RESEARCH-ARTICLE

# Evaluating HPC Scheduling Strategies for Urgent Workloads

**KETAN C MAHESHWARI**, Oak Ridge National Laboratory, Oak Ridge, TN, United States

**ANDERSON BORCH**, Colorado State University, Fort Collins, CO, United States

**JORDAN WEBB**, Oak Ridge National Laboratory, Oak Ridge, TN, United States

**BRIAN D ETZ**, Oak Ridge National Laboratory, Oak Ridge, TN, United States

**ROSS MILLER**, Oak Ridge National Laboratory, Oak Ridge, TN, United States

**FRÉDÉRIC SUTER**, Oak Ridge National Laboratory, Oak Ridge, TN, United States

View all



**Published:** 16 November 2025



SC Workshops '25: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis
*November 16 - 21, 2025*
*MO, St Louis, USA*

**Conference Sponsors:**
SIGHPC


https://doi.org/10.1145/3731599.3767474
ISBN: 9798400718717

# Evaluating HPC Scheduling Strategies for Urgent Workloads

Ketan Maheshwari
Oak Ridge National Laboratory (ORNL)
Oak Ridge, USA
km0@ornl.gov

Anderson Borch
Colorado State University
Fort Collins, USA
anderson.borch@gmail.com

Jordan Webb
Oak Ridge National Laboratory (ORNL)
Oak Ridge, USA
webbjw@ornl.gov

Brian Etz
Oak Ridge National Laboratory (ORNL)
Oak Ridge, USA
etzbd@ornl.gov

Ross Miller
Oak Ridge National Laboratory (ORNL)
Oak Ridge, USA
rgmiller@ornl.gov

Frédéric Suter
Oak Ridge National Laboratory (ORNL)
Oak Ridge, USA
suterf@ornl.gov

Sarp Oral
Oak Ridge National Laboratory (ORNL)
Oak Ridge, USA
oralhs@ornl.gov

Rafael Ferreira da Silva
Oak Ridge National Laboratory (ORNL)
Oak Ridge, USA
silvarf@ornl.gov

## Abstract

Scientific computing centers increasingly face workloads with diverse urgency requirements, driven by applications that demand rapid or even immediate execution. Appropriately configured scheduling policies can significantly improve both user satisfaction and overall cluster utilization. In this work, we present a systematic analysis of scheduler configurations under scenarios where a fraction of jobs have urgent computing needs. We evaluate multiple job scheduling simulators, develop a lightweight job-submission emulation framework, and create tools to analyze and visualize the resulting scheduling data. Our study identifies key trade-offs between responsiveness, fairness, and efficiency, and offers a set of practical scheduling configurations (particularly for Slurm) that can be tailored to HPC environments supporting mixed-urgency workloads.

## CCS Concepts

• **Computing methodologies** → **Distributed computing methodologies**; **Multi-agent systems**; **Distributed computing methodologies**; **Parallel computing methodologies**.

## Keywords

HPC scheduling, Urgent Computing

**ACM Reference Format:**
Ketan Maheshwari, Anderson Borch, Jordan Webb, Brian Etz, Ross Miller, Frédéric Suter, Sarp Oral, and Rafael Ferreira da Silva. 2025. Evaluating HPC Scheduling Strategies for Urgent Workloads. In *Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC Workshops '25), November 16–21, 2025, St Louis, MO, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3731599.3767474


ACM ISBN 979-8-4007-1871-7/25/11
https://doi.org/10.1145/3731599.3767474

## 1 Introduction

Scientific computing is increasingly shaped by dynamic, data-rich, and event-triggered workflows that must respond to real-world phenomena in near real time. This shift is enabled by advances in high-speed networking, AI-enhanced orchestration, and tighter integration between experimental and computational infrastructures. As a result, high-performance computing (HPC) centers face growing demands to support workloads that require urgent, adaptive, and low-latency execution [12]. Supporting such cross-facility, experiment-in-the-loop workflows requires rethinking traditional HPC scheduling practices and infrastructure configurations to accommodate responsiveness alongside utilization and fairness [4].

Figure 1 illustrates a conceptual spectrum of urgency across application types, highlighting the diversity of Quality of Service (QoS) requirements that must be addressed within a unified scheduling framework. Applications addressing extreme events and disasters require results almost immediately, while feedback-based computing, where subsequent iterations depend on prior results, also falls on the high-urgency side. Applications such as those with reporting or publishing deadlines occupy the middle range, and exploratory analysis or software feature testing is positioned at the lower urgency end.

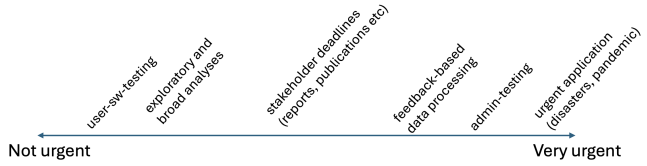


**Figure 1: A Classification of job types based on their urgency requirements.**

However, existing job schedulers and system configurations at most HPC centers are designed for batch-oriented, large-scale, and static job models. These approaches often struggle to handle workloads with urgent computing requirements such as rapid feedback loops from experiments or time-sensitive AI inference. Additionally, system administrators often lack tools that allow them to

explore and optimize scheduler configurations for urgent computing, and empirical evidence on scheduler behavior under these constraints remains limited. Current simulators frequently omit realistic user behaviors, dynamic QoS adjustments, or job-level preemption, which restricts the ability to support diverse and evolving use cases central to the United States Department of Energy (DOE) Integrated Research Infrastructure (IRI) vision [10].

Addressing these gaps requires systematic approaches to simulate, evaluate, and optimize scheduling configurations in HPC environments where urgent jobs coexist with conventional workloads [1]. Such approaches must incorporate realistic user submission patterns, QoS policies, and scheduler features such as preemption and backfilling. Furthermore, easily deployable simulation environments and tunable configuration profiles are essential for enabling reproducibility and experimentation at HPC centers aiming to support urgent computing demands.

In this paper, we present a lightweight and extensible framework for analyzing and optimizing HPC scheduling behavior in the presence of urgent computing workloads. The framework includes a job emulation environment that captures realistic user behavior and supports configurable urgency levels, enabling controlled studies of scheduler responses. We evaluate multiple job scheduling simulators, examining their capabilities and limitations in modeling QoS, job preemption, and dynamic submission patterns. To enable actionable system tuning, we design a catalog of Slurm [6, 7] configuration scenarios that reveal trade-offs between responsiveness for urgent jobs and overall system utilization. The framework also incorporates visual analytics and a taxonomy of urgency levels to help HPC centers classify job types and prioritize scheduling policies accordingly. Specifically, we make the following contributions:

(1) A comparative evaluation of four widely used job scheduling simulators, identifying their capabilities and critical limitations for modeling urgent and mixed-urgency HPC workloads.
(2) The design and implementation of an emulation-based methodology that operates on a real Slurm cluster, enabling reproducible experiments with realistic user submission patterns, urgency levels, and QoS policies.
(3) A suite of nine experimental scenarios that systematically explore combinations of QoS configurations, preemption strategies, requeueing options, and workload characteristics, allowing detailed analysis of trade-offs between responsiveness, fairness, and efficiency.
(4) Integration of visual analytics, including aggregated and per-user timelines, cluster utilization trends, and backfilling behavior, to provide actionable insights into how scheduler policies affect both urgent and normal workloads.
(5) A mapping between workload profiles and recommended scheduler configurations, resulting in a practical hybrid scheduling strategy that HPC centers can adapt to balance urgent computing needs with sustained system utilization.

## 2 Background on Job Scheduling Simulators and Their Limitations

HPC centers rarely have the flexibility to experiment with production schedulers as it could impact active user workloads. As a result, researchers often rely on job scheduling simulators to explore new policies, evaluate configuration changes, and analyze performance trade-offs in a controlled and repeatable environment. Simulation also lowers operational costs, reduces risks, and makes it possible to reproduce experimental scenarios for consistent comparison across studies. Given these advantages, we began by evaluating whether job scheduling simulators could be used as the primary environment for our study of urgent computing workloads. We sought tools that could: (1) support urgent and mixed-urgency workloads; (2) allow dynamic job arrivals and realistic user submission patterns; (3) provide QoS differentiation and urgency-specific scheduling policies; (4) model job preemption with both cancellation and requeueing; and (5) integrate with custom workload generation and analysis scripts.

We selected four representative simulators for evaluation: BatSim [3], AccaSim [5], Alea [8], and the Slurm Simulator [13]. Table 1 summarizes the key features of BatSim and AccaSim, which emerged as the most capable of the group for modeling realistic scenarios and were therefore tested more extensively.

**BatSim** is a modular and extensible simulator that separates job submission and resource management from scheduling logic. It supports workloads in both Standard Workload Format (SWF) [2] and JSON, and includes built-in tools for generating synthetic workloads and defining simulated platforms. As shown in Table 1, BatSim supports both static and dynamic job submission and can simulate static failures. We adapted BatSim with a custom workload generator that parameterizes the number of jobs, submission intervals, walltime ranges, and failure rates. This allowed us to explore a variety of job arrival patterns. However, BatSim has no native mechanisms for urgency-aware QoS levels or job preemption. Implementing these features would require significant modification of the job dispatch and scheduling components. While BatSim is flexible enough for general scheduling studies, these limitations prevent it from fully supporting our urgent computing experiments.

**AccaSim** is a Python-based simulator designed for evaluating workload management and dispatcher policies. Like BatSim, it can ingest SWF workload traces and supports workload generation and static failure simulation. Table 1 highlights several differences between the two tools. For example, AccaSim lacks dynamic job submission support, meaning that modeling on-the-fly user arrivals would require code changes. It also outputs results in SWF format and provides built-in plotting, but some output fields lack sufficient labeling, which complicates post-processing and integration with our own visualization tools. We adapted our BatSim workload generator to produce SWF files compatible with AccaSim's requirements. While this allowed us to run controlled workload scenarios, AccaSim's scheduler library is more limited in scope than BatSim's, offering fewer built-in algorithms. Like BatSim, AccaSim has no native support for urgent QoS or preemption, and extending it with these features would involve modifying core scheduling logic. Documentation is also relatively sparse, increasing the configuration

**Table 1: Comparison of Features and used values between BatSim and AccaSim**

| Simulator / Features | BatSim | AccaSim |
|---|---|---|
| Static Job Submission | Yes | Yes |
| Dynamic Job Submission | Yes | No |
| Static Failure Simulation | Yes | Yes |
| Dynamic Failure Simulation | No | No |
| Plots | No | Yes |
| Results Output Format | .csv | .swf |
| Algorithms | Filler, FCFS, Easy_BF, Conservative_BF | FIFO, SJF, LJF, Easy_BF (First Fit and Best Fit versions of all algorithms) |
| Automated Workload Generation | Yes | Yes |
| User Specified Workload Parameters | num_jobs<br>num_rs<br>walltime_range<br>res_range<br>subtime_range<br>job_failure_min<br>job_failure_max<br>critical_failure_min<br>critical_failure_max<br>time_failure_min<br>time_failure_max | –num_jobs<br>–job_id_start<br>–submit_time_start<br>–submit_time_step<br>–wait_time_default<br>–run_time_min<br>–run_time_max<br>–min_procs<br>–max_procs<br>–avg_cpu_frac<br>–mem_min_kb |
| Number of Hosts | 4 | 120 |
| Number of CPU's | 5 | 240 |
| CPU Speed (MegaHertz) | 100 | 3334 |
| CPU Wattage | 100-200 | N/A |
| Total Number of Network Links | 10 | 3D Torus Network |
| Types of Network Links | 8 | SCI Interconnect |
| Network Link Bandwidth (MBps) | 8-120 | 667 |
| Network Latency (microsec) | 1.5-500 | 1.46 |

effort needed for non-trivial experiments. Despite these limitations, AccaSim's Python-based design makes it more approachable for researchers comfortable in that language, and it remains a viable option for experiments where urgent computing features are not a primary requirement.

**Alea** is a Java-based simulator that supports conventional scheduling algorithms such as First-Come, First-Served (FCFS) and Easy Backfilling, as well as user-defined scheduling policies. Unlike BatSim and AccaSim, Alea provides a limited form of QoS through multiple job queues with priorities. However, it does not implement job preemption, which would require substantial changes to its scheduling engine. Alea uses workload logs to simulate user activity but does not support injecting new jobs dynamically at runtime. This limits its ability to reproduce highly variable and event-driven job arrival patterns that are characteristic of urgent computing scenarios. While Alea can generate simulation results and basic visualizations, we did not develop custom plotting for it as part of this study. We engaged in discussions with Alea's primary developer regarding the feasibility of adding urgent computing features such as preemption and runtime workload adjustments. While technically possible, these enhancements were not present in the current release and would require non-trivial development.

**The Slurm Simulator** aims to reproduce actual Slurm scheduling behavior in a controlled environment, making it a potentially valuable tool for studies that require fidelity to production HPC configurations. It is designed to run inside a Docker container, which simplifies setup in compatible environments but presents challenges on systems where containerized execution is restricted for security or policy reasons. In our case, institutional constraints on Docker usage prevented a full evaluation. The simulator can be run outside Docker, but doing so requires additional configuration steps that are prone to errors and can be time-consuming. Additionally, we found the currently available version of Slurm on the simulator was much older (20.11.9) than the one used on our systems at OLCF (25.05).

Across all four simulators, we identified one or more of the following gaps that limited their ability to support our urgent computing experiments: (1) no native support for urgent QoS or urgency-aware scheduling; (2) limited or no implementation of job preemption with realistic handling of cancellation or requeueing; (3) incomplete modeling of dynamic user behavior and event-driven job arrivals; and / or (4) practical deployment barriers in our testbed environment.

Even with extensive customization, these tools could not fully reproduce the operational conditions of a real HPC scheduler under urgent computing demands. This assessment led us to adopt an emulation-based methodology that retains the control and repeatability of synthetic workloads while operating in a real Slurm environment. This approach is described in the next section.

## 3 Emulation-based Methodology

While job scheduling simulators offer a controlled and repeatable environment for testing, they exhibit critical limitations for our research goals, particularly their inability to capture the complex, real-world dynamics and operational effects of urgent computing. These limitations reduce their value for evaluating scenarios where responsiveness, fairness, and utilization must be balanced under realistic conditions. That said, we acknowledge that simulators remain valuable for prototyping policies, exploring large parameter spaces, and conducting safe, repeatable experiments when production systems are inaccessible.

To address the limitations identified in Section 2, an alternate approach that that preserves experimental control while operating within a real HPC scheduling environment is to deploy a full Slurm installation. This approach was followed by the RM-Replay tool [9], which implements a faster system clock to speed up the emulation of the job and resource management of Slurm. However, this tool is limited to the replay of job submission traces and thus cannot handle the dynamic submission of urgent jobs.

In our case, we had a unique opportunity to experiment on a real Slurm cluster shortly before its deployment, which allowed us to implement an emulation-based methodology and capture authentic scheduler behavior without impacting production workloads. Specifically, we deploy a full, up to date Slurm installation on a dedicated test cluster and emulate workloads that reflect diverse, realistic submission patterns and urgency levels. This hybrid approach combines the flexibility of synthetic workloads with the fidelity of a production-grade scheduler, ensuring that queue dynamics, scheduling decisions, and preemption behaviors mirror those in operational HPC systems.

In our framework, urgent workloads are defined as jobs that must begin execution immediately or within a short time window. Such workloads introduce trade-offs between responsiveness, fairness, and system utilization. Emulating these workloads on a real Slurm instance allows us to directly measure the operational impacts of preemption and QoS adjustments without relying on the abstractions or omissions inherent in simulation tools. QoS settings play a central role in our study, serving as a mechanism to differentiate urgency levels in the scheduler. For example, a "high" QoS can be configured to preempt lower priority jobs, whereas a Slurm configuration may allow preferential scheduling of the same QoS without preemption. We explore both configurations, along with scenarios where preempted jobs are either requeued or canceled. Because the environment is a functioning Slurm cluster, these variations reflect the real configuration knobs available to HPC system administrators.

Our workload emulation framework uses a stochastic job submission model to mimic realistic user activity. Multiple independent "users" submit jobs at random intervals with varying sizes, walltimes, and resource requirements. Between 5% and 10% of these jobs are designated as urgent. For execution, we employ a lightweight MPI-based mock application that minimally consumes CPU and I/O but fully occupies allocated resources, ensuring that observed outcomes are attributable to scheduling policy differences rather than application performance effects.

To implement this model, we create a set of dummy users and a driver script that submits a number of jobs on their behalf. Each job is assigned randomized parameters such as walltime, node count, and QoS designation within predefined limits. Submissions occur at stochastic intervals to ensure variability in queue load and job overlap. This design enables us to capture the interplay between user activity, scheduler policies, and urgency-aware configurations under realistic but controlled conditions.

This shift from simulation to emulation provides a more faithful reproduction of HPC scheduling dynamics while preserving experimental repeatability. It enables us to evaluate urgent computing strategies under realistic conditions, measure system-level impacts such as wasted node minutes, and generate results that are directly actionable for HPC centers considering changes to their production Slurm configurations.

## 4 Experimental Evaluation

Building on the emulation-based methodology described in Section 3, we designed a set of experiments to evaluate scheduler configurations for urgent computing in a real Slurm environment. Our experiments use synthetic workloads that emulate realistic user submission patterns and a mix of urgency levels. Each experiment was configured to test specific combinations of QoS settings, preemption policies, and workload characteristics in order to measure their impact on metrics such as job wait times, wasted node minutes, and overall system utilization.

### 4.1 Experimental Platform

The Advanced Computing Ecosystem (ACE) testbed [11], located at the Oak Ridge Leadership Computing Facility (OLCF), provides a flexible platform for evaluating emerging HPC technologies, testing scheduler configurations, and experimenting with novel software solutions in a controlled environment. ACE is designed to support research and development activities that prepare applications, workflows, and system tools for deployment on future large-scale production systems.

For this study, ACE offered the ideal environment to implement our emulation-based methodology. The system supports a full production-grade Slurm installation, enabling realistic job scheduling and queue management while allowing us to control workload composition and submission patterns. The flagship system within ACE, Defiant-2, includes 20 CPU-only Intel nodes and 2 GPU nodes equipped with 8 NVIDIA H200 GPUs each. This mix of resources allows experiments to be conducted under conditions that closely reflect those of large HPC centers, while retaining the flexibility to reconfigure and reset the environment between experiments.

Using ACE allowed us to run repeated trials with different QoS, preemption, and backfilling configurations without impacting production workloads. At the same time, we were able to capture the full fidelity of Slurm's scheduling behavior on real hardware, ensuring that results accurately reflected operational conditions in HPC environments.

### 4.2 Experimental Scenarios

We designed nine experimental scenarios to systematically evaluate scheduler behavior in the presence of mixed-urgency workloads,

with a focus on user-level outcomes and overall system utilization. In each experiment, approximately 10% of jobs are designated as urgent, unless otherwise stated. For all scenarios, backfilling is enabled, which allows the scheduler to start lower-priority jobs earlier when they fit available node and time constraints.

To drive the experiments, we implement an automated orchestration environment using `tmux`, with 22 panes: 20 panes for concurrent job submissions from independent dummy users, one pane for monitoring cluster utilization, and one pane for controlling the overall setup. Our job generation script stochastically selects parameters such as the total number of jobs, walltime, node count, QoS level, and mock application runtime within predefined bounds. Jobs are submitted at randomized intervals to simulate realistic queue dynamics. A mock application, developed using MPI, performs minimal computation and I/O while fully occupying allocated resources, ensuring that observed effects are attributable to scheduling policies rather than application characteristics. Each job uses 64 ranks per node to occupy all CPUs on the allocated nodes.

The nine experimental configurations are summarized in Table 2:

- **E1 – Baseline calibration:** Initial environment validation on a newly deployed cluster. Twenty users submit jobs of varying sizes (1–20 nodes) and walltimes (up to 60 minutes) with no preemption. Application runtimes are set to avoid timeouts.
- **E2 – Timeout calibration:** Similar to E1 but with walltimes capped at 15 minutes and timeouts induced for ∼50% of jobs to test how the scheduler handles failed jobs. The experiment runs up to 1,400 minutes to ensure the queue is cleared of all the submitted jobs.
- **E3 – Preemptive urgent jobs (no requeue):** Fixed 20 jobs per user of varying sizes (1–10 nodes) and walltimes (3 to 15 minutes), with 5–10% urgent jobs (high QoS) capable of preempting normal jobs. Preempted jobs are canceled and not requeued.
- **E4 – All normal jobs baseline:** Fixed 20 jobs per user, all normal QoS, serving as a reference for other fixed-size scenarios.
- **E5 – Non-preemptive urgent jobs:** Same as E3, but urgent jobs received highest scheduling priority without preemption.
- **E6 – Preemptive urgent jobs (with requeue):** Similar to E3, but preempted jobs are requeued instead of being canceled.
- **E7 – Short jobs with requeue:** Fixed 10-minute walltimes, actual runtimes 10–20% of walltime, urgent jobs preempt with requeue. Designed to measure lost opportunity cost when short jobs are overestimated and interrupted.
- **E8 – Short jobs without requeue:** Same as E7 but preempted jobs are not requeued.
- **E9 – Long jobs without requeue:** Fixed 10-minute walltimes, actual runtimes 80–90% of walltime, urgent jobs preempt without requeue to study wasted resources near completion.

In all scenarios, QoS classes are used to model urgency. For example, a "high" QoS is configured to immediately allocate resources by preempting lower-priority jobs, while other QoS classes provide extended walltimes or scheduling priority boosts without preemption. All other jobs that have a "normal" QoS do not have any ability to influence other jobs. These configurations reflect realistic mechanisms available in production Slurm deployments and enable direct translation of our findings to HPC operational environments.

| Exp | Urgent Jobs | Urgent QoS | Requeue | Runtime |
|---|---|---|---|---|
| E1 | none | none | none | <walltime |
| E2 | none | none | none | walltime+ |
| E3 | 5-10% | preemption | none | walltime+ |
| E4 | none | none | none | walltime+ |
| E5 | 5-10% | high priority | none | walltime+ |
| E6 | 5-10% | preemption | enabled | walltime+ |
| E7 | 5-10% | preemption | enabled | 10-20% of w/t |
| E8 | 5-10% | preemption | none | 10-20% of w/t |
| E9 | 5-10% | preemption | none | 80-90% of w/t |

**Table 2: A Summary of Experiments Configuration. For each experiment, there were 20 users each submitting jobs starting at the same time but at randomized intervals. A random number of jobs were submitted by users for the first two experiments but for the rest, each user submitted 20 jobs. Walltime+ for Runtime refers to a config where jobs were made to run over walltime to induce timeouts.**

## 5 Results and Discussion

Figure 2 presents an aggregated view of all jobs submitted across the nine configurations, showing how each scheduler policy influenced job end states and average wait times. Each bar segment represents a specific end state for jobs, including completion, cancellation, preemption, timeout, or failure. The overlaid data highlight the average wait times before execution. This high-level visualization makes it immediately clear how different scheduler configurations affect both job turnaround and system responsiveness. The differences between configurations with and without preemption are evident. Preemptive setups generally reduce wait times for high-priority jobs, although the effect is not uniform across all scenarios. Control experiments such as `E1`, `E2`, and `E4`, which lack QoS differentiation or preemption, consistently exhibit the longest wait times, confirming that prioritization mechanisms are essential for supporting urgent jobs.

### 5.1 User-Centric Views and Data Availability

User-centric visualizations, such as the swimlane representation in Figure 3, provide an in-depth view of job execution over time from the perspective of individual users. These visualizations allow tracking of start times, execution phases, and preemptions for each user's jobs. They also reveal how identical configurations can produce very different outcomes depending on the timing of job submission, the size of the requested allocation, and the overall system load. For example, a user submitting an urgent job during low contention may experience near-instant execution, while another submitting during peak demand may still face delays even under a preemptive policy. This per-user view also shows that multiple high QoS jobs arriving at the same time can saturate the cluster, resulting in delays for urgent workloads despite preemption being enabled. The full set of interactive user-centric diagrams is available in our GitHub repository[1], enabling deeper exploration of queue dynamics and execution timelines.

[1] https://github.com/ketancmaheshwari/wiuhpc25

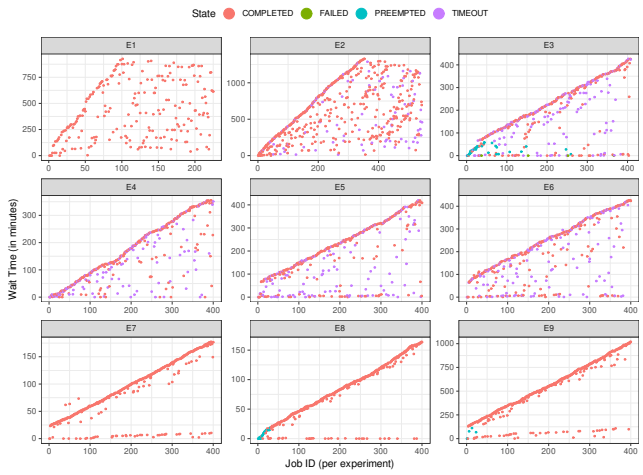


**Figure 2: Overall distribution of job end states and average wait times across all nine experimental configurations. The figure highlights how different scheduler policies influence the proportion of completed, canceled, preempted, and failed jobs, as well as the responsiveness for both high-priority and normal-priority workloads.**

In Figure 3, we present the swimlane chart for E3, where 5-10% of jobs are urgent with high QoS and the ability to preempt normal QoS jobs. This experiment represents how an HPC facility may try to support time-sensitive workflows alongside more traditional workflows and provides a perspective on how complex HPC scheduling can become. Jobs can complete normally (green) or timeout (purple) due to users not providing an appropriate wall time. With high QoS enabled, these jobs (designated with gold diamonds) receive a higher priority and can preempt (blue) running normal QoS jobs. Optimizing this complex landscape is important for maximizing system utilization, keeping queue wait times reasonable, and supporting time-sensitive workflows.

The Slurm configuration of E3 shown in Figure 3 keeps wait times for high QoS jobs low, on average 1.5 minutes, which would support most time-sensitive workflows, but this comes at the cost of preempting running jobs that amount to ~258 node minutes of wasted resources (Table 3). Preemption results in canceling a user's job which is not automatically requeued (in E3). Therefore, this configuration could adversely impact users' ability to use the resource without having high QoS. For example, the average wait time for normal QoS is ~200 minutes, and if a user's job is preempted this would further extend the wait time to much longer time scales.

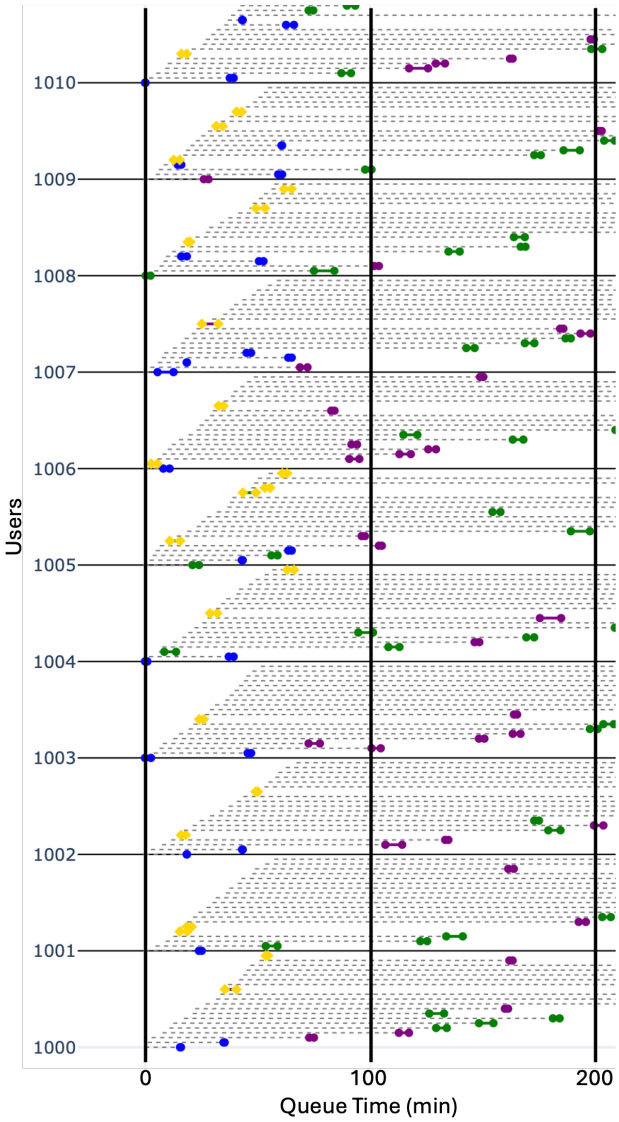


**Figure 3: Swimlane visualization for a representative experiment (E3), showing job timelines for individual users. The plot illustrates variations in wait times (gray dashed lines), execution phases (green for completed, purple for timeout, and gold for high QoS), and preemption events (blue lines and dots) across the experiment for many users.**

## 5.2 Cluster Utilization Analysis

Figure 4 shows the evolution of cluster utilization for three representative experiments with different Slurm configurations and workload characteristics, illustrating the interplay between job shape (node count and run time), arrivals, completions, and preemptions. Similar results were generated for all experiments and are also available in our GitHub repository. In scenarios with frequent preemptions such as E3, utilization profiles exhibit more pronounced fluctuations, reflecting the workload disruptions and rescheduling, while slightly increasing the overall system utilization compared to non-preemption scenarios. In contrast, non-preemptive configurations display steadier utilization but may increase urgent job wait times. However, these generalizations are highly dependent on the system and the job profiles (node count and runtime). E9, which involves preemption and contains jobs with runtimes 80–90% of walltime, maintains steady utilization due to the lower turnover of running jobs. Table 3 reports the average system utilization for

| Exp Name | Percent System Utilization ($\sigma$) | Normal QoS Wait Time | High QoS Wait Time | Wasted Node Minutes |
|---|---|---|---|---|
| E1 | 97.7 (6.9) | 430.7 | – | – |
| E2 | 92.8 (15.8) | 597.0 | – | – |
| E3 | 92.0 (11.0) | 198.2 | 1.5 | 257.7 |
| E4 | 90.3 (13.5) | 152.6 | – | – |
| E5 | 91.0 (13.5) | 206.8 | 4.5 | – |
| E6 | 90.9 (13.0) | 220.9 | 5.6 | undetermined |
| E7 | 86.8 (15.0) | 97.2 | 5.0 | undetermined |
| E8 | 87.8 (14.6) | 83.6 | 0.2 | 66.1 |
| E9 | 91.4 (10.8) | 559.4 | 48.9 | 78.1 |

**Table 3: Average system utilization (std dev), wait times (minutes) for normal and high QoS jobs, and total wasted node minutes for jobs that are preempted across the experiments. E6 and E7 contain undetermined wasted resources due to how Slurm categorizes requeued jobs as 'Completed' rather than 'Preempted'.**

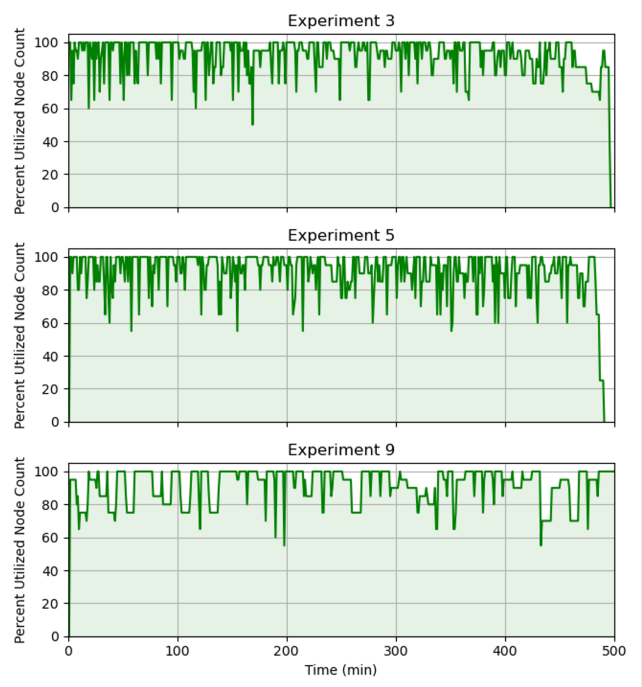


**Figure 4: Cluster utilization over time for three representative experiments with different job submission patterns and Slurm settings. The figure highlights how job arrivals, completions, and preemptions shape system occupancy patterns under the tested scheduler configuration.**

each experiment and Figure 5 presents additional statistics for system utilization across each experiment. All experiments maintain relatively high system utilization across the Slurm parameters evaluated, with a slight decrease observed for E7, and E8. As shown in Figure 5, E7 and E8 have a larger distribution of system utilization below 60% than all other experiments. These drops may be due to both experiments using jobs with runtimes scaled to 10-20% of walltime, reducing sustained load on the system.

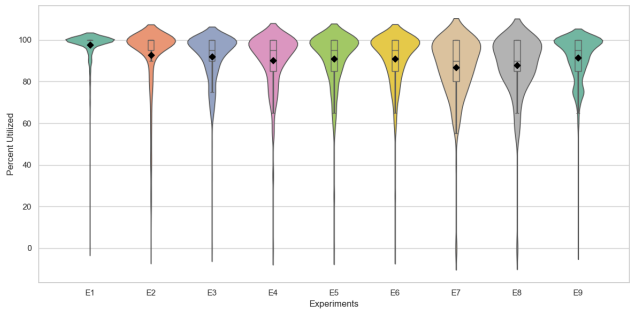


**Figure 5: Statistical performance visualization for each experiment: the black diamond represents mean percent utilization, the box shows median and quartile range. The distribution on data is captured in violin shape.**

One strategy most HPC schedulers employ to improve overall system utilization is backfill scheduling. All experiments in this assessment enable Slurm's backfill scheduling to allow the scheduler to start lower-priority jobs earlier when they fit available node and time constraints. Generally, backfilling benefits jobs that request smaller node counts and shorter wall times that can easily be fit into open slots on the resource. Figure 6 further supports this generalization as the experiments performed in this study demonstrate that larger jobs have a markedly lower probability of being backfilled and that backfill opportunities shrink as job size increases. Observations across the experiments indicate that the scheduler does not appear to backfill purely based on job size; instead, it conservatively considers the priority rank assigned to jobs, which can reduce the chance of filling idle gaps with lower-priority work. E5 is particularly notable in this regard: by granting high priority to urgent jobs without preemption, it achieved steady utilization and competitive urgent job wait times, suggesting that priority adjustments can provide a balance between responsiveness and efficiency, without the need for preemption.

## 5.3 Observations from Experiments

Table 3 summarizes system utilization, average wait times for normal and high QoS jobs, and wasted node minutes due to preemptions. The results show that preemption can substantially reduce urgent job wait times, as in E3, E6, E7, and E8, but does not guarantee minimal delays under high contention scenarios. E9 stands out as a case where average urgent job wait time reached 49 minutes despite preemption being enabled, underscoring the challenge of handling concurrent bursts of urgent jobs. This behavior can be observed in Figure 7, in which the system is fully saturated with high QoS jobs at the onset of the experiment (only high QoS jobs are run in the first 150 minutes), resulting in long wait times for all jobs regardless of priority.

Enabling preemption comes at a cost of potentially wasting resources as shown for E3, E8, and E9. The wasted node minutes presented in Table 3 are the cumulative elapsed time that jobs ran

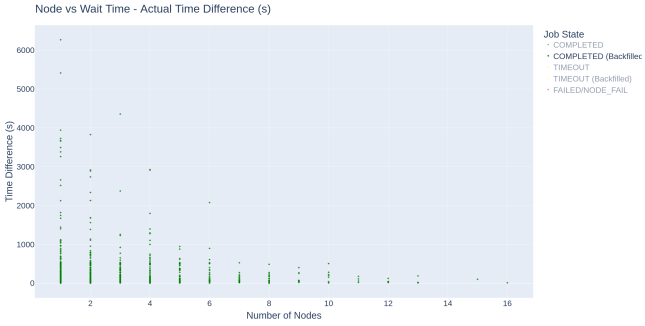


**Figure 6: Relationship between job size and the difference between wait time and execution time for backfilled jobs across all experiments. The plot shows that larger jobs are less likely to be backfilled and emphasizes the impact of job size on scheduling opportunities.**

prior to being preempted by high QoS jobs. Wasted resources for E8 and E9 are much lower than E3 due to how the experiments were setup and the timing of high QoS job submission. In E8, the runtime of jobs is scaled to 10-20% of the wall time meaning jobs have a higher likelihood of finishing before preemption can occur and jobs that are preempted likely were not running for very long lowering the cumulative wasted node minutes. In E9, high QoS jobs saturate the available resources and the queue, and have long run times (80-90% of the submitted wall time), meaning that only a few normal QoS jobs get preempted (two shown with blue circles in Figure 7). E3 represents a more general view of how costly preemption can be as job run times and high QoS job submission are more randomly distributed. Wasted resources could be improved if users have incorporated sufficient check-pointing to aid job restarts, however this depends on user computing proficiency and is not incorporated into this analysis. Although the overall wasted resource may seem minimal in this analysis, when scaled to full-size production HPC facilities, the amount of wasted resources due to preemption would be significant. Conversely, E5, which relied solely on priority escalation, maintained low urgent job wait times without incurring any wasted compute time.

For E6 and E7, the amount of wasted resources cannot be determined as Slurm automatically requeues any preempted jobs and classifies those jobs as completed as opposed to preempted in the Slurm logs. However, we can still assess the number of jobs that were requeued automatically by Slurm to get a general idea of the preemption. In E6, 11 jobs are requeued due to preemption with, on average, 10.6 node minutes being wasted per job. This value is derived from the elapsed time of the requeued job and corresponds to a worst-case scenario as there is no way to determine when jobs were preempted. In E7, only three jobs are requeued due to preemption, resulting in a maximum wasted resource of 18.9 node minutes. This low number of preemptions is likely due to the scaled run times (10-20% of the wall time) and the system getting saturated with high QoS jobs lowering the occurrence of preemption.

The experiments also confirm that both short and long jobs suffer under preemption but in different ways. Short-running jobs preempted early lose the opportunity for quick completion, representing a missed opportunity with little gain in urgency; long-running jobs preempted near completion waste significant consumed resources and frustrate user expectations. E8 and E9 illustrate these extremes, with 21 short jobs preempted in the former and only five long jobs preempted in the latter.

Overall, the combination of Figures 2–6 and Table 3 demonstrates that responsiveness, efficiency, and fairness cannot be optimized simultaneously with a single static configuration.

## 5.4 Towards a Hybrid Scheduling Strategy

When considering all experimental results and the visualizations, it becomes clear that no single configuration is optimal for all workloads. Instead, the configurations tested here can be aligned with specific points on the urgency spectrum (as introduced in Figure 1), producing a practical mapping between workload characteristics and scheduling policy. For example, high QoS with preemption is well suited to urgent applications or administrative testing; high QoS without preemption fits feedback-driven data processing or stakeholder deadlines; normal QoS with preemption and requeueing supports exploratory or broad analyses; and normal QoS with preemption but no requeueing is appropriate for software testing scenarios.

Urgent short jobs are best served by preemptive high QoS policies, where the small execution time minimizes waste. Urgent long jobs benefit more from high QoS priority boosts without preemption, balancing faster starts with resource preservation. Exploratory normal QoS jobs can be run with preemption and requeueing enabled, allowing them to yield to urgent work while still completing eventually. Large-scale production jobs should be protected from preemption to safeguard efficiency, while feedback-driven workloads may require non-preemptive high priority to ensure regular updates. Testing and validation runs can be placed in preemptible queues to free capacity for critical work when needed.

Table 4 summarizes these mappings between job profiles and recommended configurations. Adopting a policy that applies these mappings dynamically enables HPC centers to accommodate urgent computing needs without disproportionately affecting other workloads, maintaining both system responsiveness and high overall utilization.

## 6 Conclusions and Future Work

In this work, we conducted a preliminary evaluation of scheduler configurations for HPC environments that must accommodate urgent computing workloads alongside traditional jobs. Using a stochastically-driven emulation framework on a real Slurm cluster, we examined nine configurations combining QoS policies, preemption strategies, requeueing options, and workload profiles. Our results show that the balance between user responsiveness and overall system utilization depends strongly on workload characteristics and scheduler configuration. Preemption can dramatically reduce urgent job wait times but at the cost of wasted compute time, particularly for long jobs. Conversely, priority adjustments without preemption can achieve competitive responsiveness with

| Job Profile | Recommended Configuration | Rationale |
|---|---|---|
| Urgent, short runtime (< 15 min) | High QoS with preemption | Maximizes responsiveness with minimal wasted compute time. |
| Urgent, long runtime (> 1 hr) | High QoS priority boost without preemption | Reduces wait time while avoiding costly interruptions. |
| Normal priority, exploratory | Normal QoS with possible preemption and requeue | Balances availability for urgent jobs with continued progress for non-critical work. |
| Normal priority, production-scale | Normal QoS without preemption | Protects long-running, resource-intensive jobs from disruption. |
| Feedback-driven workloads | High QoS priority boost without preemption | Enables timely updates while preserving stability. |
| Testing and validation runs | Normal QoS with preemption allowed | Facilitates rapid turnover to make room for critical work. |

**Table 4: Recommended scheduler configurations for different job profiles.**

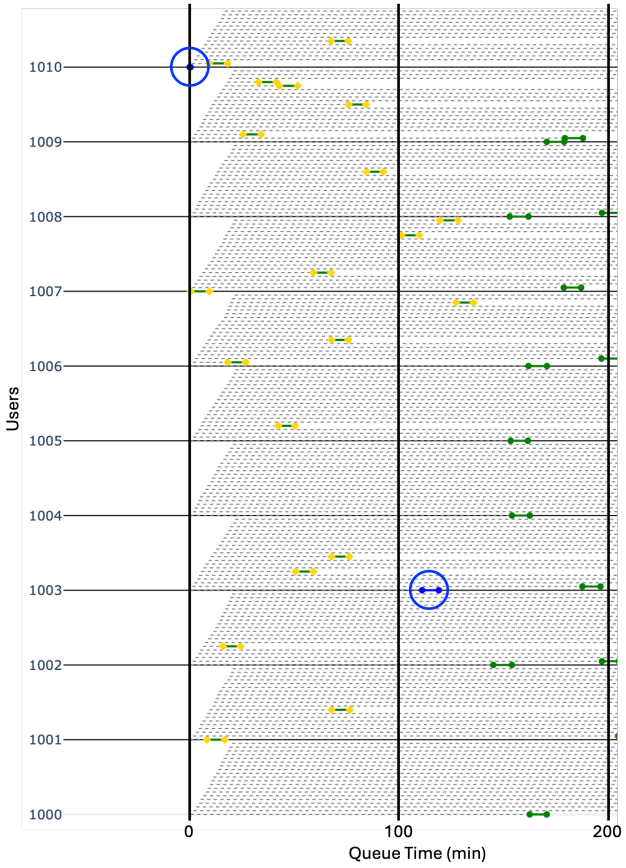


**Figure 7: Swimlane plot for E9 depicting how long queue wait times may occur for urgent jobs if the system is saturated with high QoS jobs.**

no wasted resources but may still lead to delays under heavy contention. Backfilling remains beneficial for small jobs, although its impact is constrained by job priority settings and system load. A key outcome of this study is the development of a hybrid scheduling strategy that maps workload profiles to specific scheduler configurations. This mapping provides a practical framework for HPC centers to dynamically adapt their policies, ensuring urgent computing support without disproportionately affecting other workloads.

Future work will expand this study in several directions. First, we plan to integrate workload traces from production systems to validate the hybrid strategy under real user behavior patterns and bursty arrival conditions. Second, we aim to evaluate additional scheduler features such as job aging, QoS-based resource reservations, and adaptive priority adjustments informed by machine learning models. Third, we intend to explore cross-facility urgent computing scenarios, where scheduling decisions must consider resource availability and priority policies across multiple HPC sites. Finally, we will develop and release a publicly available toolkit that automates the emulation process and provides interactive visual analytics, enabling broader adoption by HPC centers seeking to test and refine their urgent computing policies.

## Acknowledgments

This research used resources of the OLCF at ORNL, which is supported by DOE's Office of Science under Contract No. DE-AC05-00OR22725.

## References

[1] Chansup Byun, Jeremy Kepner, William Arcand, David Bestor, Bill Bergeron, Vijay Gadepally, Michael Houle, Matthew Hubbell, Michael Jones, Andrew Kirby, Anna Klein, Peter Michaleas, Lauren Milechin, Julie Mullen, Andrew Prout, Antonio Rosa, Siddharth Samsi, Charles Yee, and Albert Reuther. 2020. Best of Both Worlds: High Performance Interactive and Batch Launching. In *2020 IEEE High Performance Extreme Computing Conference (HPEC)*. 1–7. doi:10.1109/HPEC43674.2020.9286142

[2] Steve J. Chapin, Walfredo Cirne, Dror G. Feitelson, James Patton Jones, Scott T. Leutenegger, Uwe Schwiegelshohn, Warren Smith, and David Talby. 1999. Benchmarks and Standards for the Evaluation of Parallel Job Schedulers. In *Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing*, Dror G. Feitelson and Larry Rudolph (Eds.). Springer, San Juan, Puerto Rico, 67–90. doi:10.1007/3-540-47954-6_4

[3] Pierre-François Dutot, Michael Mercier, Millian Poquet, and Olivier Richard. 2016. Batsim: a Realistic Language-Independent Resources and Jobs Management Systems Simulator. In *Proceedings of the 20th Workshop on Job Scheduling Strategies for Parallel Processing (Lecture Notes in Computer Science, Vol. 10353)*. Springer, Chicago, IL, 178 —- 197. doi:10.1007/978-3-319-61756-5_10

[4] Rafael Ferreira da Silva, Rosa M. Badia, Deborah Bard, Ian T. Foster, Shantenu Jha, and Frédéric Suter. 2024. Frontiers in Scientific Workflows: Pervasive Integration with HPC. *IEEE Computer* 57, 8 (2024). doi:10.1109/MC.2024.3401542

[5] 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 (March 2020), 107–122. doi:10.1007/s10586-019-02905-5

[6] M Jette, C Dunlap, J Garlick, and M Grondona. 2002. SLURM: Simple Linux Utility for Resource Management. (07 2002). https://www.osti.gov/biblio/15002962

[7] Morris A. Jette and Tim Wickberg. 2023. Architecture of the Slurm Workload Manager. In *Job Scheduling Strategies for Parallel Processing: 26th Workshop, JSSPP 2023, St. Petersburg, FL, USA, May 19, 2023, Revised Selected Papers* (St. Petersburg,

FL, USA). Springer-Verlag, Berlin, Heidelberg, 3–23. doi:10.1007/978-3-031-43943-8_1

[8] Dalibor Klusacek, Mehmet Soysal, and Frédéric Suter. 2019. Alea - Complex Job Scheduling Simulator. In *Proceedings of the 13th International Conference on Parallel Processing and Applied Mathematics (Lecture Notes in Computer Science, Vol. 12044)*. Springer, Bialystok, Poland, 217 – 229. doi:10.1007/978-3-030-43222-5_19

[9] Maxime Martinasso, Miguel Gila, Mauro Bianco, Sadaf R. Alam, Colin McMurtrie, and Thomas C. Schulthess. 2018. RM-Replay: A High-Fidelity Tuning, Optimization and Exploration Tool for Resource Management. In *Proceedings of the 2018 International Conference for High Performance Computing, Networking, Storage and Analysis*. 320–332. doi:10.1109/SC.2018.00028

[10] William L Miller, Deborah Bard, Amber Boehnlein, Kjiersten Fagnan, Chin Guok, Eric Lançon, Sreeranjani Jini Ramprakash, Mallikarjun Shankar, Nicholas Schwarz, and Benjamin L Brown. 2023. *Integrated research infrastructure architecture blueprint activity (final report 2023).* Technical Report. US Department of Energy (USDOE).

[11] Sarp Oral, Rafael Ferreira Da Silva, Subil Abraham, Ryan Adamson, Valentine Anantharaj, Tom Beck, Ashley Barker, Katie Bethea, Joshua Brown, Michael Brim, et al. 2024. *OLCF's Advanced Computing Ecosystem (ACE): FY24 Efforts for the DOE Integrated Research Infrastructure (IRI) Program.* Technical Report. Oak Ridge National Laboratory (ORNL), Oak Ridge, TN (United States). doi:10.2172/2477506

[12] Albert Reuther, Nick Brown, William Arndt, Johannes Blaschke, Christian Boehme, Antony Chazapis, Bjoern Enders, Robert Henschel, Julian Kunkel, and Maxime Martinasso. 2024. Interactive and Urgent HPC: Challenges and Opportunities. arXiv:2401.14550 [cs.DC] https://arxiv.org/abs/2401.14550

[13] Nikolay A. Simakov, Robert L. DeLeon, Martins D. Innus, Matthew D. Jones, Joseph P. White, Steven M. Gallo, Abani K. Patra, and Thomas R. Furlani. 2018. Slurm Simulator: Improving Slurm Scheduler Performance on Large HPC systems by Utilization of Multiple Controllers and Node Sharing. In *Proceedings of the Practice and Experience on Advanced Research Computing: Seamless Creativity*. ACM, Pittsburgh, PA, Article 25, 8 pages. doi:10.1145/3219104.3219111