Proceedings of the 2024 Winter Simulation Conference
H. Lam, E. Azar, D. Batur, S. Gao, W. Xie, S. R. Hunter, and M. D. Rossetti, eds.

SCALABLE HPC JOB SCHEDULING AND RESOURCE MANAGEMENT IN SST

Abubeker Abdurahman[1], Abrar Hossain[1], Kevin A. Brown[2], Kazutomo Yoshii[2], and Kishwar Ahmed[1]

[1] Dept of Electrical Engineering and Computer Science, The University of Toledo, Toledo, OH, USA
[2] Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL, USA

ABSTRACT

Efficient job scheduling and resource management contributes towards system throughput and efficiency maximization in high-performance computing (HPC) systems. In this paper, we introduce a scalable job scheduling and resource management component within the structural simulation toolkit (SST), a cycle-accurate and parallel discrete-event simulator. Our proposed simulator includes state-of-the-art job scheduling algorithms and resource management techniques. Additionally, it introduces a workflow management components that supports the simulation of task dependencies and resource allocations, crucial for workflows typical in scientific computing and data-intensive applications. We present validation and scalability results of our job scheduling simulator. Simulation shows that our simulator achieves good accuracy in various metrics (e.g., job wait times, number of nodes usage) and also achieves good parallel performance.

1 INTRODUCTION

High-performance computing (HPC) systems have witnessed unprecedented growth in recent years, not only in the sheer system size but also in their architectural complexity and their sophisticated HPC application processing capabilities. Many advanced job scheduling and resource management techniques have been developed to achieve desired performance objectives of complex scientific applications running on large HPC systems. These techniques aim to increase both resource utilization and job turnaround times, thereby justifying the increasing resource demands, ranging from compute nodes and memory to power, of next-generation HPC systems. With increase in larger and computationally-demanding applications (e.g., machine learning workloads), it is important that we have efficient scheduling and resource management evaluation techniques in HPC (Soysal and Streit 2021).

Simulation emerges as an effective tool in evaluating the efficiency of job scheduling and resource management techniques as it offers insights into these techniques' performance implications on overall HPC system's performance and utilization. A number of job scheduling simulators exist. For example, CQsim (SPEAR Lab. 2023), Alea 2 (Klusáček and Rudová 2010), Slurm Simulator (Simakov et al. 2017), and GridSim (Buyya and Murshed 2002) are some of the existing job scheduling simulators. CQsim excels in event-driven job scheduling with its diverse input options. Alea 2 supports most of the common scheduling algorithms and includes an intuitive visualization interface. The Slurm simulator enhances modeling fidelity by mirroring the Slurm resource manager's (Yoo et al. 2003) functionality without taxing the actual system performance. GridSim's Java-based framework adeptly models heterogeneous grid environments, focusing on the distributed management of resources. However, despite their respective strengths, common shortfalls among these simulators are their limited scalability and the absence of comprehensive scheduling algorithms essential for simulating the dynamic HPC job behavior. We propose an HPC job scheduling simulator that integrates state-of-the-art HPC job scheduling algorithms, and achieves scalability and high fidelity on simulating HPC jobs, resources, and workflows.

The structural simulation toolkit (SST) is a well-known architecture and network simulator that supports both cycle-accurate and parallel discrete-event based simulation (Rodrigues et al. 2011). SST's scalability is underscored by its modular design. SST contains three libraries – `SST Core`, `SST Elements`, and `SST Macros`. The `SST Core` facilitates the fundamental simulation framework. The `SST Elements` provides the building blocks for modeling complex systems. The `SST Macros` facilitates scalable and large-scale system simulations. SST offers parallel execution and scalability by employing conservative synchronization strategy (Rodrigues et al. 2011).

In this paper, we introduce a novel job scheduling and resource management component within SST. It supports five state-of-the-art job scheduling algorithms: first come first serve (`FCFS`), shortest job first (`SJF`), longest job first (`LJF`), `FCFS` with `Best Fit` and `FCFS` with `Backfilling`. Our simulator is integrated within SST's discrete event-driven simulation engine. Each job goes through detailed simulation of job lifecycle management (e.g., submission, execution, completion) across various system configurations. The simulation environment can be configured using SST parameters to accurately represent diverse HPC architectures and resource availabilities. Compared to other simulators, such as CQsim and Alea 2, our simulation enhances scalability and supports a comprehensive range of job scheduling algorithms. Furthermore, it leverages SST's unique capabilities for high fidelity simulations. Furthermore, the existing job scheduling simulator cannot support workflow management of complex and interdependent workflow-based science applications.

We extend the capabilities of SST by introducing a workflow management component tailored for representing task dependencies typical in scientific computing. This addition complements our existing job scheduling and resource management components by enabling the detailed simulation of workflows. The workflow management feature within SST supports basic task scheduling algorithms (e.g., `FCFS`) and captures essential workflow dynamics through the discrete event-driven simulation engine. Workflow tasks are modeled to respect interdependencies, allowing the system to handle submission, execution, and completion phases dynamically. We configure the simulation environment with SST parameters that accurately mimic real-world computing architectures.

We validated our proposed simulator against CQsim (a cluster scheduling simulator) using traces collected from Grid Workload Archive and Parallel Workload Archive. We also performed performance testing of our simulator. The simulation results show that our simulator is accurate in job scheduling and is able to achieve good parallel performance. We also conducted validation of the workflow component using simplified workflow models based on well-known workflow analysis system such as Pegasus (Deelman et al. 2015) and Apache Airflow (Pegasus. 2024). Simulation indicates that the workflow component adheres effectively to task dependency constraints, while maintaining good performance across simulations.

2 SIMULATOR DESIGN

The overall simulator diagram is shown in Figure 1. The `SST Core` initializes the simulation, creating the necessary components and establishing links. The `Job Scheduling` module receives information about jobs from the components and places them in the queue based on their priorities and dependencies. After a job is submitted, it will enter the job waiting queue and `Job Scheduling` module will be invoked. The `job Scheduling` module will determine which jobs to run from the waiting queue based on the scheduling policies and the available resources from the `Resource Management` module. When a job is scheduled to run the `job Scheduling` will remove the job from the waiting queue and will put it in the running queue. The `Job Executor` will allocate the resources using the `Resource Management`. The `Job Executor` will then simulate the job execution. When a job completes its execution, the `Job Executor` will remove the job from the list of running jobs and will reclaim the resources occupied by the job. We describe each component in detail next.

2.1 Job Arrival and Scheduling

Upon arrival, each job is encapsulated as a `TaskEvent` class instance. Unique job identifiers and detailed resource requirements are assigned to each instance. The `Job Scheduling` module, in coordination

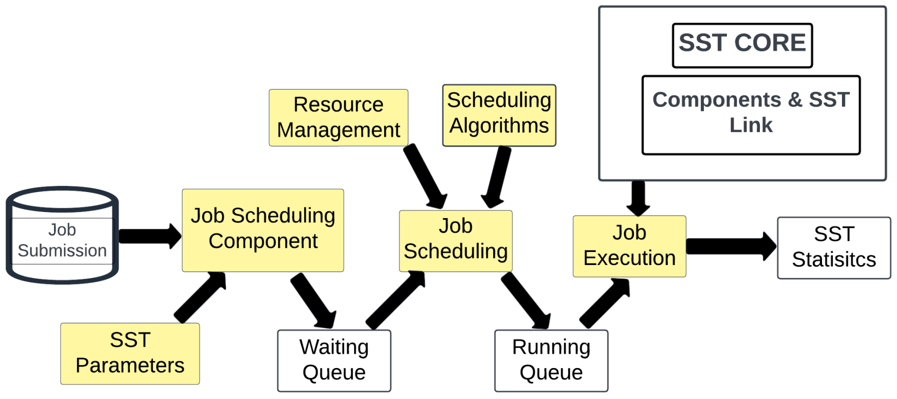


Figure 1: The simulator components.

with the `Resource Management` module assesses available and required resources, and ensures job queueing and scheduling. Serialization within the `TaskEvent` class ensures accurate transfer of task data across SST components. We show the `TaskEvent` class below.

```cpp
TaskEvent(const std::string& id, int cores, double time, size_t memory) :
        Event(), jobID(id), requiredCores(cores), executionTime(time), requiredMemory(
    memory), state(QUEUED) {}

    void serialize_order(SST::Core::Serialization::serializer& ser) override {
        Event::serialize_order(ser);
        ser & jobID;
        ser & requiredCores;
        ser & executionTime;
        ser & requiredMemory;
        ser & state;
    }
    ImplementClassName("TaskEvent");
```

Listing 1: Serialization method implementation for TaskEvent class.

Our simulator includes a range of job scheduling algorithms to support state-of-the-art HPC job scheduling techniques. We support five job scheduling algorithms: `FCFS`, `FCFS` with `Backfilling`, `FCFS` with `Best Fit`, `SJF`, and `LJF`. Each job scheduling algorithm satisfies diverse objectives (e.g., minimizing wait times, optimizing resource utilization), and reflects the multifaceted nature of job scheduling in HPC environments. We outline each job scheduling algorithm details below.

- **First-Come, First-Served (`FCFS`):** This technique schedules jobs in the order they arrive. This straightforward and fair approach ensures that jobs are processed in a sequence that they arrive, however, it may not always lead to the most efficient use of system resources, especially in systems with diverse job duration.
- **`FCFS` with `Backfilling`:** This algorithm enhances job throughput by allowing smaller jobs to proceed when larger jobs are waiting for resource availability. This method optimizes the overall utilization of system resources by dynamically adjusting the queue based on job size and available resources.
- **`FCFS` with `Best Fit`:** This technique allocates resources based on the closest match to the job's requirements, minimizing wastage and improving utilization. This strategy is particularly effective in environments where resource heterogeneity can lead to inefficiencies in allocation and utilization.

- **Longest Job First (`LJF`):** This algorithm prioritizes longer tasks to potentially optimize certain types of workloads or system conditions. The approach is less common. However, it can be beneficial in scenarios where longer jobs need to be expedited, albeit at the risk of increasing waiting times for shorter jobs.
- **Shortest Job First (`SJF`):** This technique minimizes average wait time by prioritizing shorter tasks. `SJF` is effective in homogeneous environments but hinges on the accurate estimation of job lengths, which can be challenging to ascertain (Smith 1978).

There are more recent and advanced scheduling techniques that can improve the job scheduling performance. For example, the deep reinforcement agent for scheduling (DRAS) technique proposed in (Fan et al. 2022) improves the job's adaptability and performance further. Although not currently implemented, we plan to implement such techniques for job scheduling techniques with better performance.

2.2 Job Execution and Resource Allocation

Upon determining the execution order, the `Job Scheduling` module allocates resources to jobs using the selected scheduling algorithm. The `Job Executor` module executes the jobs using SST. Algorithm 1 outlines how our simulator determines resources and executes jobs within an HPC environment. Job events are prioritized and managed in a dynamic priority queue. The simulator initializes predefined CPU resources and handles job submissions by enqueuing them in the priority queue. It allocates the necessary cores and schedules them for execution. As jobs are completed, resources are reclaimed and reallocated to pending jobs in the queue. We can evaluate the efficacy of various scheduling strategies in an HPC context with the help of continuous allocation and deallocation of job resources.

We chose SST due to number of its unique features. *First*, SST provides parallel discrete event simulation capability to achieve performance. *Second*, SST's architecture design is inherently modular. It incorporates various elements, such as `Merlin` for network modeling and `memHierarchy` for cache simulations. Each element serves specialized purposes within the simulation framework. For example, `Merlin` facilitates the simulation of diverse network topologies such as dragonfly, torus, mesh, and fat-tree, alongside various routing algorithms to simulate data packet traversal through the network. It enables users to specify network parameters such as link bandwidth, latency, and routing policies. Meanwhile, `memHierarchy` is dedicated to simulating the memory subsystem of computer architectures and supports modeling different levels of the memory hierarchy, including caches, main memory, and storage. *Third*, SST supports simulation of a wide range of applications – from computational physics to data analytics – and provides a comprehensive framework for accurate job execution simulation (Rodrigues et al. 2011). Our simulator leverages SST's robust event management capabilities (Simulator. 2024) to model the job execution and captures the dynamics of HPC job interactions. We can perform real-time monitoring of jobs via collecting SST's statistics.

After job execution, our simulator collects and analyzes job statistics utilizing SST's statistics framework. We gather statistics such as job durations, resource utilization, and system efficiency. The data collection and analysis helps in evaluating the effectiveness of different scheduling algorithms. The comprehensive statistics provided by SST enables us to refine our scheduling strategies continually.

3 WORKFLOW INTEGRATION IN SST JOB SCHEDULING

Current HPC systems increasingly demand simulations that not only manage isolated tasks, but also efficiently handle complex workflows consisting of multiple inter-dependent tasks. These workflows typically encapsulate scientific processes or business operations requiring robust scheduling and resource management. In this section, we introduce the workflow management component we incorporated within the SST. This component aims at simulating and managing the dynamics and dependencies of modern HPC workflows. The components supports both manually created and automatically generated directed acyclic graphs (DAG) to manage task dependencies. Other job scheduling simulators (e.g., CQsim, Alea

Algorithm 1 Resource Allocation/Deallocation

```
Initialize:
    Define totalCores, availableCores
    Create an empty priority queue for JobEvents based on their priority

Function scheduleJobEvent(jobEvent):
    Add jobEvent to the priority queue
    Attempt to allocate resources for jobEvent

Function allocateResources(jobEvent):
if availableCores $\geq$ jobEvent.requiredCores then
    Deduct jobEvent.requiredCores from availableCores
    Schedule jobEvent for execution after jobEvent.executionTime
    Upon job completion, trigger deallocateResources for jobEvent
end if

Function deallocateResources(jobEvent):
    Add jobEvent.requiredCores back to availableCores
if the priority queue is not empty then
    Get the next jobEvent from the queue
    Attempt to allocate resources for the next jobEvent
end if

Function mainSimulationLoop():
while simulation is running do
    if new jobEvent arrives then
        call scheduleJobEvent(jobEvent)
    end if
    Process any jobEvents whose execution time has elapsed
    for each completed jobEvent do
        call deallocateResources(jobEvent)
    end for
    Report using SST::Statistics
end while
```

2) either do not support workflow dependencies or offer limited support, such as simple precedences or chaining of jobs. The proposed workflow integrates directly into the core simulation environment of SST and enhances its utility for dynamic HPC job behavior simulation.

3.1 Task Representation Design

The `Task Scheduler` module models the individual computational jobs within a workflow. It encapsulates all the necessary information and behavior of these jobs. Following are some important attributes of this module.

- **task_id:** This is a unique identifier for the task.
- **execution_time:** This represents the estimated time to execute the task. This could be based on computational complexity or historical data.

- **resource_requirements:** This field represents required resources for the task, such as CPU cycles, memory size, and I/O bandwidth.
- **dependencies:** This represents a list of task IDs that must be completed before a particular task can start execution.
- **state:** This field represents the current state of the task (e.g., waiting, running, completed).

Within the task representation class, we check if all dependencies of the task have been satisfied. We also keep on checking if the state of a task has changed from waiting/running to completed. Once we detect that the state for a task is "completed", we trigger the rest of the tasks that have a dependency on the specific task that just completed. This helps in managing the execution order among the workflows in a particular job.

3.2 Workflow Management Component

The `Workflow Management` module in SST is designed to handle the details of task dependencies and resource allocation essential for scientific computing. Following are the key features of this module:

- **tasks:** This field represents a dictionary or list holding all `Task` objects within the workflow.
- **workflow_id:** This is a unique identifier for the workflow.
- **dependencies:** This is a representation of all dependencies in the workflow, potentially as a directed acyclic graph (DAG), where nodes are tasks and edges are dependencies.

Within the `Workflow Management` module, we process the tasks to identify and organize their dependencies, essential for scheduling and execution. We identify the dependencies of the tasks so we can schedule them to run upon the completion of the task(s) they are dependent on. We then return a list of tasks that are ready to be executed (i.e., all their dependencies have been satisfied).

DAG Representation. The DAG structure is a key modeling feature of workflows. It allows for efficient detection of ready tasks, and ensures that tasks are executed in the correct order. We implement the DAG using adjacency lists, due to the size and complexity of workflows (Gupta et al. 2017).

Scheduling and Execution. The workflow management involves a scheduler component that interacts with the `Workflow Management` module to allocate resources to Task objects based on their ready state and resource requirements. The `Workflow Management` module is responsible for triggering events in the SST simulation environment, corresponding to task starts and completions, to drive the simulation forward.

Figure 2 illustrates the integration of workflow management within the SST. It shows various components (e.g., `SST Core`, `Workflow Management`, `Task Scheduler`, `Resource Management`, and `Job Execution`) that are interconnected to facilitate simulation of HPC workflows. The `SST Core` coordinates between the `Workflow Management` system that handles task scheduling based on dependencies and resources, and the `Resource Management` system that allocates necessary resources to execute tasks. `Job Execution` processes the actual computation tasks, feeding results back into the `SST Core`, completing the cycle of task management and execution.

3.3 Workflow Input Specification

To specify workflows in our system, we use a JSON-based input format that defines tasks, their resource requirements, execution times, and dependencies. Below is an example of the JSON input format. The example illustrates how to define jobs, their dependencies, and the use of files as input and output within the workflow. This allows for efficient task management and resource utilization in a structured, JSON-based input format.

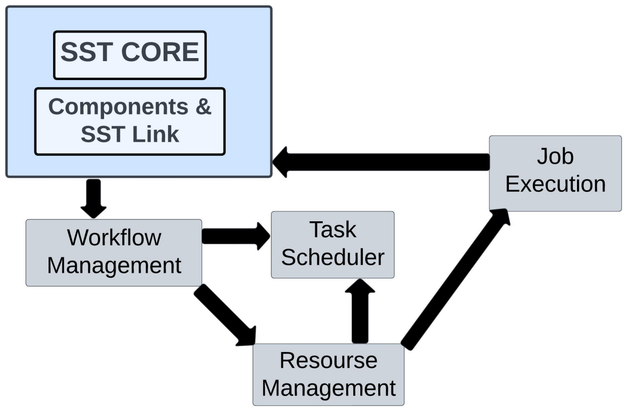


Figure 2: The workflow management components.

```
{
  "tasks": [
    {"id": 1, "execution_time": 100, "resources": {"cpu": 2, "memory": 1024}, "
   dependencies": []},
    {"id": 2, "execution_time": 150, "resources": {"cpu": 1, "memory": 512}, "
   dependencies": [1]},
    {"id": 3, "execution_time": 200, "resources": {"cpu": 1, "memory": 512}, "
   dependencies": [1]},
    {"id": 4, "execution_time": 300, "resources": {"cpu": 2, "memory": 1024}, "
   dependencies": [2, 3]}
  ],
  "resources_available": {"cpu": 10, "memory": 8192},
  "scheduling_policy": "Static",
  "preemption": false
}
```

Listing 2: Workflow input format.

For validation (Section 4.2), we tested with simplified workflow scenarios such as generated by well-established workflow systems (e.g., Pegasus and Apache Airflow (Mitchell et al. 2019)). By using these tests we were able to ensure that SST could handle basic workflows effectively, while adhering to dependency constraints and maintaining performance integrity.

Our simulator, leveraging the capabilities of the SST, creates the DAGs for managing task dependencies in HPC workflows. Users have the flexibility to manually define DAGs, which is crucial for workflows with well-understood and stable task relationships. This manual method ensures that the execution order adheres strictly to the user's specifications and control.

4 EXPERIMENTS

In this section, we present the validation and scalability evaluation of our proposed job scheduler simulator.

4.1 Experiment Setup

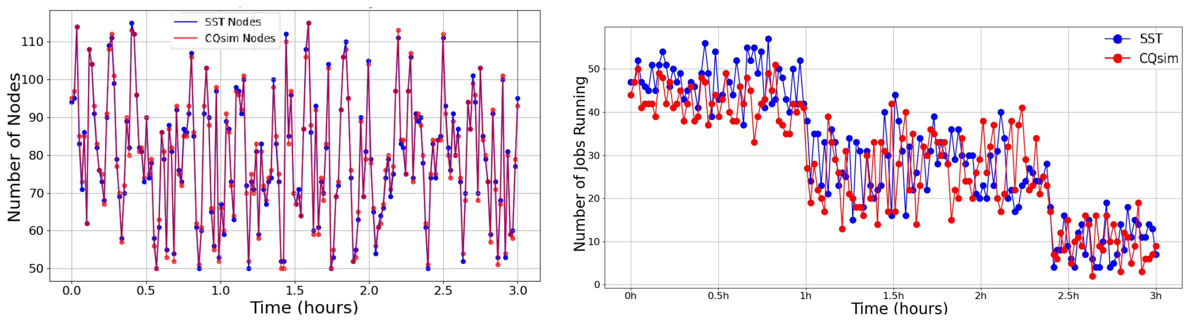
We leverage job trace from the Grid Workloads Archive (GWA) (Iosup et al. 2008). We validated our simulation output against CQsim (SPEAR Lab. 2023), an event-based cluster scheduling simulator. Specifically, we use the GWA-DAS2 trace from GWA to simulate real-world HPC environments. The

GWA-DAS2 trace contains about 1,124,772 jobs. Each job includes attributes such as job submission time, start/end time, memory usage, run time, wait time of jobs, the user information, the project or group associated, and the outcome of the job execution. The GWA-DAS2 trace provides a diverse range of job submissions, sizes, and resource requirements, offering a comprehensive basis for evaluating the scheduling algorithms. We also used the SDSC-SP2 log (San Diego Supercomputer Center. 2000b) from the Parallel Workloads Archive (San Diego Supercomputer Center. 2000a) to test the scalability of our job scheduler simulator. The SDSC-SP2 log contains attributes of jobs, such as job number, job submission time, wait time, run time, number of allocated processors, average CPU time used, requested time, requested memory, and queue number (San Diego Supercomputer Center. 2000b). The job log contains 73,496 job information. We use the trace data to determine the start time and end time of jobs in our simulator. We also utilize job trace data from various epigenomic sequencing projects to simulate real-world computational environments in bioinformatics. Specifically, our analysis includes the processing of sequences such as 4seq, 5seq, and 6seq, which are indicative of different epigenomic data complexities (Juve et al. 2013).

CQSim is a Python-based discrete-event-driven cluster scheduling simulator that evolved from QSim to enhance functionalities. It is designed with a modular Python architecture for reusability, extensibility, and efficiency. It simulates job submissions, allocations, and executions based on workload traces (SPEAR Lab. 2023). CQSim has been used in diverse HPC job scheduling mechanism evaluation such as energy-aware scheduling (Yang et al. 2013), fault-aware scheduling (Tang et al. 2009), etc.

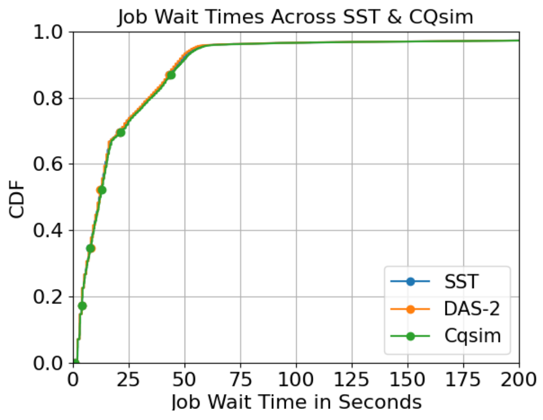
In addition to leveraging job traces from the Grid Workloads Archive (GWA) and validating against CQsim, we plan to extend our testing to encompass real-life workflow simulations using well-known workflow management systems like Pegasus. Pegasus is a robust workflow management system designed to automate the execution of complex workflows over diverse computing environments, including clouds and grids. It transforms high-level abstract workflow descriptions into executable workflows mapped onto physical resources. Notably, Pegasus optimizes the execution of these workflows by managing data placement and task clustering, which minimizes data transfer and enhances overall computational efficiency. Specifically, integrating real-life workflows from Pegasus into the Structural Simulation Toolkit (SST) environment will allow us to explore and evaluate the scheduler's effectiveness under realistic scientific computational loads. These workflows, often used in disciplines such as bioinformatics, astronomy, and climate science, provide a rigorous testbed for assessing SST's capabilities in handling complex dependency-driven job execution. The integration of Pegasus workflows will allow us to simulate more realistic and practical scenarios, ensuring that SST can support advanced scheduling techniques effectively in a high-performance computing (HPC) context. This approach will not only validate SST's performance under realistic conditions but also showcase its potential to adapt and manage workflows commonly used in scientific research, thereby enhancing its applicability and robustness in real-world HPC scenarios.

4.2 Results

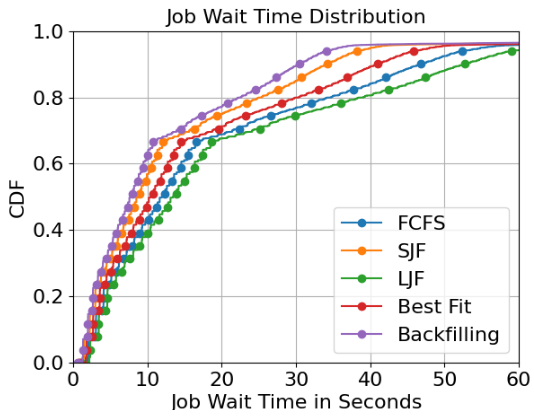


(a) Number of occupied nodes over simulation time (b) Number of jobs over simulation time

Figure 3: Comparison of our simulation output with CQSim.



(a) Wait time validation against CQSim.



(b) Wait time of different job scheduling algorithms

Figure 4: Job wait time validation and comparison of different scheduling algorithms.

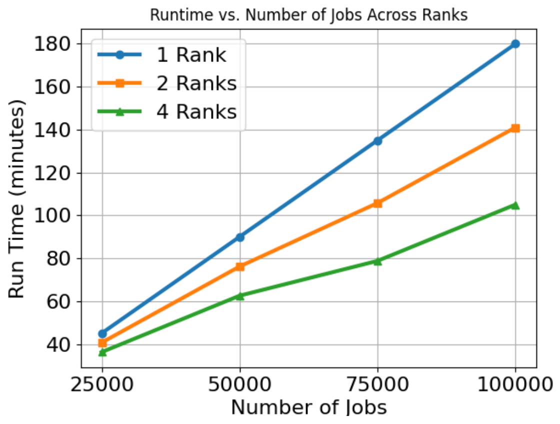
In this section, we present the results based on our simulator experiments. Figure 3(a) presents the occupancy of nodes throughout the simulation duration. We use the DAS-2 log for this simulation. We compare our simulation output with the CQsim simulation. As can be seen in the figure, the number of nodes occupied using our simulator is similar to the CQsim simulation output.

Figure 3(b) outlines the temporal distribution of job executions during the simulation. We observe a gradual decline in the number of active jobs. This trend is consistent across both simulation platforms, reinforcing the impact of job size on system throughput. The distribution of job sizes within the trace log (generally categorized as small, medium, and large across the initial, middle, and final phases, respectively) facilitates a comprehensive evaluation of the scheduling algorithm's performance across a range of workload scenarios.

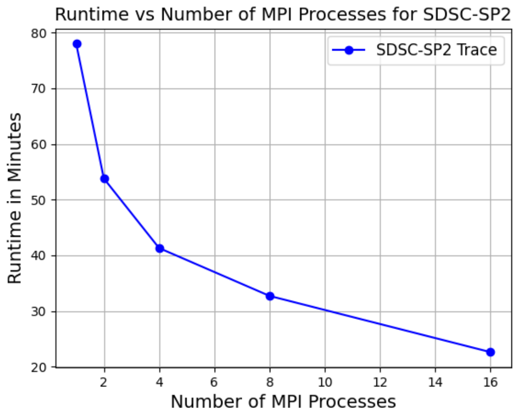
Figure 4 presents wait time analysis of our simulator. In Figure 4(a), we compare wait time measurements against CQSim. As can be seen in the figure, the job wait time of our simulator closely matches with the measurements from both CQSim and DAS-2 job trace. This validates our simulator's accuracy in prediction. In Figure 4(b), we present a comparative analysis of the five job scheduling algorithms. Although `FCFS` provides simplicity (ideal for jobs of uniform size) it has lower resource utilization. `FCFS` with `Best Fit` improves resource matching but does not significantly improve job completion times. `FCFS` with `Backfilling` maximizes resource utilization by intelligently filling scheduling gaps, while `SJF` reduces average job completion times by focusing on shorter jobs. In contrast, `LJF` is less efficient, often resulting in poorer utilization due to its preference for longer jobs.

Figure 5 demonstrates the parallel performance of our job scheduler when executing the DAS-2 & SDSC-SP2 workload on SST. Figure 5(a) shows performance for the DAS-2 log, while Figure 5(b) presents result for the SDSC-SP2 trace. As can be seen in the figures, simulator performance scales well as the number of MPI ranks increases. We also notice from Figure 5(a) that as the job sizes increased, we achieve greater speedup. SST's scalability property (e.g., support of parallel discrete-event simulation) and our model's optimizations enable good scalability. Figure 6 shows scalability result of the proposed workflow-based HPC job simulation.

Figure 6 demonstrates the parallel performance the workflow based scheduler while running the Galactic workflow from Pegasus workflow gallery. The Galactic Plane workflow uses the Montage image mosaic engine to transform all the images in 17 sky surveys to a common pixel scale of 1 second or arc, where all the pixels are co-registered on the sky and represented in Galactic coordinates and the Cartesian projection (Pegasus. 2024). As can be seen in the figures, simulator performance scales well as the number of MPI ranks increases.



(a) Running DAS-2 on SST across multiple ranks



(b) Running SDSC-SP2 on SST across multiple ranks

Figure 5: Scalability of our proposed simulator.

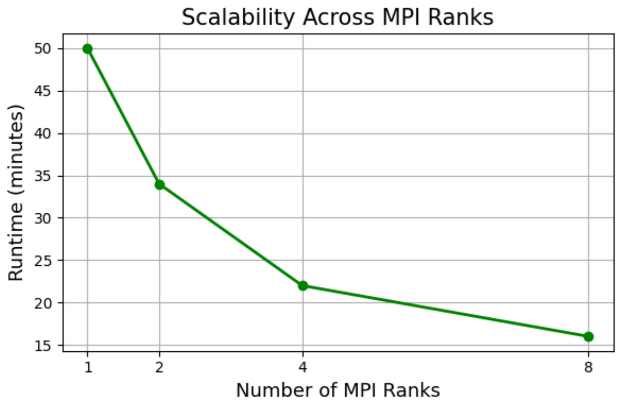


Figure 6: Scalability of workflow simulation.

Figure 7 presents the wait time comparison for the SIPHT workflow and our workflow simulator. The SIPHT workflow is an application in bioinformatics for automating the search for untranslated RNAs (sRNAs) within bacterial replicons (Juve 2014). As can be seen in Figure 7, the workflow wait time of our simulator closely matches with real-life measurements of the SIPHT workflow, validating our workflow component of the simulator.

5 CONCLUSION

In this paper, we propose a job scheduling simulator for scheduling HPC jobs on large-scale HPC platform. Our proposed approach features integration of job scheduling and resource management components on event-driven simulator, SST. We validate our proposed model against job scheduling simulator using HPC job traces. We demonstrate accuracy and parallel performance of our simulator in HPC job scheduling and resource management. Our workflow component demonstrates the potential to simulate HPC job behaviors with workflow dependencies. The preliminary results confirm its effectiveness in handling task dependencies and resource allocations.

In the next step, we plan to carefully study HPC job managers (e.g., SLURM) and develop models to support heterogeneous job and multi-cluster operation. We also plan to study other new job scheduling

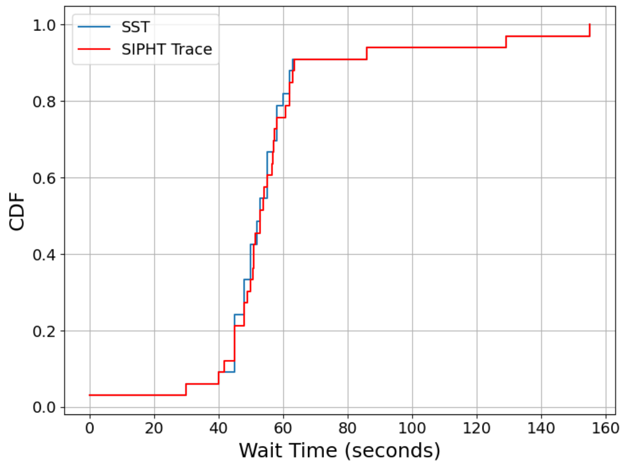


Figure 7: Job wait time validation of workflow simulation.

strategies (e.g., using reinforcement techniques) to improve job scheduling performance. We will incorporate these advanced job scheduling techniques in our simulator. We also aim to enhance SST's workflow management by integrating dynamic scheduling and preemption capabilities to better adapt to fluctuating workloads and resource availability. We will also develop support for complex, interdependent workflows, drawing on features from established systems like Pegasus and Apache Airflow. These improvements will enable more efficient handling of diverse scientific workflows, aligning SST more closely with modern high-performance computing needs. In the future, we will also focus on integrating advanced resource scheduling (e.g., dynamic scheduling) and resource allocation strategies (e.g., preemption capability) to SST's workflow management component.

ACKNOWLEDGMENTS

This work is supported in part by the U.S. National Science Foundation under grants CNS-2300124, and OAC-2411456, and supported by the U.S. Department of Energy, Office of Science under Contract No. DE-AC02-06CH11357.

REFERENCES

Buyya, R. and M. Murshed. 2002. "Gridsim: A Toolkit for The Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing". *Concurrency and Computation: Practice and Experience* 14(13-15):1175–1220.

Deelman, E., K. Vahi, G. Juve, M. Rynge, S. Callaghan, Maechling *et al*. 2015. "Pegasus, A Workflow Management System for Science Automation". *Future Generation Computer Systems* 46:17–35.

Fan, Y., B. Li, D. Favorite, N. Singh, T. Childers, P. Rich *et al*. 2022. "DRAS: Deep Reinforcement Learning for Cluster Scheduling in High Performance Computing". *IEEE Transactions on Parallel and Distributed Systems* 33(12):4903–4917.

Gupta, I., A. Choudhary, and P. K. Jana. 2017. "Generation and Proliferation of Random Directed Acyclic Graphs for Workflow Scheduling Problem". In *Proceedings of the 7th International Conference on Computer and Communication Technology*. November 24th-26th, Allahabad, India, 123–127.

Iosup, A., H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters *et al*. 2008. "The Grid Workloads Archive". *Future Generation Computer Systems* 24(7):672–686.

Juve, G. 2014. "Pegasus Workflow Gallery - SIPHT Workflow". https://pegasus.isi.edu/workflow_gallery/gallery/sipht/index.php. Accessed on: 21st April 2024.

Juve, G., A. Chervenak, E. Deelman, S. Bharathi, G. Mehta and K. Vahi. 2013. "Characterizing and Profiling Scientific Workflows". *Future Generation Computer Systems* 29(3):682–692.

Klusáček, D. and H. Rudová. 2010. "Alea 2: Job Scheduling Simulator". In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*. March 15th-19th, Malaga, Spain, 1–10.

Mitchell, R., L. Pottier, S. Jacobs, R. F. da Silva, M. Rynge, K. Vahi *et al*. 2019. "Alea 2: Job Scheduling Simulator". In *2019 IEEE International Conference on Big Data (Big Data)*. December 9th-12th, Los Angeles, CA, USA, 4537–4544.

Pegasus. 2024. "Workflow Gallery". https://pegasus.isi.edu/workflow_gallery/. Accessed: 12th April 2024.

Rodrigues, A., K. Hemmert, B. Barrett, C. Kersey, R. Oldfield, M. Weston *et al*. 2011. "The Structural Simulation Toolkit". *ACM SIGMETRICS Performance Evaluation Review* 38(4):37–42.

San Diego Supercomputer Center. 2000a. "Parallel Workloads Archive". https://www.cs.huji.ac.il/labs/parallel/workload/. Accessed: 13th October 2023.

San Diego Supercomputer Center. 2000b. "SDSC SP2 Workload Log". https://www.cs.huji.ac.il/labs/parallel/workload/l_sdsc_sp2/. Accessed: 13th October 2023.

Simakov, N. A., M. D. Innus, M. D. Jones, R. L. DeLeon, J. P. White, S. M. Gallo *et al*. 2017. "A Slurm Simulator: Implementation and Parametric Analysis". In *8th International Workshop on Performance Modeling, Benchmarking, and Simulation (PMBS)*. November 13th, Denver, CO, USA, 197–217.

SST Simulator. 2024. "SST Elements Repository". https://github.com/sstsimulator/sst-elements.git. Accessed: 7th February 2023.

Smith, J. E. 1978. "A New Class of Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment". *Journal of the ACM* 25(3):455–470 https://doi.org/10.1145/322063.322079.

Soysal, M. and A. Streit. 2021. "Collection of Job Scheduling Prediction Methods". In *24th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*. May 21st, Virtual Event, 35–42.

SPEAR Lab. 2023. "CQSim - A Discrete-Event Driven Scheduling Simulator". https://github.com/SPEAR-UIC/CQSim. Accessed: 7th February 2024.

Tang, W., Z. Lan, N. Desai, and D. Buettner. 2009. "Fault-Aware, Utility-Based Job Scheduling on Blue, Gene/p Systems". In *2009 IEEE International Conference on Cluster Computing and Workshops*. August 31st- September 4th, New Orleans, LA, USA, 1–10.

Yang, X., Z. Zhou, S. Wallace, Z. Lan, W. Tang, S. Coghlan *et al*. 2013. "Integrating Dynamic Pricing of Electricity into Energy Aware Scheduling for HPC Systems". In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. November 17st-22nd, Denver, CO, USA, 1–11.

Yoo, A. B., M. A. Jette, and M. Grondona. 2003. "Slurm: Simple Linux Utility for Resource Management". In *9th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*. June 24th, Seattle, WA, USA, 44–60.

AUTHOR BIOGRAPHIES

ABUBEKER ABDURAHMAN is currently an undergraduate Research Assistant at The University of Toledo where he is pursuing a Bachelor's of Science in Computer Science & Engineering. His email address is abubeker.abdurahman@rockets.utoledo.edu.

ABRAR HOSSAIN is currently working as a Graduate Research Assistant at The University of Toledo where he is also pursuing a Master's in Computer Science. His research focuses on High-performance computing (HPC) and Stochastic Modeling, Control, and Optimization. His email address is abrar.hossain@rockets.utoledo.edu.

KEVIN A. BROWN is the inaugural Walter Massey Fellow at Argonne where he conducts research in the Mathematics and Computer Science (MCS) Division. He received his M.Sc. and Ph.D. in Mathematical and Computing Sciences at the Tokyo Institute of Technology in 2014 and 2018, respectively. He worked as a Postdoctoral Appointee in the Argonne Leadership Computing Facility (ALCF) where he optimized network designs and communication strategies for Exascale supercomputers by utilizing advanced routing, quality-of-service, and congestion management mechanisms. His email address is kabrown@anl.gov.

KAZUTOMO YOSHII is a Software Development Specialist at Argonne National Laboraotry. He received a M.S. in computer science from Toyohashi University of Technology in Japan. His research interests include hardware/software co-design, heterogeneous and reconfigurable computing/field-programmable gate arrays, data movement and memory management, and edge computing architecture. His email address is kazutomo@mcs.anl.gov.

KISHWAR AHMED is an Assistant Professor in the Department of Electrical Engineering and Computer Science at the University of Toledo. He received a Ph.D. degree from School of Computing and Information Sciences, Florida International University. His research interests include high-performance computing, energy-efficiency, and optimization. His email address is kishwar.ahmed@utoledo.edu.