IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 20, 2020, accepted January 4, 2021, date of publication January 20, 2021, date of current version February 1, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3053162

Preemptive Parallel Job Scheduling for Heterogeneous Systems Supporting Urgent Computing

MULYA AGUNG[1], YUTA WATANABE[2], HENNING WEBER[3], RYUSUKE EGAWA[1,4], (Member, IEEE), AND HIROYUKI TAKIZAWA[1,5]

[1]Cyberscience Center, Tohoku University, Sendai 980-8578, Japan
[2]AI Platform Division, NEC Corporation, Tokyo 108-8001, Japan
[3]Department Systems and Operations, Deutscher Wetterdienst, 63067 Offenbach, Germany
[4]Graduate School of Engineering, Tokyo Denki University, Tokyo 120-8551, Japan
[5]Graduate School of Information Sciences, Tohoku University, Sendai 980-8578, Japan

Corresponding author: Mulya Agung (agung@tohoku.ac.jp)

This work was supported in part by the Ministry of Education, Culture, Sports, Science and Technology (MEXT) Next Generation High-Performance Computing Infrastructures and Applications Research and Development Program ''Research and Development of A Quantum-Annealing-Assisted Next Generation High Performance Computing (HPC) Infrastructure and its Applications'', and in part by the Grant-in-Aid for Scientific Research (A) #20H00593.

ABSTRACT Dedicated infrastructures are commonly used for urgent computations. However, using dedicated resources is not always affordable due to budget constraints. As a result, utilizing shared infrastructures becomes an alternative solution for urgent computations. Since the infrastructures are meant to serve many users, the urgent jobs may arrive when regular jobs are using the necessary resources. In such a case, it is necessary to preempt the regular jobs so that urgent jobs can be executed immediately. Most conventional methods for job scheduling have focused on reducing the response times and waiting times of all jobs. However, these methods can delay urgent jobs and hinder them from being completed within a stipulated deadline. Furthermore, in heterogeneous systems with coprocessors, preemption becomes more difficult because coprocessors rely on several system software functionalities provided by the host processor. In this paper, we propose a parallel job scheduling method to effectively use shared heterogeneous systems for urgent computations. Our method employs an in-memory process swapping mechanism to preempt jobs running on the coprocessor devices. The results of our simulations show that our method can achieve a significant reduction in the response time and slowdown of regular jobs without substantial delays of urgent jobs.

INDEX TERMS Job scheduling, urgent computing, heterogeneous systems, preemption, process swapping.

I. INTRODUCTION

Large-scale scientific computing has played an important role in critical decision-making systems. For example, when disasters such as earthquakes, tsunamis, storms, and floods happen, making timely decisions for mitigating the damages and reducing casualties is of vital importance. Urgent computing (UC) is a class of computing to support such computations [1], [2]. In UC, a computation operates under a strict deadline after which the computation results are useless.

The most widely used and reliable approach for supporting UC is to use dedicated resources [3], [4]. The main advantage of using dedicated resources is that it can provide immediate and dedicated accesses to urgent computations. Despite this advantage, it also has limitations. Setting up dedicated resources for each specific urgent computation is not economically viable, especially if the computations occur rarely and require vast amounts of resources. Moreover, the damage to the dedicated resources can make it impossible to perform urgent computations that can help to mitigate the damages [5]. Due to these limitations, using existing shared infrastructures becomes an invaluable approach to supporting UC [6].

Several studies [1], [6] have shown that several challenges must be addressed to enable shared infrastructures for UC. One important challenge is to provide job scheduling methods that can handle urgent jobs. However, since the shared infrastructures also serve regular uses of the resource, urgent jobs can be significantly delayed by regular jobs. On the other hand, considering only urgent jobs may also greatly increase

The associate editor coordinating the review of this manuscript and approving it for publication was Bong Jun David Choi.

the response time of regular jobs. Therefore, in shared infrastructures supporting UC, it is important to prevent delays of urgent jobs while also decreasing the response time of regular jobs.

Not only at the research level but also at the practical operation level, efficient use of shared resources is strongly demanded. In fact, some major HPC centers have their daily workloads and respond to urgent jobs if necessary by temporarily suspending daily jobs. The Deutscher Wetterdienst (DWD) is one such center, and punctual calculations of weather forecasts are its daily urgent jobs. Other urgent jobs are the rapid provision of radionuclide dispersion forecasts for emergency situations. All the urgent jobs preempt regular jobs of scientific workloads. Therefore, there is a strong demand for using shared infrastructures for urgent jobs as well as other jobs.

To support UC, various job scheduling methods have been recommended by a previous study [1]. This study also discussed the importance of preemption for guaranteeing the immediate execution of urgent jobs. However, most of the previous job scheduling methods do not support preemption [6]. Preemption is particularly hard to realize in shared infrastructures because the job scheduling methods rely on preemption mechanisms supported by the resource providers. Furthermore, preemption can delay the execution of urgent jobs because it needs to save the intermediate state of the preempted jobs. Hence, in shared infrastructures for UC, reducing the preemption delay is necessary.

At present, coprocessors, such as graphics processing units (GPUs) and vector processors, have been successfully used in shared infrastructures to accelerate various scientific applications [7]. In this work, a system equipped with different kinds of processors is referred to as a heterogeneous system. Typically, a CPU hosts other kind of processors, such as the vector processors considered in this paper. For this kind of system, resource providers need to handle the preemption of jobs running on coprocessor devices. However, preemption becomes more challenging due to integration between coprocessor devices and the host processor [8]. To preempt a job running on the device, the memory used by all processes of the job must be moved from the device to the host memory because coprocessors do not fully support system software functionalities in general and thus rely on the host processor. Moving process memory from device to host memory and vice versa is called *process swapping* [9]. Therefore, process swapping is crucial for job scheduling methods in heterogeneous systems supporting UC.

In this work, we propose a job scheduling method for shared infrastructures supporting UC. The proposed method consists of a preemption mechanism for heterogeneous systems using in-memory process swapping, named *partial process swapping* (PPS), and a preemption-based job scheduling algorithm, named *urgent job first with backfilling* (UJFB), that can prevent significant delays of urgent jobs while decreasing the response time and slowdown of regular jobs. By swapping processes to memory, the proposed method can achieve low preemption delays, and thus, it can avoid the slowdown of urgent jobs. Moreover, it decreases the response time and slowdown by backfilling regular and urgent jobs.

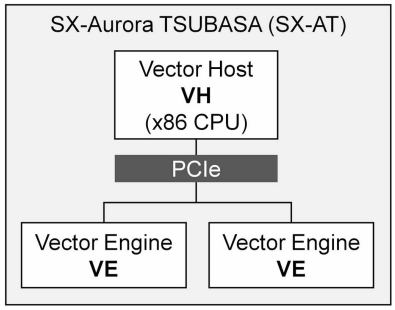


FIGURE 1. SX-AT architecture.

PPS has been evaluated on a real system of host CPUs and vector processors, called NEC SX-Aurora TSUBASA (SX-AT) [10]. As illustrated in Figure 1, SX-AT is a heterogeneous system consisting of a vector host (VH) and one or more vector engines (VEs) as coprocessors. The VH is an x86-based CPU featuring a standard x86 Linux kernel that provides standard operating system functions, and each VE is implemented as a PCI express (PCIe) card equipped with the vector processor. VEs that run on a VH are controlled by a sub-operating system called VEOS. However, a VE runs only user processes, not a fully featured OS. Some system functions need the data to be on VH memory because, for OS, the VE process appears to be running on the VH. Therefore, if a job running on the VE needs an OS functionality, the job data on VE memory need to be transferred to the host memory.

The evaluation results show that PPS can achieve negligible preemption delays, indicating the importance of the mechanism in preventing delays of urgent jobs. UJFB has been extensively evaluated with four workloads of real systems on a job scheduling simulator. The evaluation results show that it can achieve almost no slowdown of urgent jobs and a response time and slowdown of regular jobs comparable to those of existing backfilling-based job scheduling algorithms. These results demonstrate the importance of the proposed method in enabling shared infrastructures for supporting UC.

In this paper, we provide the following contributions:

- A low-delay preemption mechanism for heterogeneous systems supporting UC.
- A metric for evaluating the effects of job scheduling methods on the delays of urgent jobs.
- A preemption-based job scheduling algorithm for preventing the delays of urgent jobs while decreasing the response time and slowdown of regular jobs.

To the best of our knowledge, this work is the first to consider a new job scheduling method and a preemption mechanism for shared heterogeneous systems supporting UC.

The remainder of this paper is organized as follows. The related studies and problem description are presented in Sections II and III, respectively. Then, we describe the proposed method in Section IV. To validate the proposed

method, we discuss our evaluation results in Section V. Finally, Section VI gives the conclusions and future work of this paper.

II. RELATED WORK

This section reviews related work. First, several studies on urgent computing are reviewed. Then, this section discusses parallel job scheduling methods, which include nonpreemptive and preemptive job scheduling algorithms and job scheduling methods considering deadlines. Finally, it discusses preemption mechanisms for heterogeneous systems.

A related study [2] provided a definition of UC and identified unique challenges and requirements of UC systems. One important requirement is that a UC system must be able to guarantee immediate executions of urgent jobs. The previous work also discusses the necessity of preemption-based job scheduling in UC systems. A system, called SPRUCE [1], was proposed for supporting UC. The system aims to provide resources quickly and efficiently to high-priority applications that must be executed without delay. However, a more recent study [6] shows that enabling shared infrastructures for UC remains challenging. One important challenge is that no existing job scheduling policies are suitable for shared infrastructures supporting UC. In job scheduling policies targeting UC, the preemption is mandatory for preventing delays of urgent jobs. In this work, we propose a job scheduling method to address this challenge. To the best of our knowledge, this work is the first to develop a preemption-based UC system for heterogeneous shared infrastructures while considering a new job scheduling method and a preemption mechanism.

Parallel job scheduling methods have been widely studied [11]–[14]. Most conventional studies on parallel job scheduling have focused on reducing the average response time and slowdown among jobs. First come first serve (FCFS) is the simplest method for scheduling jobs. Although it is predictable, it suffers from low system utilization [15]. Thus, many related studies have focused on backfilling-based job scheduling methods. A job scheduling method called EASY [14] has been proposed to improve the performance of backfilling for smaller jobs. Another related study [16] focused on improving the EASY scheduler using job runtime predictions. Although the previous studies have shown the effectiveness of backfilling-based methods in reducing the response time and slowdown, all the previous methods do not guarantee immediate executions of urgent jobs. In contrast to these studies, this work proposes a preemptive backfilling method that prevents delays of urgent jobs while reducing the response time and slowdown of regular jobs.

Some recent studies have proposed job scheduling methods considering the deadlines of jobs. Deadline-based backfilling (DBF) [17] reduces the average wait time of jobs by considering the deadlines of jobs. By distinguishing deadline-driven jobs from regular jobs, it can execute a deadline-driven job earlier by delaying the regular jobs. A related study [18] has proposed an algorithm for guaranteeing the deadlines of jobs in cloud computing systems. However, all these studies do not assume unpredictability of arrival times of urgent jobs. Moreover, they do not support preemption, and thus they cannot guarantee immediate execution of urgent jobs.

The popularity of GPU-CPU heterogeneous systems has led to many studies focusing on preemption mechanisms for GPU. CheCUDA [19] and CheCL [20] have been proposed for checkpointing CUDA and OpenCL applications, respectively. However, these mechanisms have several limitations. CheCL suffers from large runtime overheads when managing the GPU states, while CheCUDA is no longer available in the recent CUDA environment. Moreover, checkpointing overheads are not acceptable for UC because they can significantly increase the preemption delay. A more recent study has proposed a preemption mechanism to schedule multiprocess applications on heterogeneous clusters with GPUs [8]. However, the mechanism focuses on improving GPU utilization and does not consider the unpredictability of arrival times of urgent jobs. Therefore, these previous mechanisms are not applicable to heterogeneous systems supporting UC.

In addition to GPUs and vector processors, field-programmable gate arrays (FPGAs) are now commonly used in HPC centers [21]. However, a recent study has shown that FPGAs are currently not suitable for preemptive executions [22]. Furthermore, preempting a job running on an FPGA results in significant performance penalties and implementation difficulties because saving and restoring the job state can be a very complex operation [23], [24]. Cooperative scheduling mechanisms [22], [25] have been proposed to reduce the performance penalties. However, in these mechanisms, a job can only be preempted if a predefined state is met. In UC, these mechanisms can delay an urgent job because it may need to wait until regular jobs can be preempted. Due to these limitations, our proposed preemption mechanism does not consider process swapping between an FPGA and host processors.

III. PROBLEM DESCRIPTION

This section describes the problem of job scheduling for shared infrastructures supporting UC. First, we describe two scheduling metrics that are commonly used to evaluate parallel job scheduling methods. Then, we use these metrics to discuss a motivating example with tsunami simulations.

A. PARALLEL JOB SCHEDULING METRICS

This paper considers the workload as the data regarding each job submitted to a HPC center during a certain time period. The workload contains the arrival time, the number of requested processors, the estimated runtime, and the actual runtime of each job during the period. This workload is then processed by a scheduler to evaluate the scheduling performance using the job scheduling metrics. The metrics are measured by analyzing the job trace of the evaluation. This trace contains information that is not provided in the workload, such as the starting time and completion time of each job during the evaluation.

Two job scheduling metrics, the response time and slowdown, have been used extensively in related studies [26]–[28]. Slowdown is defined as the response time of the job normalized by the running time. Thus, slowdown represents the job response time proportional to its running time. The running time is the wall clock time from when the job is started until it completes its execution, while the response time is the total wall clock time from when the job is submitted to the system until it completes its execution. Hence, the response time considers the running time and the time spent by the job in waiting before execution, as shown by Equation (1).

$$Response = T_w + T_r, \tag{1}$$

$$Slowdown = \frac{Response}{T_r}, \tag{2}$$

where T_w and T_r are the waiting time and running time of the job, respectively. Then, slowdown can be calculated using Equation (2). A slowdown equal to 1 means that the job executes without any delays. Therefore, when an urgent job is executed, a slowdown higher than 1 can cause it to miss the deadline.

B. A MOTIVATING EXAMPLE WITH TSUNAMI SIMULATIONS

As a motivating example, we evaluate the performance of regular and urgent jobs with the SDSC-DS workload [29]. It is composed of 96,089 jobs, which are executed on the DataStar cluster of the San Diego Supercomputer Center between March 2004 and March 2005. The cluster consists of 184 nodes, with a total of 1,664 processors. We run simulations with the workload using a job scheduling simulator, called Alea [30], which is based on the GridSim toolkit [31].

Since no urgent jobs are defined in the workload, we randomly inject an urgent job at busy times in different simulated months. The busy time is when the resources utilization is equal to or higher than 75%. The urgent jobs are injected at busy times to simulate the cases in which urgent jobs arrive when most of the resources are used by regular jobs. Thus, these cases will likely delay executions of urgent jobs. Table 1 shows job configurations that are used for injecting urgent jobs. Each configuration consists of the number of CPUs required by the job and the job length, which is the running time of the job. These values are adopted from the empirical results of tsunami simulations obtained in related studies [32], [33]. As shown in these studies, the tsunami simulation is expected to finish with a 10-minute deadline to mitigate the loss caused by the tsunami event.

Figure 2 shows the arrival times of the urgent jobs injected in the SDSC workload with the FCFS algorithm. The x-axis shows the hours in a period of three months after the first job arrival, while the y-axis shows resource utilizations during that period. The vertical red lines indicate the hours at which the urgent jobs arrive. As shown in the figure, three urgent jobs are injected at busy times in different simulated months. Each urgent job is injected at the hour when the resources utilization is at least 75%. Thus, to avoid delaying the urgent job that arrives at that hour, it is necessary to preempt the regular jobs that are still running.

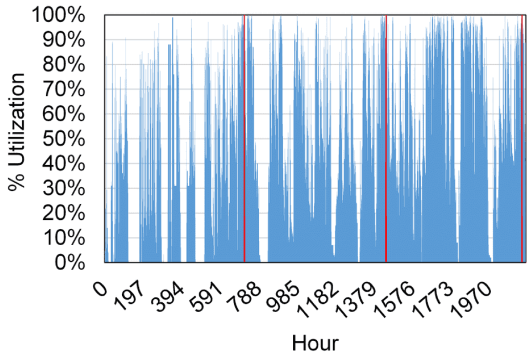
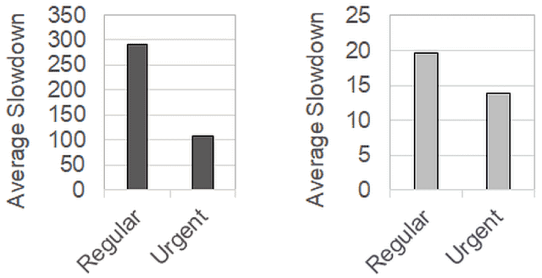


FIGURE 2. Arrival times of injected urgent jobs.

TABLE 1. The number of processors and lengths of urgent jobs for the tsunami simulation.

Number of CPUs	Job length (minutes)	Maximum slowdown
128	10	1
256	6.5	1.54
512	4	2.5



(a) FCFS. (b) Conservative Backfill.

FIGURE 3. Slowdowns of regular and urgent jobs.

This evaluation compares two widely used job scheduling algorithms, FCFS and conservative backfilling [34], [35]. In the backfilling algorithm, the actual runtime is used as the estimated runtime. Thus, it assumes accurate estimates of the runtime. For the evaluation, we measure the average slowdown of regular and urgent jobs. Table 1 shows the maximum slowdown with which a system can satisfy the 10-minute deadline requirement of tsunami simulations. As shown in this table, the running time of an urgent job with 128 processors is 10 minutes, which is equal to the deadline requirement. This result means that this job does not allow any delays to its execution, i.e., the slowdown is equal to 1. On the other hand, the running times of jobs with 256 and 512 processors are less than 10 minutes. Therefore, slowdowns less than or equal to their maximum slowdowns remain acceptable for these jobs.

Figure 3 shows the slowdown results of the two algorithms for regular and urgent jobs. As shown in the figure, the conservative backfill algorithm shows smaller slowdowns than FCFS, indicating that backfilling can reduce the wait times of regular and urgent jobs. For both methods, the slowdown of urgent jobs is smaller than that of regular jobs.

However, urgent jobs still suffer from significant delays because they need to wait for regular jobs to complete their execution. In the figure, these two algorithms delay urgent jobs with slowdown factors higher than 10, which are much higher than the maximum slowdowns allowed for the urgent jobs of the tsunami simulation.

The lateness of urgent jobs not only wastes computational resources but also implies that no damage mitigation effort can be performed. In the case of a tsunami simulation, an early deadline is preferred because the most important indicator occurs within a few minutes after the first detectable urgent event [36]. These results show that these conventional methods are not applicable to UC. Therefore, to support UC, shared infrastructures require scheduling algorithms that can reduce the slowdown of not only regular jobs but also urgent jobs.

Since urgent events usually occur unexpectedly at any time, an urgent job may arrive when the necessary resources are used by regular jobs. In this case, urgent jobs need to wait for the regular jobs to finish their executions. However, this will indeed delay the urgent job and may prevent it from finishing within its deadline. To prevent this occurrence, preempting the running regular jobs is a mandatory step for reallocating resources so that the urgent job can commence immediately. Therefore, preemption is vital for achieving job scheduling that can support urgent computations in shared infrastructures.

IV. PREEMPTIVE JOB SCHEDULING FOR URGENT COMPUTING

In this section, we describe the proposed job scheduling method that addresses the problems in enabling shared heterogeneous systems for UC. The proposed method consists of two parts:

1) A preemption mechanism using in-memory process swapping.
2) A preemption-based job scheduling algorithm for preventing significant delays of urgent jobs.

In addition to the two mechanisms, we also describe the metric for evaluating the lateness of urgent jobs.

A. PREEMPTION USING IN-MEMORY PROCESS SWAPPING

Several types of preemption have been used in related studies [37], [38]. The first type is kill/restart. In this type, preempting a job will kill the job and start it again at a later time. Once the job has been killed, it will return to the scheduler queue. Any partial results will be lost, and the time spent running the preempted job is wasted. Due to this waste, the actual running time of the preempted job will be longer than normal. Therefore, this type of preemption can significantly increase the response time of the system if there are many preempted jobs.

The second type of preemption is suspend/resume. In this type of preemption, a job is suspended and then resumed at a later time. All the processes associated with the job are stopped, but the state of each process is retained so that the job can continue when it is resumed. When preempting a job to execute an urgent job, the operating system will need to transfer the states to a swap memory space before the new job can start. The time spent for swapping the memory space is referred to as the *swapping time* [39], [40].

The main advantage of the suspend/resume mechanism is that it can significantly decrease response times of preempted regular jobs compared with the kill/restart mechanism because the jobs do not need to restart their execution from the beginning. Due to this advantage, this work focuses on a suspend/resume mechanism for preempting jobs. However, the swapping time may cause additional delays in the start time of the urgent job compared with the kill/restart mechanism. Therefore, it is important to achieve short swapping times to prevent significant delays of urgent jobs.

Since a job can have multiple processes, preempting the job must wait until all the processes have been swapped out. Thus, preemption delay depends on the swapping time of each process and the number of processes to be swapped. The swapping time can vary depending on several factors, such as the process size and the bandwidths of the memory and interconnect. Several studies have shown that swapping multiple processes simultaneously can significantly reduce the total swapping time of the processes because it can increase the bandwidth utilization [41]–[43]. Therefore, in UC, simultaneously swapping the job processes is crucial for reducing the preemption delay. In this case, the preemption delay (PD) is determined by the longest swapping time among the processes, as shown in Equation (3). In this equation, S_p is the swapping time of the process p, and N_p is the total number of processes of the job.

$$PD = \max_{1 \le p \le N_p} S_p. \tag{3}$$

Many operating systems, such as Linux, provide suspend/resume mechanisms for jobs running on host CPUs. However, these mechanisms are often not applicable for jobs running on coprocessors. As an example of heterogeneous systems with coprocessors, this work focuses on a preemption mechanism for SX-AT [10]. Linux uses a demand paging mechanism that copies data from the disk into the main memory only when a page fault occurs, in which case an attempt is made to access the data [44]. This mechanism relies on in-order executions of instructions for proper timing of the swapping. However, in vector processors such as SX-AT, instructions can be executed in an out-of-order fashion [45]. Thus, the demand paging mechanism cannot be applied to vector processors. Therefore, to achieve suspend/resume preemption in SX-AT, we propose a memory swapping mechanism, called PPS. It swaps memory on a per process basis, as opposed to a per instruction basis. PPS has already been incorporated into the VEOS and is available on SX-AT [46].

PPS provides two system-level functions that save and restore the VE memory of a suspended VE process. The functions work at the system level because they need to save and restore the address space of the process between the

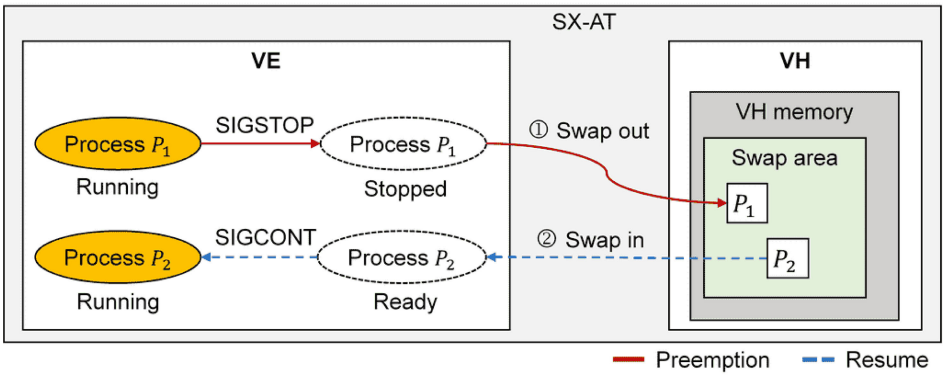


FIGURE 4. Preemption using PPS.

VE and VH. These two functions are illustrated in Figure 4. The first function, named *swap out*, releases VE memory by saving a part of the memory of one or more temporarily stopped VE processes to VH side memory. For this function, a memory space, named *swap area* [47], is allocated in the VH. The second function, named *swap in*, restores the memory saved in the VH memory to the VE memory so that the execution of the original process can be resumed.

When multiple processes are running on different VEs on a host, each process can transfer data to other processes through the interconnect between VEs using the message passing interface (MPI) [48] library. Consequently, each process needs to access data on VE memory during the transfer. Since different processes can asynchronously access VE memory, it is difficult to guarantee that no process will access the data on the VE memory that is swapped out. Swapping out all the data on VE memory will cause data transfer errors because the VE memory might be accessed after the swapping. PPS retains only a part of VE memory that could potentially be accessed by the MPI library, while all the other data in VE memory are swapped out.

The red lines in Figure 4 show the flow of preempting a job process using PPS. First, the process is stopped by sending the kernel SIGSTOP signal to the process. Then, PPS is used to swap out the process memory to the swap area. After the memory has been swapped out, the VE memory is released and another process can be executed on the VE using the memory. In a multiprocess job, PPS simultaneously swaps the processes running on different VEs to reduce the preemption delay. The same flow is executed for each process, and PPS will wait until all the processes have been swapped out. In practical uses of SX-AT, the swapping time also depends on contentions on the PCIe interconnect between VEs and the VH. In the case of simultaneous swapping, a contention will occur when swapping all the processes needs a data transfer larger than the bandwidth of the interconnect [43], which is approximately 40 GB/s for SX-AT [49]. When the contention occurs, the swapping time of each process could be longer than usual. In Equation 3, this effect is subsumed into the S_p of each process. Consequently, the preemption delay will increase in this case. Therefore, to accurately measure the preemption delay of the job, it is necessary to consider the swapping times of all processes altogether when the job is executed on the VE.

After a job is preempted, it can be resumed using the flow shown by the blue lines in Figure 4. First, PPS is used to swap in the process memory from the swap area to VE memory. After the process has been swapped in, it will release the memory on the VH. The process then can be resumed by sending the kernel SIGCONT signal to the process. Similar with the preemption flow, the resume flow is executed for all the processes in a multiprocess job. However, a preempted job needs to be resumed on the same VE and host from which it was preempted for two reasons. First, PPS swaps data from VE memory to the host memory, and thus, the data are available only on the same host. An important advantage of this mechanism is that it avoids longer preemption delays from migrating data across hosts [50]. Second, PPS partially swaps the VE memory. Therefore, resuming the execution of the process requires the data previously kept on the same VE.

B. UJFB: A PREEMPTIVE JOB SCHEDULING ALGORITHM CONSIDERING URGENT JOBS

As previously mentioned in Section III, the main purpose of UJFB is to simultaneously prevent urgent jobs from being delayed by regular jobs while also reducing the response time and slowdown of regular jobs. The UJFB algorithm is summarized as follows. It executes an urgent job by performing one of the following three subtasks.

1) Prioritize urgent jobs to be executed before regular jobs.
2) Backfill urgent jobs.
3) Preempt regular jobs to give resources to urgent jobs.

Figure 5 shows the proposed UJFB algorithm for executing urgent jobs. First, it evaluates if an urgent job can be executed using the currently available resources. If the available resources are sufficient for the execution, it will directly commence the urgent job. However, there is a possibility that the resources are previously reserved for regular jobs. In this case, UJFB will backfill the urgent job so that it can be executed immediately. If the available resources are insufficient for executing the urgent and regular jobs, the urgent job will

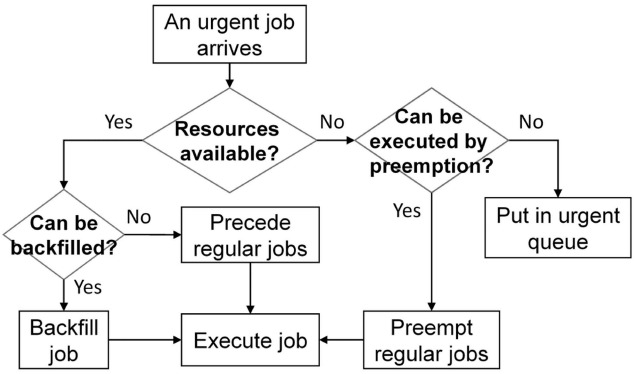


FIGURE 5. UJFB algorithm for executing urgent jobs.

precede the regular jobs. On the other hand, if the resources available for executing the urgent job are insufficient, UJFB will preempt the regular jobs that are currently running. In this case, the regular jobs are preempted using the flow shown in Figure 4. If the number of running jobs is higher than that required by the urgent job, UJFB will prioritize jobs that have longer remaining running times to be preempted. After the regular jobs are suspended, it can submit the urgent job for execution.

For backfilling, UJFB uses an approach similar to that of the conservative backfill algorithm [14]. However, in contrast to the conservative backfill algorithm, UJFB prioritizes urgent jobs to be backfilled before regular jobs. As a result, an urgent job can be executed immediately without preemption when the available resources are sufficient for the job execution.

As shown in Figure 5, it is possible that the urgent job cannot be executed even with preemption. This situation occurs when previous urgent jobs are currently running and using the necessary resources. In this case, UJFB moves the urgent job to a separate scheduler queue that is dedicated for urgent jobs, named *urgent-queue*. Thus, for this step, UJFB maintains two scheduler queues, one for regular jobs and the other for urgent jobs. Using these two queues, it can prioritize urgent jobs to be scheduled prior to regular jobs.

In UJFB, urgent jobs are executed under three cases. First, the urgent job is executed by preceding regular jobs that are waiting in the queue. Figure 6 shows an example of this case. In this figure, job 2 is preceded to execute the urgent job. In this case, although the urgent job can be executed immediately without preemption, the preceded job will be delayed. Second, the urgent job is executed by backfilling, as shown in Figure 7. In this case, the urgent job can be executed not only without preemption but also without preceding any regular jobs if they cannot be backfilled. However, this case only occurs if executing regular jobs does not delay the urgent job. In the example shown in the figure, the urgent job can be executed while executing jobs 6 and 7. Thus, UJFB does not delay all of these jobs in this case.

Third, UJFB executes the urgent job by preempting regular jobs. As shown in Figure 5, this occurs only if the urgent job cannot be executed under the previous two cases. Figure 8 shows an example of this case. In this example, the urgent job cannot be executed because all the necessary resources are used by regular jobs. Thus, to prevent delaying the urgent job, UJFB preempts jobs 9 and 10 that are currently running to provide the resources to the urgent job. The preempted jobs are then put back to the head of the regular queue. They are put at the head of the queue so that they can be immediately resumed after the urgent job finishes its execution. The resume flow in Figure 4 is used to resume these jobs. Therefore, among the three cases, only this case causes a preemption delay to execution of the urgent job.

Algorithm 1 shows the procedure of UJFB when a new job is submitted to the system. First, it determines if the job is regular or urgent (Line 2). If the job is urgent, the flow shown in Figure 5 is used to execute the job. Otherwise, the algorithm will put the job into the regular queue (Line 17). The complexity of UJFB is determined by the complexities of the `backfill` (Line 5) and the `selectJobs` (Line 10) functions. As with the conservative backfill algorithm, UJFB scans all jobs in the regular queue for each new job. Therefore, the complexity of the `backfill` function is also linear in the number of jobs [14]. The `selectJobs` function selects the regular jobs to be preempted by sorting the jobs based on their remaining running times. Therefore, its complexity is determined by the complexity of the sorting algorithm. In the case of Quicksort, the complexity is $O(n \log n)$ on average [51], where n is the number of regular jobs currently running.

C. THE URGENT LATENESS METRIC

In the case of regular jobs, the wait time and response time metrics are useful for showing how fast the system responds to a user. However, in the case of urgent jobs, these two metrics are not sufficient because urgent jobs are expected to finish as close as possible to their deadlines. Thus, for job scheduling algorithms that target UC, it is necessary to evaluate the maximum slowdown of the urgent jobs. To evaluate the effects of a job scheduling method on urgent jobs, we proposed a metric, named *Urgent Lateness* (U_L). To measure this metric, we first calculate the slowdown of each urgent job using Equation (2).

Then, U_L is defined as the maximum slowdown among urgent jobs, as shown in Equation (4).

$$U_L = \max_{1 \le u \le N_U} Slowdown_u, \tag{4}$$

where N_u is the number of urgent jobs. A higher value of U_L indicates a longer maximum slowdown of urgent jobs.

V. EXPERIMENTAL EVALUATION

In this section, we present the experimental setup and discuss the performance results on the Alea simulator. First, we discuss the performance results of PPS using OpenMP implementations of NAS parallel benchmarks (NPB) [52]. The results are obtained by measuring the swapping times during the executions of the applications. Then, we use the swapping

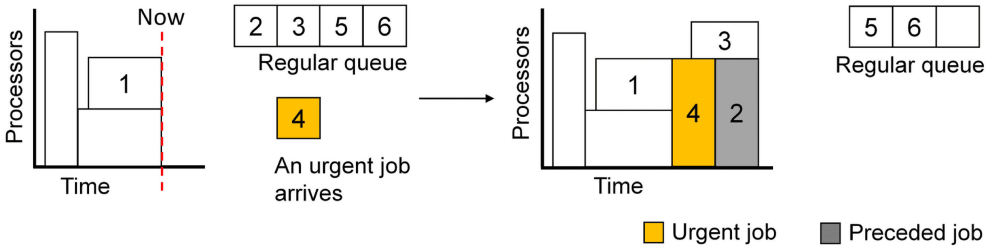


FIGURE 6. Urgent job execution by preceding regular jobs.

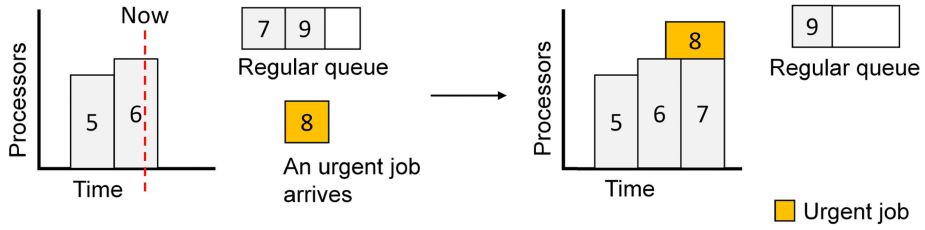


FIGURE 7. Urgent job execution by backfilling.

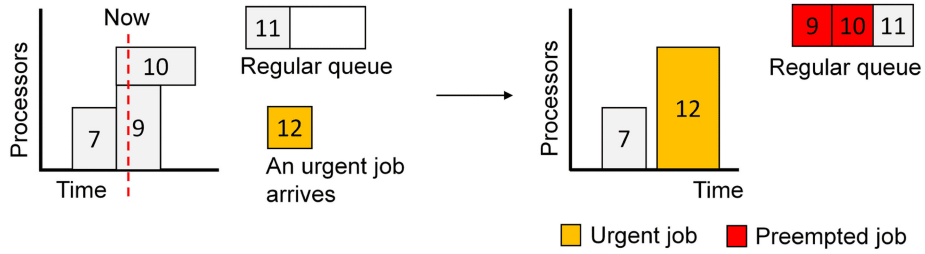


FIGURE 8. Urgent job execution by preemption.

Algorithm 1 The UJFB Algorithm

```
Input: urgentQ                          ▷ Queue of urgent jobs
Input: regularQ                         ▷ Queue of regular jobs
Input: runningJobs                      ▷ Currently running jobs
procedure onArrival(job)
    if job is urgent then               ▷ An urgent job
        if canExecute(job) then         ▷ Resources are available
            if canBackfill(job) then    ▷ Backfilling
                backfill(job)
            else                        ▷ Precede regular jobs
                execute(job)
            end if
        else if canRunWithPreempt(job) then
            pJobs ← selectJobs(runningJobs)
            preempt(pJobs)              ▷ Preemption
            execute(job)
        else
            enqueue(urgentQ, job)
        end if
    else                                ▷ A regular job
        enqueue(regularQ, job)
    end if
end procedure
```

time results to evaluate the performance of the proposed scheduling method on the simulator. We also evaluate the performance with longer swapping times and the workload of a real UC system.

A. PROCESS SWAPPING PERFORMANCE

We evaluate the performance of PPS on an SX-AT system, which consists of an Intel Xeon 6126 processor as a VH and an 8-core VE processor as a coprocessor. As a workload, we execute each application of the NPB on the VE with 8 threads using all cores of the VE. In each application, PPS considers multiple threads from their parent processes because a thread shares the memory space with the parent process. Thus, PPS swaps only the parent processes for each application. The performance results are obtained by measuring the time spent by PPS to swap the application from the VE to the VH.

The performance of process swapping depends on the process size, that is, the amount of memory used by the application process. To evaluate the effects of process swapping on different process sizes, we execute the applications with different input sizes, which are A, B, C, and D classes. However, even with the same input size, the process size can change during the application runtime. Thus, to measure the swapping time, we swap out each application from the VE to the VH at random times during its execution.

Figure 9 shows the swapping time results with different swap sizes. The swap size is the size of swappable memory of each process. Since PPS partially swaps the VE memory, only the swappable memory is swapped from the VE to

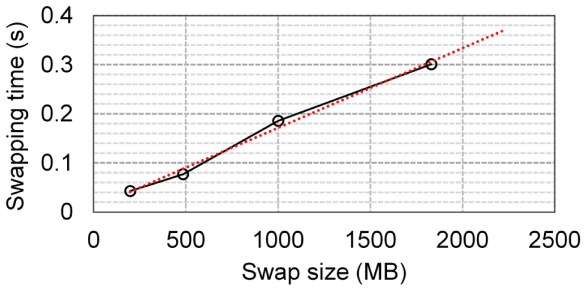


FIGURE 9. Performance of PPS.

the VH. Therefore, the swapping time is significantly affected by the swap size. The swapping times and swap sizes are the averages obtained from the executions with all the NPB applications with each input size. The x-axis represents the swap size, while the y-axis represents the swapping time corresponding to the swap size. We also show the extrapolated values of the swapping times for larger swap sizes. All of the swap sizes and swapping times of NPB applications are listed in the Appendix. The figure shows that the swapping time increases linearly with the swap size. However, for most of the observed swap sizes, the swapping times are less than or equal to 0.3 seconds.

The linear relationship between swapping time and swap size in Figure 9 indicates that the swapping time of each application is primarily affected by swap size. The reason is that the swapping time is dominated by the time needed for the data transfer between the VE and VH. For most of the NPB applications, the total swap size is smaller than the bandwidth of the PCIe interconnect. Thus, no contention occurs on the interconnect while swapping these applications. However, for the NPB applications that have much larger swap sizes, such as FT, the longest swapping time remains below 1 second. These performance results indicate that using PPS can achieve preemption with a negligible delay compared to the deadline at common UC.

B. JOB SCHEDULING PERFORMANCE

To evaluate the job scheduling performance, we run simulations on the Alea simulator and analyze the simulation results using three metrics. The first two metrics are the response time and slowdown. These two metrics are used to evaluate the effects of the job scheduling methods on regular jobs. The third metric is U_L, which is used to evaluate the effects of the job scheduling methods on urgent jobs.

For this evaluation, we randomly inject urgent jobs with the same configuration used in the previous evaluation of Section III. UJFB is compared with four job scheduling algorithms, the urgent job first (UJF), FCFS, backfill, and EASY scheduling algorithms. Backfill corresponds to the conservative backfill algorithm used in the previous evaluation. Thus, UJFB, backfill, and EASY are backfilling-based algorithms. For all these backfilling-based algorithms, the actual runtime is used as the estimated runtime. On the other hand, the UJF algorithm is similar to FCFS. However, it prioritizes urgent jobs to be executed before regular jobs. We implemented UJFB and UJF on the Alea simulator, while the other three algorithms are originally available in the simulator. In addition, we extend the simulator to support preemption.

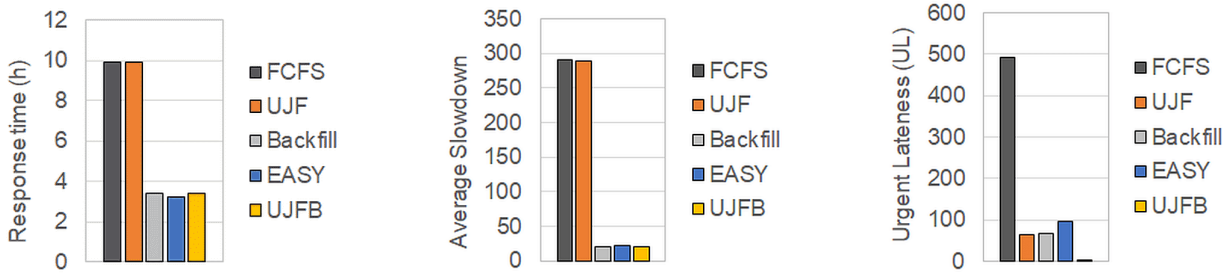
The evaluation uses three workloads of real systems, SDSC-DS, LANL-CM5, and SX-ACE. The first two workloads are obtained from the parallel workloads archive [29], [53], while the SX-ACE workload is obtained from the SX-ACE supercomputer of Tohoku University [32]. The LANL-CM5 workload is composed of 201,387 jobs, which are executed on the CM-5 cluster. This cluster consists of 1,024 nodes, with a total of 1,024 processors. On the other hand, the SX-ACE workload comprises 54,123 jobs, which are executed on the SX-ACE cluster consisting of 1,024 processors.

For the evaluation, the preemption delay caused by preempting a job is obtained by calculating the swapping times for the process sizes of the job using the function derived from the PPS results. All of the workloads do not provide information about the swap size. Thus, the evaluation uses the process size as the swap size of each job. Since no information about the memory usage in the SDSC and SX-ACE workloads is available, we randomly set the process sizes of each job between 500 megabytes and 1.25 gigabytes only for these two workloads. The same random seed is used for all the simulations of each workload. Thus, the simulations with different algorithms use the same swapping time configuration for each job.

The range of process sizes is chosen for three empirical reasons. First, most of the NPB applications have a process size smaller than 1.25 gigabytes. Second, as shown in the LANL workload [54] and three real-world workloads of a related study [55], each of the majority of jobs, not a process, uses no more than 1 gigabyte of memory. In Table 1 of [55], the total percentage of the small memory jobs is much higher than that of the large memory jobs. Third, the results of a previous study [56] show that 10 of the 13 production HPC applications under investigation have process sizes in the range of hundreds of megabytes. In Figure 8 of [56], only three of the applications have a process size larger than 1 gigabyte. Therefore, we can discuss the effects of preemption delay on the SDSC and SX-ACE workloads using this range.

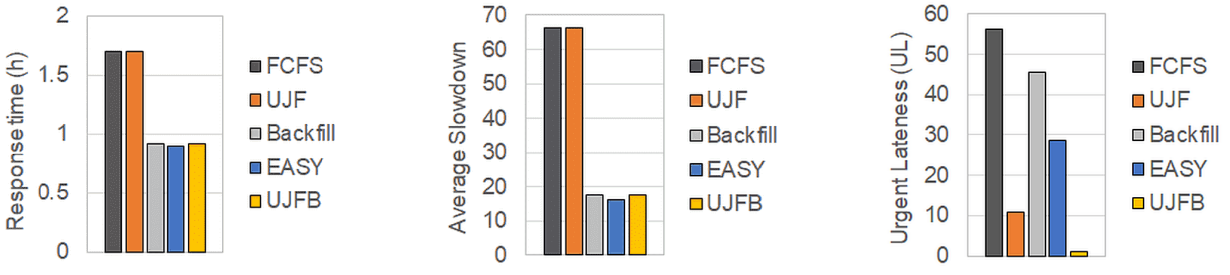
Figure 10 shows the results of the SDSC workload. For regular jobs, the three backfilling-based algorithms show similar response times and slowdowns. On the other hand, FCFS and UJF show much longer response times and slowdowns. This fact indicates that backfilling has a significant impact on reducing the response time and slowdown of regular jobs. Accordingly, UJFB can achieve a comparable response time and slowdown of regular jobs.

Figure 10(c) presents the U_L results of the algorithms. As shown in the figure, all the algorithms except UJFB exhibit significant delays of urgent jobs. UJF shows a lower U_L than FCFS, indicating that prioritizing urgent jobs has a significant impact on reducing the urgent lateness. However, the lateness of UJF is a 62-fold slowdown, which means that it cannot satisfy the deadline requirement of UC. On the other hand,



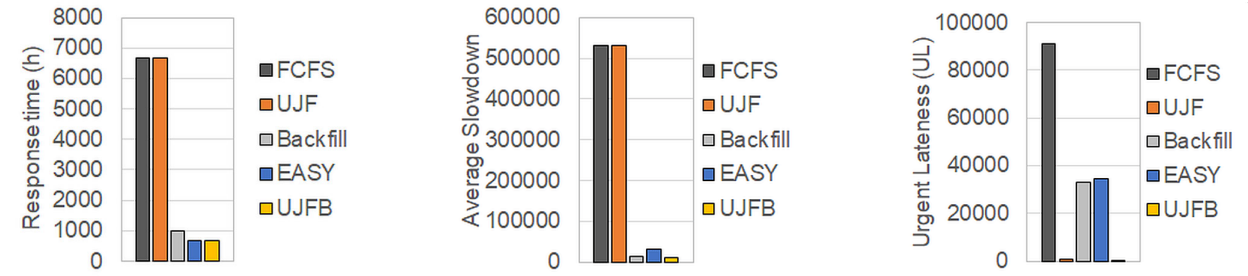
(a) SDSC response time of regular jobs. (b) SDSC slowdown of regular jobs. (c) SDSC U_L.

FIGURE 10. The results of the SDSC workload.



(a) LANL response time of regular jobs. (b) LANL slowdown of regular jobs. (c) LANL U_L.

FIGURE 11. The results of the LANL workload.



(a) SX-ACE response time of regular jobs. (b) SX-ACE slowdown of regular jobs. (c) SX-ACE U_L.

FIGURE 12. The results of the SX-ACE workload.

UJFB shows only a 1.01-fold slowdown of urgent jobs, which is the lowest urgent lateness among the algorithms. For the urgent jobs shown in Table 1, UJFB can satisfy the deadline requirement of tsunami simulations. These results show the effectiveness of the proposed method in preventing the delay of urgent jobs.

Figure 11 shows the results of the LANL workload. The results of this workload are similar to those of the SDSC. FCFS and UJF show a much longer response time and slowdown of regular jobs than those of the three backfill-based algorithms. This fact suggests that backfilling is also effective for reducing the response time and slowdown of regular jobs in this workload. As shown in Figure 11(b), all the algorithms except UJFB show significant delays of urgent jobs. In contrast to the results of the SDSC, UJF shows a lower U_L than those of all other algorithms except UJFB, indicating that UJF can reduce the urgent lateness. Moreover, backfill and EASY show much higher U_L values than that of UJF. This result means that in this workload, backfilling can cause significant delays of urgent jobs. On the other hand, UJFB shows a 1.02-fold slowdown, and thus, it can allow urgent jobs to finish within the deadline.

The results of the SX-ACE workload are shown in Figure 12. The response times and slowdowns of the regular jobs of SX-ACE are higher than those of the SDSC and LANL workloads. We observe that this is because the average number of jobs waiting in the queue of SX-ACE is much higher than those of the other workloads. In this workload, UJFB shows a lower slowdown and U_L, which means that using preemption and backfilling can also reduce the slowdown of regular jobs. By preempting regular jobs, it can execute urgent jobs immediately and backfill more regular jobs during the execution of urgent jobs.

Figure 12(c) shows that UJF and UJFB achieve a much lower U_L than the other algorithms. However, as shown in Table 2, the difference in U_L between these two algorithms

TABLE 2. SX-ACE results for urgent jobs.

Algorithm	U_L	Num. preemption
FCFS	90996.73	0
UJF	641.61	0
Backfill	32853.86	0
EASY	34688.48	0
UJFB	1.05	28

is large. UJF shows a 641-fold slowdown of urgent lateness, while UJFB shows an only 1.05-fold slowdown. These results show that although UJF can achieve a significantly lower U_L, it still cannot allow urgent jobs to finish within the deadline. Moreover, although UJFB preempts regular jobs to execute urgent jobs, it can still achieve a response time and slowdown of regular jobs comparable to those of the other two backfilling-based algorithms. The results of the SX-ACE workload suggest that by using backfilling and preemption, the proposed method can effectively reduce the urgent lateness without increasing the response time and slowdown of regular jobs.

The results of the SDSC, LANL, and SX-ACE workloads show that the proposed method can satisfy the deadline requirement of UC, while the response time and slowdown of regular jobs are unchanged. Accordingly, this fact also demonstrates the effectiveness of PPS in reducing the preemption delays, and thus, UJFB can achieve negligible results of U_L for all the tested workloads.

C. EVALUATION WITH LONGER SWAPPING TIMES

As previously described in Section I, a longer swapping time will increase the preemption delay, and thus it can have significant impacts on the job scheduling performance. To evaluate the effects, we run simulations with the LANL workload and different swapping time configurations. Three ranges of swapping times are used for this evaluation. The first range is the range of swapping times of PPS, which is used in the previous evaluation. The other two ranges are from 1.8 to 40 seconds and from 3.6 to 80 seconds. We use the latter two ranges to simulate the swapping time with slower storage devices, such as in disk-based swapping mechanisms. As shown in previous studies [57], [58], longer times are expected when using disk-based mechanisms. We obtain the latter two ranges by multiplying the range of PPS by factors of 20 and 40. These factors are adopted from the empirical results of the related work [58].

Figure 13 shows the results of the response time, slowdown, and urgent lateness with different swapping time configurations. The results are normalized to the results of PPS. This figure shows similar response times and slowdowns among the swapping time configurations, indicating that longer swapping times do not affect the performance of regular jobs. The reason is that the swapping times are too short compared with the running times of the jobs. However, the U_L results show a significant difference among the swapping time configurations. The figure shows that for the latter two ranges, the U_L value is up to 1.95-fold longer than that of

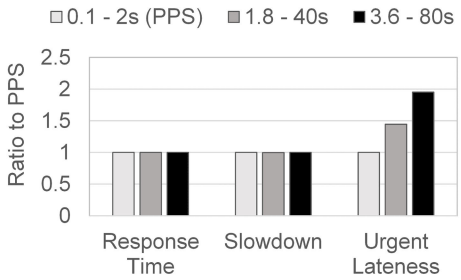


FIGURE 13. The LANL results with longer swapping times.

the range of PPS. For the urgent jobs shown in Table 1, these longer swapping times can prevent the urgent jobs from finishing within the 10-minute deadline. These results show that although longer swapping times barely affect the performance of regular jobs, they can significantly delay urgent jobs. This fact also shows the advantage of using the proposed method PPS, which can achieve negligible preemption delays.

In the proposed PPS, job processes are swapped from the VE to the swap area that is located in the VH memory. PPS needs to preallocate the swap area for the swapping. As a result, this allocation will reduce the memory that is available for processes running on the host processor. Several studies have proposed hybrid memory/disk mechanisms to overcome this limitation [58], [59]. However, the evaluations of the related studies have shown that the hybrid mechanisms incur significant overheads compared with the in-memory mechanisms. These overheads can significantly increase the swapping time. As suggested by the results for the LANL workload, higher swapping times can significantly increase preemption delays and consequently increase the lateness of urgent jobs. Therefore, to avoid significant delays of urgent jobs, this work uses the in-memory swapping mechanism to preempt regular jobs. In practice, an appropriate selection of swapping mechanisms would depend on various factors such as the system operation policy and deadline requirement, i.e., putting a higher priority on either the available memory capacities for regular jobs or the short preemption delay.

D. EVALUATION WITH THE WORKLOAD OF A REAL UC SYSTEM

To evaluate the proposed method on the job scheduling performance of a real UC system, we run simulations with the workload from the UC system at DWD. The workload consists of 13,807 jobs, which are executed on the system on September 17, 2020. Figures 14 and 15 show the frequency distributions of job arrivals and number of nodes required by jobs in the workload. As shown in Figure 14, regular jobs have the highest number of arrivals among the hours. However, compared to the workloads with injected urgent jobs, this workload has a much higher number of arrivals of urgent jobs. At some hours, such as hours 1, 4 and 16, more than 500 urgent jobs were submitted to the system.

Figure 15 shows that urgent jobs require a higher number of nodes than regular jobs. Most of the urgent jobs require

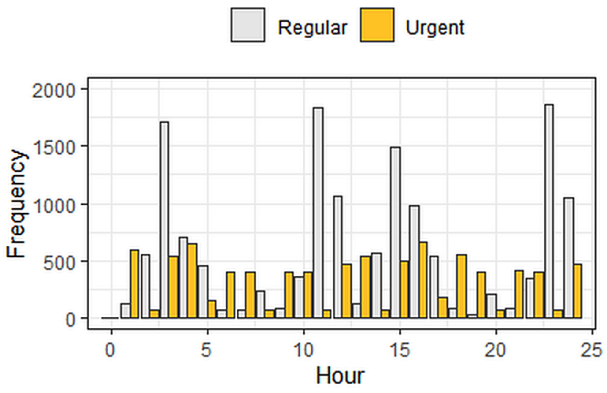


FIGURE 14. Job arrival times.

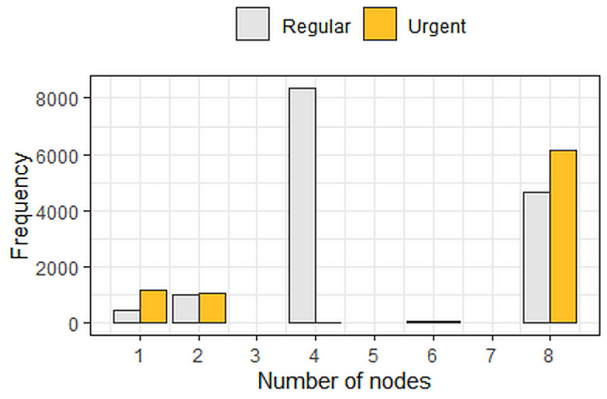


FIGURE 15. Number of nodes required by jobs.

8 nodes for their execution. Thus, during the hours when many urgent jobs are submitted to the system, there is a high possibility that one or more urgent jobs cannot be executed immediately due to insufficient resources. This possibility becomes higher when there are also regular jobs running on the system. Thus, preventing urgent jobs from being delayed by the regular jobs becomes more important.

Figure 16 shows the simulation results with the DWD workload. These results are obtained from the simulations of a system with a total of 1,024 nodes. We have also conducted simulations with more nodes. However, we observed that the slowdown results of the job scheduling algorithms are small with larger simulated systems, indicating that these systems are too large for the workload. Thus, we only show

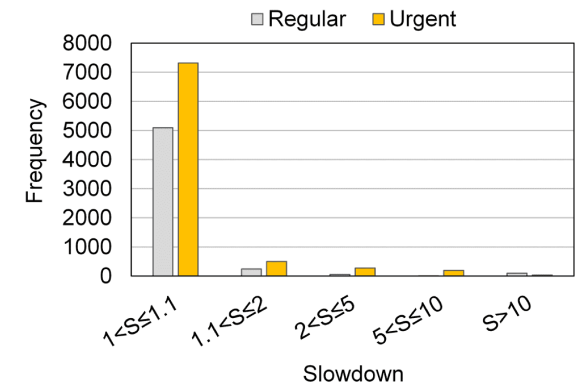
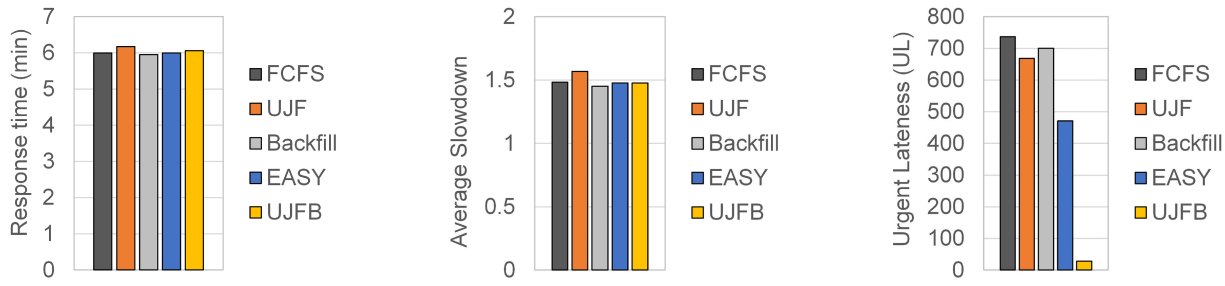


FIGURE 17. DWD slowdowns with UJFB.

the simulation results with the 1024-node system to discuss the effects of the proposed method on the DWD workload.

Figures 16(a) and 16(b) show similar response times and slowdowns of regular jobs among the algorithms. Moreover, the results of these two metrics are much lower than those of the SDSC, LANL, and SX-ACE workloads. These results indicate that the simulated system is far from being a bottleneck for regular jobs. However, Figure 16(c) shows much higher values of U_L, indicating that the effects of urgent jobs on the job scheduling performance are much higher than those of regular jobs. As shown in Figure 15, this is because urgent jobs demand more resources than regular jobs. Accordingly, the U_L result of UJFB in this workload is higher than those of the SDSC, LANL, and SX-ACE workloads, which is because some urgent jobs need to wait in the urgent-queue. However, as shown in the U_L results, UJFB can still achieve the smallest lateness of urgent jobs, suggesting that the proposed method remains effective in reducing delays of urgent jobs.

To further investigate the effects of UJFB on the urgent lateness, we analyze the frequency distribution of the slowdowns of all jobs, which is shown in Figure 17. In this figure, the x-axis represents the ranges of the slowdowns, while the y-axis represents the frequencies of the slowdowns of all jobs. The figure shows that most of the jobs have small slowdowns, that is, a less than 1.1-fold slowdown. For the higher ranges of slowdowns, the frequencies are low and decreasing with the higher ranges. For urgent jobs, only 5.9% of all jobs show slowdowns higher than 2, which explains the results



(a) DWD response time of regular jobs. (b) DWD slowdown of regular jobs. (c) DWD U_L.

FIGURE 16. The results of DWD workload.

TABLE 3. Swapping time results of the NPB applications.

Input Size (Class)	Application	Avg. Swap Size (Megabytes)	Avg. S_p (Seconds)
A	BT	118.531	0.022
	CG	123.727	0.023
	EP	76.020	0.014
	FT	396.707	0.073
	IS	133.559	0.024
	LU	110.238	0.073
	MG	506.758	0.093
	SP	120.523	0.022
B	BT	248.621	0.040
	CG	512.705	0.082
	EP	76.020	0.012
	FT	1630.167	0.259
	IS	331.559	0.053
	LU	215.742	0.034
	MG	608.109	0.097
	SP	256.770	0.041
C	BT	573.680	0.106
	CG	741.656	0.138
	EP	76.023	0.014
	FT	2602.986	0.676
	IS	842.672	0.156
	LU	633.914	0.118
	MG	1741.941	0.323
	SP	797.586	0.148
D	BT	1223.390	0.201
	CG	1071.495	0.176
	EP	76.023	0.013
	FT	5127.443	0.844
	IS	2116.223	0.348
	LU	1116.267	0.184
	MG	2469.829	0.406
	SP	1441.358	0.237

in Figure 16(c) that UJFB can achieve the lowest U_L among the algorithms. For weather forecast simulations, UJFB can satisfy the deadline requirement for most of urgent jobs. These results also demonstrate the effectiveness of the proposed method in reducing slowdowns of urgent and regular jobs in the DWD workload.

VI. CONCLUSIONS AND FUTURE WORK

Due to the limitations of dedicated infrastructures, use of shared infrastructures becomes an alternative solution for supporting UC. However, in shared infrastructures, it is important not only to avoid delaying urgent jobs but also to prevent increasing the response time and slowdown of regular jobs. In this paper, we proposed a parallel job scheduling method for shared infrastructures supporting UC. It consists of an in-memory process swapping mechanism for heterogeneous systems and a preemption-based backfill scheduling algorithm. The process swapping mechanism is employed to reduce the delays caused by preempting regular jobs. On the other hand, the scheduling algorithm is proposed to prevent delaying urgent jobs while reducing the response time and slowdown of regular jobs. In addition, we propose a metric for evaluating the effects of job scheduling methods on the lateness of urgent jobs.

The process swapping mechanism has been evaluated on a heterogeneous system with vector processors, called SX-AT. The results show that the process swapping mechanism can achieve negligible swapping times, indicating the importance of this mechanism in reducing the preemption delay. The job scheduling algorithm has been evaluated on a simulation environment with three workloads of real systems. It has been compared with four conventional job scheduling algorithms. The simulation results show that the proposed algorithm can achieve almost no lateness of urgent jobs without significant increases in the response time and slowdown of regular jobs. The evaluation with larger swapping times suggests that minimizing the preemption delay is crucial in shared infrastructures supporting UC. In addition, the proposed method has been evaluated with the workload of a real UC system at DWD. The evaluation results demonstrate the effectiveness of the method in reducing the slowdowns of regular and urgent jobs when urgent jobs frequently arrive.

One important future work is to use the proposed method for the operation of a real system and to establish the operation policy. We are now planning to apply the method to actual UC operation of the tsunami simulation system at the Cyberscience Center, Tohoku University.

APPENDIX
SWAPPING TIME RESULTS

See Table 3.

ACKNOWLEDGMENT

The authors would like to thank Yasuhisa Masaoka, Yoko Isobe, and Tatsuyoshi Ohmura of NEC for valuable discussions.

REFERENCES

[1] P. Beckman, S. Nadella, N. Trebon, and I. Beschastnikh, "SPRUCE: A system for supporting urgent high-performance computing," in *Grid-Based Problem Solving Environments*, P. W. Gaffney and J. C. T. Pool, Eds. Boston, MA, USA: Springer, 2007, pp. 295–311.

[2] S. H. Leong and D. Kranzlmüller, "Towards a general definition of urgent computing," *Procedia Comput. Sci.*, vol. 51, pp. 2337–2346, Jan. 2015.

[3] Y. Fujinawa and Y. Noda, "Japan's earthquake early warning system on 11 March 2011: Performance, shortcomings, and changes," *Earthq. Spectra*, vol. 29, no. 1, pp. 341–368, Mar. 2013.

[4] B. Blanton, J. McGee, J. Fleming, C. Kaiser, H. Kaiser, H. Lander, R. Luettich, K. Dresback, and R. Kolar, "Urgent computing of storm surge for North Carolina's coast," *Procedia Comput. Sci.*, vol. 9, pp. 1677–1686, Jan. 2012.

[5] N. Uchida, K. Takahata, and Y. Shibata, "Disaster information system from communication traffic analysis and connectivity (quick report from Japan Earthquake and Tsunami on March 11th, 2011)," in *Proc. 14th Int. Conf. Netw.-Based Inf. Syst.*, Sep. 2011, pp. 279–285.

[6] S. H. Leong, A. Frank, and D. Kranzlmüller, "Leveraging e-infrastructures for urgent computing," *Procedia Comput. Sci.*, vol. 18, pp. 2177–2186, Jan. 2013.

[7] E. Strohmaier, H. W. Meuer, J. Dongarra, and H. D. Simon, "The TOP500 list and progress in high-performance computing," *Computer*, vol. 48, no. 11, pp. 42–49, Nov. 2015.

[8] K. Sajjapongse, X. Wang, and M. Becchi, "A preemption-based runtime to efficiently schedule multi-process applications on heterogeneous clusters with GPUs," in *Proc. 22nd Int. Symp. High-Perform. Parallel Distrib. Comput. (HPDC)*. New York, NY, USA: Association for Computing Machinery, 2013, p. 179.

[9] J. M. Smith, "A survey of process migration mechanisms," *ACM SIGOPS Operating Syst. Rev.*, vol. 22, no. 3, pp. 28–40, Jul. 1988.

[10] K. Komatsu, S. Momose, Y. Isobe, O. Watanabe, A. Musa, M. Yokokawa, T. Aoyama, M. Sato, and H. Kobayashi, "Performance evaluation of a vector supercomputer SX-aurora TSUBASA," in *Proc. SC Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2018, pp. 685–696.

[11] O. Arndt, B. Freisleben, T. Kielmann, and F. Thilo, "A comparative study of online scheduling algorithms for networks of workstations," *Cluster Comput.*, vol. 3, no. 2, p. 95–112, Apr. 2000.

[12] P. J. Keleher, D. Zotkin, and D. Perkovic, "Attacking the bottlenecks of backfilling schedulers," *Cluster Comput.*, vol. 3, no. 4, pp. 245–254, Oct. 2000.

[13] D. B. Jackson, Q. Snell, and M. J. Clement, "Core algorithms of the Maui scheduler," in *Proc. Revised Papers 7th Int. Workshop Job Scheduling Strategies Parallel Process. (JSSPP)*. Berlin, Germany: Springer-Verlag, 2001, p. 87–102.

[14] A. W. Mu'alem and D. G. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 6, pp. 529–543, Jun. 2001.

[15] D. G. Feitelson and A. M. Weil, "Utilization and predictability in scheduling the IBM SP2 with backfilling," in *Proc. 1st Merged Int. Parallel Process. Symp. Symp. Parallel Distrib. Process.*, 1998, pp. 542–546.

[16] D. Tsafrir, Y. Etsion, and D. G. Feitelson, "Backfilling using system-generated predictions rather than user runtime estimates," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 6, pp. 789–803, Jun. 2007.

[17] T. N'takpé and F. Suter, "Don't hurry be happy: A deadline-based backfilling approach," in *Job Scheduling Strategies for Parallel Processing*, D. Klusáček, W. Cirne, and N. Desai, Eds. Cham, Switzerland: Springer, 2018, pp. 62–82.

[18] S. Shin, Y. Kim, and S. Lee, "Deadline-guaranteed scheduling algorithm with improved resource utilization for cloud computing," in *Proc. 12th Annu. IEEE Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2015, pp. 814–819.

[19] H. Takizawa, K. Sato, K. Komatsu, and H. Kobayashi, "CheCUDA: A checkpoint/restart tool for CUDA applications," in *Proc. Int. Conf. Parallel Distrib. Comput., Appl. Technol.*, Dec. 2009, pp. 408–413.

[20] H. Takizawa, K. Koyama, K. Sato, K. Komatsu, and H. Kobayashi, "CheCL: Transparent checkpointing and process migration of OpenCL applications," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, May 2011, pp. 864–876.

[21] A. Vaishnav, K. D. Pham, and D. Koch, "A survey on FPGA virtualization," in *Proc. 28th Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2018, pp. 131–1317.

[22] A. Vaishnav, K. D. Pham, and D. Koch, "Heterogeneous resource-elastic scheduling for CPU+FPGA architectures," in *Proc. 10th Int. Symp. Highly-Efficient Accel. Reconfigurable Technol. (HEART)*. New York, NY, USA: Association for Computing Machinery, 2019, pp. 1–6, doi: 10.1145/3337801.3337819.

[23] D. Koch, C. Haubelt, and J. Teich, "Efficient hardware checkpointing: Concepts, overhead analysis, and implementation," in *Proc. ACM/SIGDA 15th Int. Symp. Field Program. Gate Arrays (FPGA)*. New York, NY, USA: Association for Computing Machinery, 2007, p. 188, doi: 10.1145/1216919.1216950.

[24] M. Happe, A. Traber, and A. Keller, "Preemptive hardware multitasking in ReconOS," in *Applied Reconfigurable Computing*, K. Sano, D. Soudris, M. Hübner, and P. C. Diniz, Eds. Cham, Switzerland: Springer, 2015, pp. 79–90.

[25] A. Vaishnav, K. D. Pham, D. Koch, and J. Garside, "Resource elastic virtualization for FPGAs using OpenCL," in *Proc. 28th Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2018, pp. 111–1117.

[26] D. G. Feitelson, "Metrics for parallel job scheduling and their convergence," in *Proc. 7th Int. Workshop Job Scheduling Strategies Parallel Process. (JSSPP)*. Berlin, Germany: Springer-Verlag, 2001, pp. 188–206.

[27] C. Gómez-Martín, M. A. Vega-Rodríguez, and J.-L. González-Sánchez, "Fattened backfilling: An improved strategy for job scheduling in parallel systems," *J. Parallel Distrib. Comput.*, vol. 97, pp. 69–77, Nov. 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731516300788

[28] C. Barberato, P. E. Strazdins, E. McCreath, and M. Atif, "Efficient evaluation of scheduling metrics using emulation: A case study in the effect of artefacts," in *Proc. 47th Int. Conf. Parallel Process. Companion*. New York, NY, USA: Association for Computing Machinery, Aug. 2018, pp. 1–10, doi: 10.1145/3229710.3229751.

[29] (2020). *Parallel Workloads Archive*. Accessed: May 2020. [Online]. Available: https://www.cs.huji.ac.il/labs/parallel/workload

[30] D. Klusáček, V. Tóth, and G. Podolníková, "Complex job scheduling simulations with Alea 4," in *Proc. 9th EAI Int. Conf. Simulation Tools Techn. (SIMUTOOLS)*. Brussels, Belgium: ICST, 2016, pp. 124–129.

[31] R. Buyya and M. Murshed, "GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *Concurrency Comput., Pract. Exper.*, vol. 14, nos. 13–15, pp. 1175–1220, Nov. 2002.

[32] A. Musa, O. Watanabe, H. Matsuoka, H. Hokari, T. Inoue, Y. Murashima, Y. Ohta, R. Hino, S. Koshimura, and H. Kobayashi, "Real-time tsunami inundation forecast system for tsunami disaster prevention and mitigation," *J. Supercomput.*, vol. 74, no. 7, pp. 3093–3113, Jul. 2018, doi: 10.1007/s11227-018-2363-0.

[33] F. Lovholt, S. Lorito, J. Macias, M. Volpe, J. Selva, and S. Gibbons, "Urgent tsunami computing," in *Proc. IEEE/ACM HPC Urgent Decis. Making (UrgentHPC)*, Nov. 2019, pp. 45–50.

[34] T. Mukherjee, A. Banerjee, G. Varsamopoulos, S. K. S. Gupta, and S. Rungta, "Spatio-temporal thermal-aware job scheduling to minimize energy consumption in virtualized heterogeneous data centers," *Comput. Netw.*, vol. 53, no. 17, pp. 2888–2904, Dec. 2009.

[35] J. Lelong, V. Reis, and D. Trystram, "Tuning EASY-backfilling queues," in *Job Scheduling Strategies for Parallel Processing*, D. Klusáček, W. Cirne, and N. Desai, Eds. Cham, Switzerland: Springer, 2018, pp. 43–61.

[36] G. Blewitt, C. Kreemer, W. C. Hammond, H. Plag, S. Stein, and E. Okal, "Rapid determination of earthquake magnitude using GPS for tsunami warning systems," *Geophys. Res. Lett.*, vol. 33, no. 11, pp. 1–4, Jun. 2006.

[37] G. Berry, "Preemption in concurrent systems," in *Foundations of Software Technology and Theoretical Computer Science*, R. K. Shyamasundar, Ed. Berlin, Germany: Springer, 1993, pp. 72–93.

[38] Q. O. Snell, M. J. Clement, and D. B. Jackson, "Preemption based backfill," in *Job Scheduling Strategies for Parallel Process.*, D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Berlin, Germany: Springer, 2002, pp. 24–37.

[39] N. R. Nielsen, "An analysis of some time-sharing techniques," *Commun. ACM*, vol. 14, no. 2, p. 79–90, Feb. 1971.

[40] B. Furht and V. Milutinovic, "A survey of microprocessor architectures for memory management," *Computer*, vol. 20, no. 3, pp. 48–67, Mar. 1987.

[41] T. Newhall, S. Finney, K. Ganchev, and M. Spiegel, "Nswap: A network swapping module for Linux clusters," in *Euro-Par 2003 Parallel Processing*, H. Kosch, L. Böszörményi, and H. Hellwagner, Eds. Berlin, Germany: Springer, 2003, pp. 1160–1169.

[42] J. Kehne, J. Metter, and F. Bellosa, "GPUswap: Enabling oversubscription of GPU memory through transparent swapping," *SIGPLAN Not.*, vol. 50, no. 7, p. 65–77, Mar. 2015.

[43] J. Gu, Y. Lee, Y. Zhang, M. Chowdhury, and K. G. Shin, "Efficient memory disaggregation with Infiniswap," in *Proc. 14th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*. Boston, MA, USA: USENIX Association, Mar. 2017, pp. 649–667. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/gu

[44] G. Lee, W. Jin, W. Song, J. Gong, J. Bae, T. J. Ham, J. W. Lee, and J. Jeong, "A case for hardware-based demand paging," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit. (ISCA)*, May 2020, pp. 1103–1116.

[45] R. Espasa, M. Valero, and J. E. Smith, "Out-of-order vector architectures," in *Proc. 30th Annu. Int. Symp. Microarchitecture*, 1997, pp. 160–170.

[46] NEC. (2016). *SX-Aurora TSUBASA Installation Guide*. [Online]. Available: https://www.hpc.nec/documents/guide/pdfs/InstallationGuide_E.pdf

[47] NEC. (2020). *VEOS Partial Process Swapping Reference*. [Online]. Available: https://veos-sxarr-nec.github.io/doc/VEOS_Partial_Process_Swapping_Reference.txt

[48] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard Version 3.1*. Accessed: Jun. 2015. [Online]. Available: http://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf

[49] M. Yokokawa, A. Nakai, K. Komatsu, Y. Watanabe, Y. Masaoka, Y. Isobe, and H. Kobayashi, "I/O performance of the SX-aurora TSUBASA," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2020, pp. 27–35.

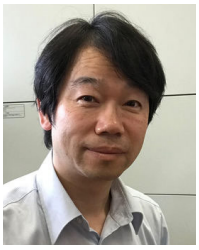
[50] C. Wang, F. Mueller, C. Engelmann, and S. L. Scott, "Proactive process-level live migration in HPC environments," in *Proc. SC Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2008, pp. 1–12.
[51] D. Cederman and P. Tsigas, "GPU-quicksort: A practical quicksort algorithm for graphics processors," *ACM J. Experim. Algorithmics*, vol. 14, pp. 1–4, Dec. 2009, doi: 10.1145/1498698.1564500.
[52] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, and H. D. Simon, "The NAS parallel benchmarks," *Int. J. High Perform. Comput. Appl.*, vol. 5, no. 3, pp. 63–73, 1991.
[53] D. G. Feitelson, D. Tsafrir, and D. Krakov, "Experience with using the parallel workloads archive," *J. Parallel Distrib. Comput.*, vol. 74, no. 10, pp. 2967–2982, Oct. 2014.
[54] A. Sodan, "Predictive space-and time-resource allocation for parallel job scheduling in clusters, grids, clouds," in *Proc. 39th Int. Conf. Parallel Process. Workshops*, Sep. 2010, pp. 313–322.
[55] X. Li, N. Qi, Y. He, and B. McMillan, "Practical resource usage prediction method for large memory jobs in HPC clusters," in *Supercomputing Frontiers*, D. Abramson and B. R. de Supinski, Eds. Cham, Switzerland: Springer, 2019, pp. 1–18.
[56] D. Zivanovic, M. Pavlovic, M. Radulovic, H. Shin, J. Son, S. A. Mckee, P. M. Carpenter, P. Radojković, and E. Ayguadé, "Main memory in HPC: Do we need more or could we live with less?" *ACM Trans. Archit. Code Optim.*, vol. 14, no. 1, pp. 1–26, Apr. 2017, doi: 10.1145/3023362.
[57] G. Zheng, L. Shi, and L. V. Kale, "FTC-Charm++: An in-memory checkpoint-based fault tolerant runtime for Charm++ and MPI," in *Proc. IEEE Int. Conf. Cluster Comput.*, Sep. 2004, pp. 93–103.
[58] X. Dong, Y. Xie, N. Muralimanohar, and N. P. Jouppi, "Hybrid checkpointing using emerging nonvolatile memories for future exascale systems," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 2, pp. 1–29, Jul. 2011.
[59] A. Badam and V. S. Pai, "SSDAlloc: Hybrid SSD/RAM memory management made easy," in *Proc. 8th USENIX Conf. Netw. Syst. Design Implement. (NSDI)*. New York, NY, USA: USENIX Association, 2011, pp. 211–224.

MULYA AGUNG received the master's degree from the Bandung Institute of Technology, in 2015, and the Ph.D. degree from Tohoku University, in 2020. He is currently a Postdoctoral Researcher at the Cyberscience Center, Tohoku University. His current work focuses on data science and high-performance computing.

YUTA WATANABE is currently a Principal Engineer at the AI Platform Division, NEC Corporation. He is responsible for the development of VEOS, which provides operating system features for SX-Aurora TSUBASA.

HENNING WEBER was born in Frankfurt, Germany, in 1970. He received the Ph.D. degree in theoretical physics from Goethe University, Frankfurt, in 2002. From 1997 to 2002, he was a Research Assistant at the Institute for Theoretical Physics, University of Frankfurt. Since 2003, he has been working for the National Meteorological Service of Germany (DWD), where he is currently the Head of IT, responsible for time-critical operations and the HPC center.

RYUSUKE EGAWA (Member, IEEE) received the B.E. and master's degrees in information sciences from Hirosaki University, in 1999 and 2001, respectively, and the Ph.D. degree in information sciences from Tohoku University, in 2004. He is currently a Professor at Tokyo Denki University. His research interests include computer architecture and high-performance computing. He is a member of the IEEE CS, IEICE, and IPSJ.



HIROYUKI TAKIZAWA received the B.E. degree in mechanical engineering and the M.S. and Ph.D. degrees in information sciences from Tohoku University, in 1995, 1997, and 1999, respectively. He is currently a Professor at the Cyberscience Center, Tohoku University. His research interests include high-performance computing systems and their applications. He is a member of the IEEE CS, ACM SIGHPC, IEICE, and IPSJ.