of jobs is necessary. Figure 8(a) shows the job counts grouped by their turnaround times for these jobs. It is seen that more jobs have a lesser turnaround time compared to when the jobs ran on Polaris, despite the 4x longer running time. This indicates that most of the benefit comes from shorter wait times as shown by Figure 8(b) for this same set of jobs.

5.1.3 Simulation Overhead. As a symbiotic scheduling simulator, CQSim+ must maintain a low runtime overhead to ensure seamless integration with the job scheduler. In our experiments and case study, CQSim+ achieved an average simulation cost of just 67.334 ms per job. In practice, job scheduling typically requires making decisions within 15-30 seconds [11]. Hence, this low overhead enables CQSim+ to operate effectively in real-time environments, providing timely insights for online deployment. All simulations were conducted on a personal computer employing AMD Ryzen 9 5900HX with 8 cores and 16 threads.

6 Conclusion and Future Work

This paper presents CQSim+, a symbiotic simulation framework designed for multi-resource scheduling by integrating a simulator with the physical job scheduler for dynamic, prediction-based decision-making. Through extensive experiments, CQSim+ optimizes job scheduling across homogeneous and heterogeneous systems. The results demonstrate that SGS-T by using CQSim+ consistently achieves the best average wait times and balances the load across systems. Additionally, a case study involving the Polaris and Theta systems at Argonne demonstrates that multi-scheduling with CQSim+ significantly reduces wait times — Polaris jobs processed by Theta experience a 95.2% reduction compared to siloed scheduling, and overall average wait times improve by 58.4% for Polaris and 27.5% for Theta. The real-time symbiotic aspect of CQSim+ is also tested using these experiments, which show that predictions can be made on average under 70 milliseconds.

Future work will enhance CQSim+'s predictive capabilities and expand its functionality in several directions. First, we plan to develop an extended prediction horizon, enabling the simulator to forecast beyond the immediate next job for more strategic and proactive scheduling decisions. Second, we intend to implement advanced what-if analysis by spawning a tree of processes, which will enable the exploration of more complex scenarios and the optimization of different objectives.

Acknowledgments

This work is supported in part by US National Science Foundation grant OAC-2402901 and the U.S. Department of Energy under Contract DE-SC0024271.

References

[1] Adaptive Computing Inc. 2024. *PBS-BigBook.* https://adaptivecomputing.com/moab-hpc-suite/. Accessed: 8 November 2024.
[2] William Allcock, Paul Rich, Yuping Fan, and Zhiling Lan. 2017. Experience and Practice of Batch Scheduling on Leadership Supercomputers at Argonne. In *Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP).*
[3] Altair Engineering Inc. [n. d.]. *PBS Professional.* https://altair.com/pbs-professional
[4] Altair Engineering Inc. 2022. *PBS-BigBook.* https://help.altair.com/2022.1.0/PBS%20Professional/PBS2022.1.pdf. Accessed: 8 November 2024.
[5] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N Calheiros. 2009. Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities. In *2009 international conference on high performance computing & simulation.* IEEE, 1–11.
[6] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. 2014. Versatile, scalable, and accurate simulation of distributed applications and platforms. *J. Parallel and Distrib. Comput.* 74, 10 (2014), 2899–2917.
[7] Henri Casanova, Suraj Pandey, James Oeth, Ryan Tanaka, Frédéric Suter, and Rafael Ferreira Da Silva. 2018. Wrench: A framework for simulating workflow management systems. In *2018 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS).* IEEE, 74–85.
[8] Karl Czajkowski, Ian Foster, Nick Karonis, Carl Kesselman, Stuart Martin, Warren Smith, and Steven Tuecke. 1998. A resource management architecture for metacomputing systems. In *Job Scheduling Strategies for Parallel Processing*, Dror G. Feitelson and Larry Rudolph (Eds.). Springer Berlin Heidelberg.
[9] Pierre-François Dutot, Michael Mercier, Millian Poquet, and Olivier Richard. 2017. Batsim: a realistic language-independent resources and jobs management systems simulator. In *Job Scheduling Strategies for Parallel Processing: 19th and 20th International Workshops, JSSPP 2015, Hyderabad, India, May 26, 2015 and JSSPP 2016, Chicago, IL, USA, May 27, 2016, Revised Selected Papers 19.* Springer, 178–197.
[10] Argonne Leadership Computing Facility. [n. d.]. ACLF Public Data. https://reports.alcf.anl.gov/data/index.html.
[11] Yuping Fan, Zhiling Lan, Taylor Childers, Paul Rich, William Allcock, and Michael E. Papka. 2021. Deep Reinforcement Agent for Scheduling in HPC. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS).* https://doi.org/10.1109/IPDPS49936.2021.00090
[12] Y. Fan, Z. Lan, P. Rich, W. E. Allcock, M. E. Papka, B. Austin, and D. Pau. 2019. Scheduling Beyond CPUs for HPC. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing.* Phoenix, AZ, USA, 97–108.
[13] Ian Foster and Carl Kesselman (Eds.). 1998. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
[14] Cristian Galleguillos, Zeynep Kiziltan, Alessio Netti, and Ricardo Soto. 2020. AccaSim: a customizable workload management simulator for job dispatching research in HPC systems. *Cluster Computing* 23, 1 (2020), 107–122.
[15] Michael R. Garey and Ronald L. Graham. 1975. Bounds for multiprocessor scheduling with resource constraints. *SIAM J. Comput.* (1975).
[16] Dalibor Klusáček, Mehmet Soysal, and Frédéric Suter. 2020. Alea–complex job scheduling simulator. In *Parallel Processing and Applied Mathematics: 13th International Conference, PPAM 2019, Bialystok, Poland, September 8–11, 2019, Revised Selected Papers, Part II 13.* Springer, 217–229.
[17] Arnaud Legrand, Loris Marchal, and Henri Casanova. 2003. Scheduling distributed applications: the simgrid simulation framework. In *CCGrid 2003. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2003. Proceedings.* IEEE, 138–145.
[18] William Leinberger, George Karypis, and Vipin Kumar. 1999. Job scheduling in the presence of multiple resource requirements. In *Proceedings of the 1999 ACM/IEEE Conference on Supercomputing* (Portland, Oregon, USA) *(SC '99).* Association for Computing Machinery, New York, NY, USA, 47–es. https://doi.org/10.1145/331532.331579
[19] Boyang Li, Yuping Fan, Matthew Dearing, Zhiling Lan, Paul Rich, William Allcock, and Michael Papka. 2022. MRSch: Multi-Resource Scheduling for HPC. In *2022 IEEE International Conference on Cluster Computing (CLUSTER).* 47–57. https://doi.org/10.1109/CLUSTER51413.2022.00020
[20] Weiwei Lin, Siyao Xu, Ligang He, and Jin Li. 2017. Multi-resource scheduling and power simulation for cloud computing. *Information Sciences* 397-398 (2017), 168–186. https://doi.org/10.1016/j.ins.2017.02.054

[21] A. Mu'alem and D. Feitelson. 2001. Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling. *IEEE Trans on Parallel and Distributed Systems (TPDS)* (2001).

[22] Bhakti Stephan Onggo, Navonil Mustafee, Andi Smart, Angel A Juan, and Owen Molloy. 2018. Symbiotic simulation system: Hybrid systems model meets big data analytics. In *2018 Winter Simulation Conference (WSC)*. IEEE, 1358–1369.

[23] Python Software Foundation. 2024. *Python Documentation v3.13.* https://docs.python.org/3/library/threading.html. Accessed: 8 November 2024.

[24] Gonzalo P Rodrigo, Erik Elmroth, Per-Olov Östberg, and Lavanya Ramakrishnan. 2018. Scsf: A scheduling simulation framework. In *Job Scheduling Strategies for Parallel Processing: 21st International Workshop, JSSPP 2017, Orlando, FL, USA, June 2, 2017, Revised Selected Papers 21*. Springer, 152–173.

[25] SchedMD LLC. 2022. *Slurm Plugins Documentation.* https://slurm.schedmd.com/plugins.html. Accessed: 8 November 2024.

[26] Nikolay A Simakov, Martins D Innus, Matthew D Jones, Robert L DeLeon, Joseph P White, Steven M Gallo, Abani K Patra, and Thomas R Furlani. 2018. A slurm simulator: Implementation and parametric analysis. In *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation: 8th International Workshop, PMBS 2017, Denver, CO, USA, November 13, 2017, Proceedings 8*. Springer, 197–217.

[27] SPEAR Lab. 2024. *CQSim on GitHub.* https://github.com/SPEAR-UIC/CQSim. Accessed: 8 November 2024.

[28] Hongyang Sun, Redouane Elghazi, Ana Gainaru, Guillaume Aupy, and Padma Raghavan. 2018. Scheduling parallel tasks under multiple resources: List scheduling vs. pack scheduling. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*.

[29] Talby, David, Feitelson, Dror, and Jones, James. 2024. *The Standard Workload Format.* https://www.cs.huji.ac.il/labs/parallel/workload/swf.html. Accessed: 8 November 2024.

[30] Wei Tang, Zhiling Lan, Narayan Desai, and Daniel Buettner. 2009. Fault-Aware Utility-Based Job Scheduling on Blue Gene/P Systems. In *Processing of IEEE Cluster*.

[31] Xu Yang, Zhou Zhou, Sean Wallace, Zhiling Lan, Wei Tang, Susan Coghlan, and Michael Papka. 2013. Integrating Dynamic Pricing of Electricity into Energy Aware Scheduling for HPC Systems. In *Proceedings of IEEE/ACM SC*.

[32] Andy B Yoo, Morris A Jette, and Mark Grondona. 2003. Slurm: Simple linux utility for resource management. In *Workshop on job scheduling strategies for parallel processing*. Springer, 44–60.