Effectiveness of the Oversubscribing Scheduling on Supercomputer Systems

SHOHEI MINAMI, Prometech Software, Inc., Japan and Tokyo Institute of Technology, Japan
TOSHIO ENDO and AKIHIRO NOMURA, Tokyo Institute of Technology, Japan

High responsiveness is substantial for users' satisfaction in supercomputer systems. Recently, the use of interactive jobs in addition to traditional batch jobs is attracting attention. It is getting important to handle those jobs consolidated for responsive systems. Here we show oversubscribing scheduling, in which multiple HPC jobs share computational resources, can effectively process jobs. This paper builds the job scheduling simulator considering oversubscribing and evaluates the oversubscribing system using actual supercomputer workload trace data. While keeping the short users' response time, our solution achieves some strengths not found in a conventional solution; benefits on normal jobs, alleviating the slowdown, and the unnecessariness of the effort of good system configuration.

CCS Concepts: • **Software and its engineering → Massively parallel systems**; • **Theory of computation → Scheduling algorithms**.

Additional Key Words and Phrases: Job Scheduling, Oversubscription, Simulator, Short-running jobs

ACM Reference Format:
Shohei Minami, Toshio Endo, and Akihiro Nomura. 2023. Effectiveness of the Oversubscribing Scheduling on Supercomputer Systems. 1, 1 (May 2023), 21 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Recently, the real-time usage of supercomputing resources is getting more significant due to the recent spread of machine-learning and big-data applications on supercomputers. For such purposes, typical users may make data analysis code, execute computation on supercomputers, observe the results and then modify the code again. During the development loop, high responsiveness of short-running jobs or interactive usage is critical. Hereafter, we call those jobs high-responsiveness-requesting jobs (*HRR jobs*). On a supercomputer with a traditional scheduling configuration, however, users of HRR jobs may suffer from long waiting time. If the system is already filled with long-running jobs that take more than one day, the submitter of the short job may observe the waiting time of several days.

This issue has been partly alleviated by several ways, such as backfilling scheduling algorithm[10], node partitioning for multiple jobs[11] and introducing dedicated nodes for HRR or interactive jobs[1, 11]. Unfortunately, they tend to be insufficient especially when a system has high utilization ratio.

From the above discussion, this paper focuses on more aggressive way to improve responsiveness, *oversubscribing scheduling*, with which multiple jobs can coexist on the same CPU cores[8]. While the idea of time-sharing of CPU cores is common and supported by well-known schedulers such as Slurm[16], most of production supercomputers avoid oversubscription. One of main reasons is that jobs suffer from speed down, which become even heavier for parallel jobs.

Authors' addresses: Shohei Minami, minami@prometech.co.jp, Prometech Software, Inc., Bunkyo, Tokyo, Japan and Tokyo Institute of Technology, Meguro, Tokyo, Japan; Toshio Endo, endo@is.titech.ac.jp; Akihiro Nomura, nomura@gsic.titech.ac.jp, Tokyo Institute of Technology, Meguro, Tokyo, Japan.

Not only the extension of running time itself is problematic, also it may break the assumption of backfilling scheduling, backfilled job must finish untill the head-of-queue job starts, and cause failure of the scheduling.

Towards the above issues, we take the following approaches.

- During oversubscribing scheduling, we control the *multiplicity*, the number of jobs sharing the same cores, for each core in order to alleviate speed down of parallel jobs.
- We describe our scheduling simulator, named *node conscious oversubscribing scheduler simulator* (NCS).
- We revise backfilling scheduling algorithm to consider speed down by oversubscription.
- We evaluate jobs' behavior with oversubscribing using publicly available supercomputer workload traces.

With these approaches, we demonstrate oversubscribing scheduling largely improve responsiveness of HRR jobs, which makes the usage of supercomputers much more attractive for interactive or short jobs users, without sacrificing long-running jobs largely.

2 OVERSUBSCRIBING ON SUPERCOMPUTER SYSTEMS

Several production systems already use a sort of oversubscribing in a conservative style, called node partitioning[11]. With node partitioning, a single node can be shared by multiple jobs, while each processor (physical or logical) core is still dedicated to a single job. It is useful to improve resource utilization ratio and reduce job waiting time compared with systems without any partitioning. However, when almost all cores in the system are busy, jobs suffer from long waiting time, critical for short or interactive jobs.

From the above discussion, we prefer to introduce oversubscribing more aggressively. This paper focuses on *processor core-level oversubscribing*, hereafter called oversubscribing simply, which allows each core to be shared by multiple jobs. Oversubscribing can improve the responsiveness of jobs since they can be started even when there are not enough idle resources. Additionally, the total CPU utilization of the system can be improved with interactive jobs with fluctuating CPU utilization.

Conventional production supercomputers, however, do not usually adopt this strategy. We consider this is mainly for the following issues.

(1) Each job may suffer from performance degradation for other jobs coexist on the same cores.
(2) The performance degradation can be even worse with parallel jobs.
(3) Demands for the main memory capacity increase.

Hereafter we describe our proposed policies to make oversubscribing more controllable and practical, and some assumptions used in our simulation evaluation.

In order to discuss the issue (1), we define the *multiplicity* m for each core in the system as the number of jobs that are sharing the core. Apparently the speed of a running job, the ratio of processor core timeslice for the job, is degraded according to m of the cores used by the job. In our simulation, we make a simple assumption for the speed down; if the multiplicity of a core is m, we assume the performance of a sequential job on the core becomes $1/m$ of the original job speed without oversubscribing [1].

Our basic policy to mitigate the speed down is to introduce a parameter named *maximum multiplicity* M, configured by the system administrator. The oversubscribing scheduler controls jobs so that multiplicity m does not exceed M at

[1]Actually speed of jobs are affected by many factors, including cache pollution, synchronization within parallel jobs, and so on. A report has analyzed this performance degradation in detail[12]

any core. Note that since we avoid "infinite" oversubscription, jobs may still experience waiting. The evaluation of this paper includes investigation of the trade-off introduced by M.

Related to the issue (2), we discuss slow down of parallel jobs. Since threads of a parallel job are placed on different cores (that may be on different nodes), those threads may suffer from different multiplicity. In our simulation, we assume that the speed down of the entire job is determined by its slowest thread. For example, let us consider a parallel running job A, which occupies 10 cores. Then a serial job B is started on one of those cores. Although the resource available for the job A is 95%, the speed of A may degrade to around 50%. The above assumption is based on a consideration that threads in job A may use synchronization and communication heavily. Actually, the speed down could be more modest if the job is tolerant to heterogeneous core speeds. Anyway, our simulation uses this pessimistic assumption.

We also mention the issue (3), the increase in memory usage. Our current policy is to avoid oversubscribing of memory capacity to avoid page swapping costs. Combined with the above discussion related to M, a new coming job is suspended if (a) there is no sufficient cores that have multiplicity less than M, or (b) there is no node that have sufficient free memory. While detailed investigation of memory usage is our future work, Shalf's report[15], which describes that 50% of jobs at NERSC HPC center use $<$ 20% of the node memory, supports the feasibility of oversubscribing.

In Section 3 and later, we describe the scheduling and simulation method with core-level oversubscribing with the above-mentioned policies and assumptions. Although our assumptions includes pessimistic ones, we demonstrate that oversubscribing largely improves the responsiveness of jobs.

3 METHODOLOGY FOR SIMULATING OVERSUBSCRIBING SCHEDULING

The purpose of this paper is to demonstrate the benefits of oversubscribing scheduling with our policy in the previous section. For this purpose, we conduct experiments using our new simulator, node conscious oversubscribing scheduler simulator (NCS). NCS is designed to simulate oversubscribing job scheduling considering speed down of jobs considering multiplicity. The source code is available at the author's Bitbucket site[3][2].

NCS takes a configuration file for system description and an SWF file[5, 7] for information of a set of jobs as inputs. The configuration file contains the number of nodes, number of cores per node, and memory capacity per node. It also contains maximum multiplicity M and a scheduling policy (Section 3.3).

Then NCS simulates the scheduling of jobs described in the SWF file. Here oversubscribing is allowed if multiplicity does not exceed the maximum multiplicity M at any cores. During the simulation, unlike the typical scheduling simulators, NCS needs to simulate performance degradation of jobs under oversubscribing.

In the rest of this section, we show our system model in the simulation and basic scheduling method in Section 3.1. Then we discuss how the performance degradation due to oversubscribing is estimated (Section 3.2). And then we describe the detailed scheduling algorithm of NCS, which includes modification of a well-known algorithm, EASY Backfilling to support oversubscribing (Section 3.3).

3.1 System Model and Basic Scheduling

Hereafter, we use the notation such as *N16C8* (capitalized) to indicate a system with 16 nodes, each of which has 8 cores. The notation *n2c8* corresponds to a job that requests 2 nodes in total and occupies 8 cores in each node.

[2]This repository is anonymized for the double-blinded review process

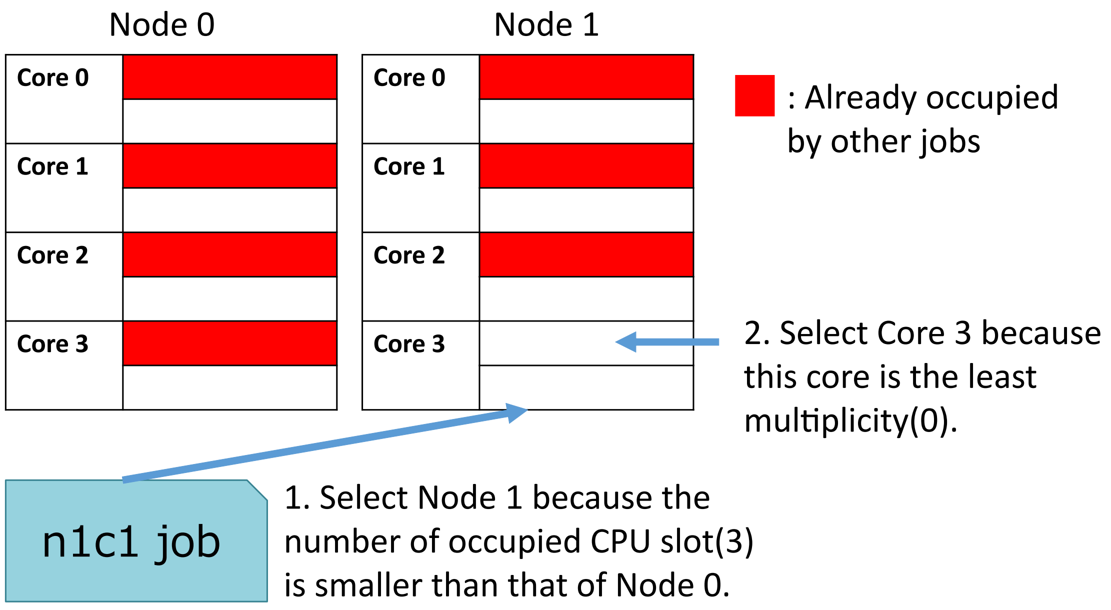

Node 0
Node 1
Core 0
Core 1
Core 2
Core 3
: Already occupied by other jobs
2. Select Core 3 because this core is the least multiplicity(0).
n1c1 job
1. Select Node 1 because the number of occupied CPU slot(3) is smaller than that of Node 0.

Fig. 1. An example of oversubscribing scheduling on a N2C4 system. Two slots are prepared per core since the maximum multiplicity M is two.

NCS internally maintains the status of the target system during the simulation as shown in Figure 1, which shows an example of $N2C4$ system with $M = 2$. Each core has $M = 2$ slots used to maintain running jobs on the core. Here red slots have been occupied by running jobs.

When NCS takes a new job submission, it needs to decide on nodes and cores for the job. This allocation is done in a hierarchical style as follows. In Figure 1, NCS is going to schedule a new n1c1 (serial) job. First, NCS determines an appropriate node for the job, which has sufficient empty slots and free memory. If there are several candidates for nodes, NCS consider the total occupied slots per node, and selects a node with the least occupied slots, which is Node 1 in the figure with 3 total occupied slots. Then NCS determines a core with the least occupied slots, Core 3. For parallel jobs, NCS determines nodes and cores similarly.

If the above process fails, since there are not enough slots for the new coming job, it is put in the waiting queue. Jobs in the queue are examined later when the status of slots is changed (Section 3.3).

3.2 Simulation of Job Progress

Unlike typical job scheduling simulators, NCS for oversubscribing scheduling has to consider jobs' performance degradation. Here we are based on assumptions that degradation is determined by multiplicity m as described in Section 2. We also need to consider that m on each core changes dynamically.

Let us explain the behavior of NCS using a small system of N1C4 and $M = 2$ as shown in Figure 2. We consider two jobs, job0 = n1c4 and job1 = n1c2. Also, job0 is submitted at $t = 0$ and the original execution time, included in the input data, is 30. Job1 is submitted at $t = 10$ and the original execution time is 10.

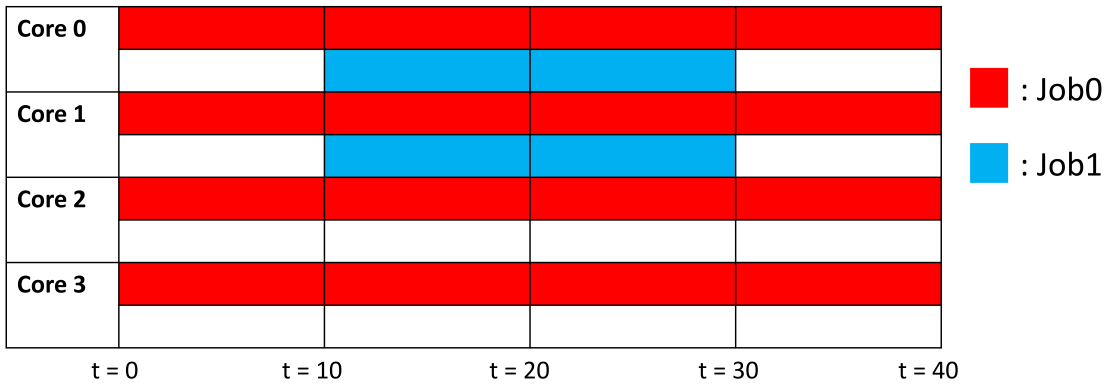


Fig. 2. An example of time-series of oversubscribing scheduling. The speeds of Job0 and Job1 are degraded while they share CPU cores.

- $t = 0$: Job0 is submitted and starts immediately.
- $0 < t < 10$: Job0 uses four dedicated cores. During this period, job0 does not suffer from degradation.
- $t = 10$: Job1 is submitted and starts immediately because there is room in the CPU core slots. Here cores 0 and 1 are used.
- $10 < t < 30$: Both job0 and job1 share cores 0 and 1. Since we assume that the performance of each job is determined by the slowest threads, the performance of job0 and job1 becomes 1/2 of the original. Although cores 2 and 3 are used only by job0, the speed of entire job0 is halved.
- $t = 30$: Job1 finishes at this time, taking $10/(1/2) = 20$ time considering performance degradation.
- $30 < t \leq 40$: Job0 uses four dedicated cores again and the performance is recovered. Job0 finishes at $t = 40$.

In total, while the original execution time of Job0 is 30, its execution time with oversubscribing is 40. Note that in the instance, since Core 2 and 3 are spped down due to multiplicity, wasted CPU time occurs. The wasted CPU time approximate to 0.50 [speed] * 2 [core] * 10 [s] = 10 [CPU Core time]. Increasing the number of speed down cores may invite a reduction in system throughput. However, if you sticked to avoiding speed down cores completely, it would not achieve the effectiveness sufficiently since jobs can not start immediately. We should admit that the waste always accompanies this scheduling policy and explore optimum node/core selection methods in the future.

3.3 Scheduling Considering Oversubscribing

With our oversubscribing policies, a job may suffer from waiting time due to a lack of enough slots or memory. Thus NCS needs to maintain a job waiting queue like conventional schedulers. We have implemented two scheduling algorithms on NCS based on well-known algorithms, first-come first-served (FCFS) and EASY backfilling that are revised to support oversubscribing. This added implementation mainly focuses on logical consistency to execute oversubscribing scheduling without any problem. Thus at present, we have not undertaken some complicated optimization specific to oversubscribing: considering the preference of oversubscribing for each job, local maximum multiplicity m for each job, minimizing the number of speed down processor cores, and so on.

First, FCFS is easy to support oversubscribing. The main modification is the treatment of requesting time. Execution time may exceed the original requesting time owing to oversubscribing, which leads the unintentional termination. To avoid it, the scheduler scales the requesting time proportionally when oversubscribing. The scheduler takes a job from the head of the waiting queue and tries to allocate resources for it. If there are not enough slots, considering multiplicity, the job is stuck. After the job is successfully scheduled, the scheduler can take the next job.

On the other hand, backfilling algorithms needs to be modified considerably for the following reason. After the scheduler makes future schedules of several jobs, a new coming job, which is typically a short-running job may be scheduled earlier (backfilled) than the existing jobs. Among backfilling algorithms, we adopt EASY backfilling (EB hereafter[10]), where backfilling is allowed only if *the start times of existing jobs are not delayed*. Note that the decision of backfilling requires information on the execution time of jobs, which are changed due to oversubscribing.

We discuss how EB is modified for oversubscribing while keeping the above-mentioned condition. Figure 3 (a) is a simple case, where Job0, Job1, and Job2 have already been scheduled. When Job3 (n2c1) is submitted at "Current Time" in the figure, the scheduler temporarily allocates cores for it, core1 on node0 and core0 on node1 in this case. Then the scheduler estimates the finish times of Job3 if it is started immediately, considering the performance degradation. And it checks whether the start times of existing jobs are changed or not. If the degraded execution of Job3 is shorter than T, we can estimate that the start time of the existing Job2 is not changed. We consider more about Job0. Since the speed of Job0 is already degraded to 50% due to the existence of Job1, starting Job3 does not degrade the speed of Job1 furthermore. From the above consideration, Job3 can be started immediately.

Figure 3 (b) shows a bit complicated case. Here Job0, Job1, Job2, and Job3 have been scheduled. At "Current Time", Job4 (n2c1) is newly submitted, and core1 on node0 and core0 on node1 are allocated temporarily. In this case, we need to consider performance degradation both of Job4 and existing Job1. This is because starting Job4 would increase the multiplicity of Job1. If either Job1 or Job4 finishes later than Job3's original start time, it would cause a change in the schedule. Thus we abandon backfilling.

In the above case, we just checked only Job1. Generally, however, it is even more complicated since invoking a new parallel job may cause multiple victims to be affected, which has been implemented in NCS.

4 EVALUATION OF OVERSUBSCRIBING SCHEDULING WITH WORKLOAD TRACES

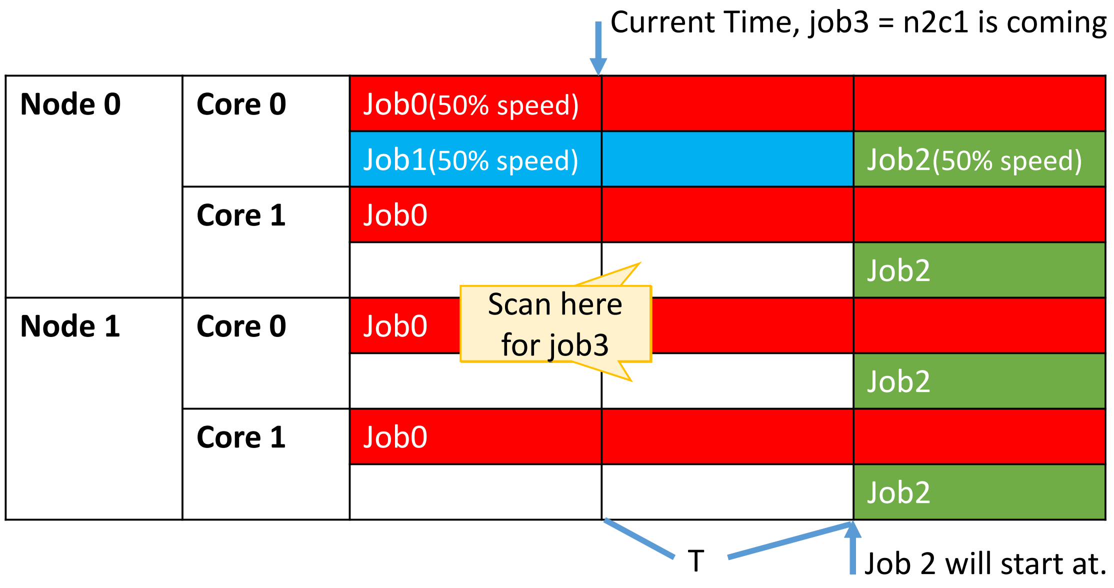
This section evaluates the effects of oversubscribing scheduling using two workload traces on our NCS simulator. The evaluation uses EASY backfilling (EB) scheduling algorithm that is modified for oversubscribing. On the oversubscribing system, the maximum multiplicity M is varied within $1 \leq M \leq 8$. While we execute all jobs in the workload traces, we focus more on behavior of high-responsiveness-requesting jobs (HRR jobs). Basically HRR jobs are smaller in size in terms of parallelism and running time, which are defined in detail in 4.2.

4.1 Evaluation Method

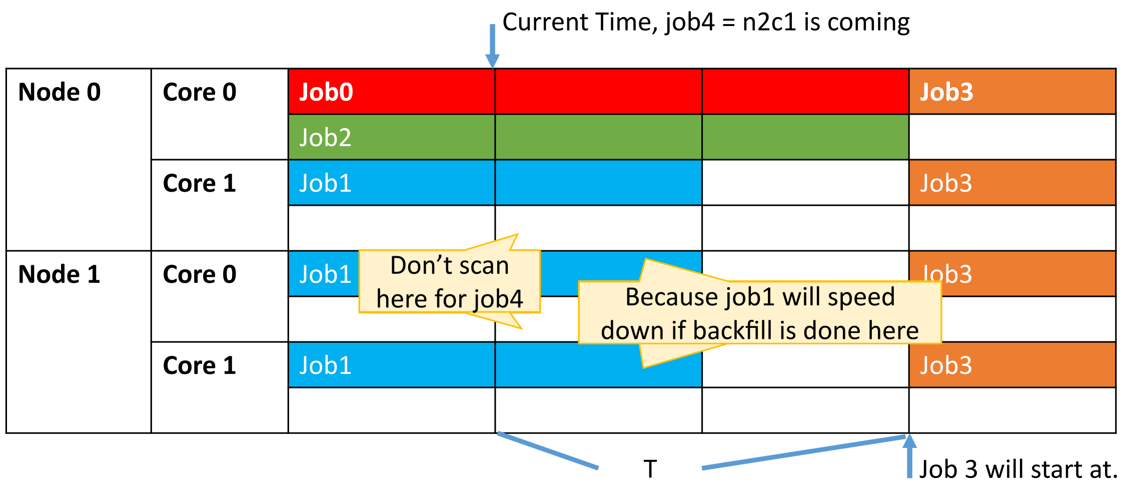
4.1.1 Comparison Target System. We evaluate the advantages of the oversubscribing system by comparing it with a system with another configuration as described below.

While many production supercomputers do not use oversubscribing, some systems prepare several dedicated nodes for HRR jobs separated from the normal nodes for non-HRR jobs (normal jobs) [1, 11]. On such a system (which we call *conventional* system), it has been expected that responsiveness of HRR jobs are kept better.

On the conventional system, we introduce a parameter R, which means the ratio of dedicated nodes for HRR jobs to the entire system, which is configured for each simulation. For example, if the system is N100C10 and R = 5%, the N5C10 system is used for HRR jobs, N95C10 for the rest of the jobs.



(a) Case 1



(b) Case 2

Fig. 3. Behaviors of EASY backfilling with oversubscribing.

On the oversubscribing system, we do not distinguish between HRR jobs and non-HRR jobs. All jobs may be executed on any nodes.

4.1.2 Definition of Evaluation Criteria under Oversubscribing. We reconsider evaluation metrics to support oversubscribing. With oversubscribing, the running time T_r gets larger because other jobs are sharing cores. Thus the value of *slowdown*, a popular metric, may become smaller superficially, which causes underestimation of oversubscribing overhead. Therefore, we consider keeping the denominator constant regardless of the multiplicity. For this purpose, we use T_a, the running time of a job with the multiplicity of one (dedicated execution). We define a new metric, oversubscribing conscious slowdown S_{OSub}. We represent typical and new slowdowns as follows:

$$S = (T_e - T_s)/T_r = (T_r + T_w)/T_r \quad (1)$$

$$S_{\mathrm{OSub}} = (T_e - T_s)/T_a = (T_r + T_w)/T_a \quad (2)$$

Here T_s, T_e, and T_w are each job submission, end, and wait time[6]. We use S_{OSub} to evaluate the responsiveness of jobs.

4.2 Target Workload Traces

In our simulation, we use workload traces in the SWF format that are publicly available, UniLu-Gaia-2014-1 [1] and KIT-FH2-2016-1 [2]. We choose these traces since they are collected on systems with multiple job queues.

4.2.1 UniLu-Gaia-2014-1. This data set (UniLu hereafter) contains three months data from the Gaia cluster at the University of Luxemburg[1]. It contains multiple queues prepared for interactive jobs and batch jobs, respectively. Thus we regard interactive jobs as HRR jobs.

Note that HRR jobs are originally interactive jobs, but they are regarded like (short) batch jobs in the simulation, whose CPU utilization is constant during resource allocation. The speed down of job is determined by m, as described before. The accurate simulation of interactive jobs will be investigated as future work, and we discuss the direction in Section 5.2.

Table 1 shows the system information and the workload information. The ratio of HRR jobs is 3.4%, thus the ratio of HRR nodes R on the conventional system is configured to be around it. In the evaluation, we use $R = 1, 3, 5, 7, 10$ [%].

Table 1. UniLu-Gaia-2014-1

System Configuration	N150C12
# of Nodes	150
# of Cores per Node	12
Workload Characteristic	
# of Jobs	51,871
HRR Jobs	1,762(3.4%)
Maximum Degree of Parallelism	516
HRR Jobs	12
Maximum Execution Time[s]	1,800,012
HRR Jobs[s]	43,507
# of Users	82

4.2.2 KIT-FH2-2016-1. This data set (KIT hereafter) contains one and a half years worth of accounting records from the ForHLR II system located at the Karlsruhe Institute of Technology in Germany[2].

We observed that two queues are used differently depending on the job size; one queue is used for smaller scale jobs. Thus we regard jobs in the queue as HRR jobs.

Table 2 shows the system information and the workload information. We evaluate the conventional system with $R = 0.5, 1, 3, 5$ [%].

Table 2. System and workload information for KIT-FH2-2016-1

System Configuration	N1152C20
# of Nodes	1,152
# of Cores per Node	20
Workload Characteristic	
# of Jobs	114,347
HRR Jobs	2,349(2.1%)
Maximum Degree of Parallelism	22,960
HRR Jobs	48
Maximum Execution Time[s]	604,800
HRR Jobs	259,200
# of Users	161

4.3 Evaluation Results

4.3.1 The Responsiveness of HRR Jobs. First, we evaluate the waiting time of jobs. Figure 4 shows the maximum waiting time among all HRR jobs. If the value is zero, no HRR job suffers from waiting time, which is the best case for the users.

In the conventional system, waiting times for HRR jobs get better as R increases since the system is configured preferably for HRR jobs. When R reaches 10%, all jobs can be processed immediately. Also in the oversubscribing system, the responsiveness improves with more aggressive oversubscribing with larger M. When M reaches four, all jobs, including normal jobs though not displayed in the figure, can commence with zero waiting time.

Figure 5 shows results with jobs in the KIT trace. While we observe a similar tendency on the conventional system, waiting time is not eliminated on the oversubscribing system with $M = 8$. Nevertheless, we expect that around 6 minutes of the longest waiting time is still tolerable for typical users.

From the discussion, oversubscribing scheduling can immediately provide the resources for HRR jobs as well as a well-configured conventional system.

4.3.2 Evaluation of Slowdown. Next, we evaluate the slowdown of jobs mainly for normal jobs. Here we use S_{OSub}, which has been revised in Section 4.1.2. Figure 6 shows S_{OSub} on the conventional and the oversubscribing system with UniLu trace.

Slowdowns of normal jobs are largely different between the two systems. In the conventional system, the values for normal jobs get worse as R increases. On the other hand, the slowdown values of all HRR jobs are one with R = 10 [%], which corresponds to the fact that the waiting time is zero. This indicates that when the conventional system is configured for efficient execution of HRR jobs, it is unfavorable for normal jobs, introducing a trade-off. When R is 10% with UniLu, the slowdown of normal jobs is quite terrible. We also observe a similar tendency with KIT (Figure 7); all HRR jobs can be started without waiting when R is 5% as shown in Figure 5, however, the response of normal jobs is awful.

On the other hand, with oversubscribing scheduling, slowdown gets better both for normal and HRR with larger M. On the other hand, unlike on the conventional system, the slowdown does not reach one. This is due to performance

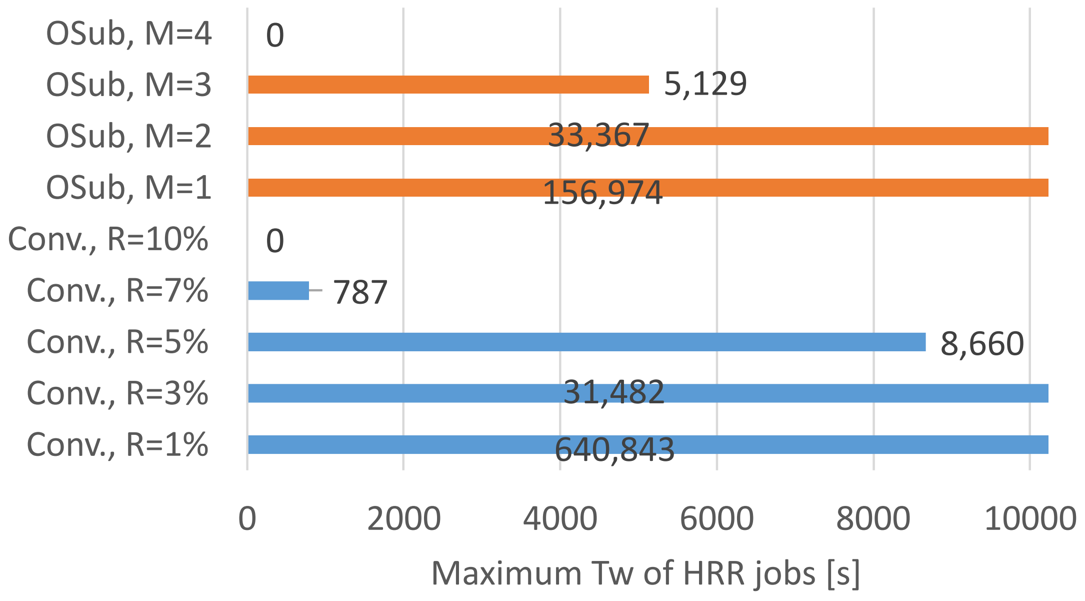


Fig. 4. Maximum waiting time for HRR jobs when conventional system and oversubscribing system with UniLu-Gaia-2014-1 workload trace. Oversubscribing results with $M > 4$ are the same with $M = 4$. Since some cases exceed the upper bound of axis, the value for each case is included near the bar.

degradation caused by oversubscribing. While we suffer from this overhead, we can conclude that normal HRR and normal jobs can coexist efficiently on the oversubscribing system.

4.3.3 Detailed Evaluation of Slowdown. In the previous section, we examined the maximum value of slowdown. In order to evaluate tendency in detail, Figures 8 and 9 show distribution of slowdown values among jobs. The X-axis is the cumulative ratio of jobs and the Y-axis is the slowdown value. Here HRR and normal jobs are mixed.

The cumulative ratio on the oversubscribing system reaches one faster than that of the conventional case, which means that the maximum slowdown can be suppressed by oversubscribing. Now we compare conventional system with $R = 5\%$ and oversubscribing system with $M = 4$ with UniLu trace. While slowdown with oversubscribing is 4 at maximum, on the conventional system, the line is still at 96.2%. Also, 650 jobs (1.3%) have $S_{OSub} \geq 100$, though we cannot see it in the figure. The KIT trace shows similar results. On conventional system with $R = 3\%$, 3.2% jobs suffer from $S_{OSub} \geq 100$.

With the KIT data trace, while "OSub, M=8" reaches one at $S_{OSub} = 6$, other lines are still around 90%, which means around 10% jobs suffer from $S_{OSub} > 6$. This is explained by the that the KIT trace represents a more crowded system than the UniLu trace. Thus oversubscribing scheduling is preferable to improve responsiveness on such a crowded system with a proper configuration of M. Also oversubscribing tends to be fair for all the jobs in the aspect of slowdown.

Those figures also show the middle of oversubscribing curves is higher than conventional ones, which means that some jobs are degraded by oversubscribing. Figures 10 and 11 show the slowdown comparing conventional and oversubscribing systems. They are average slowdown ratios between conventional and oversubscribing in normal and

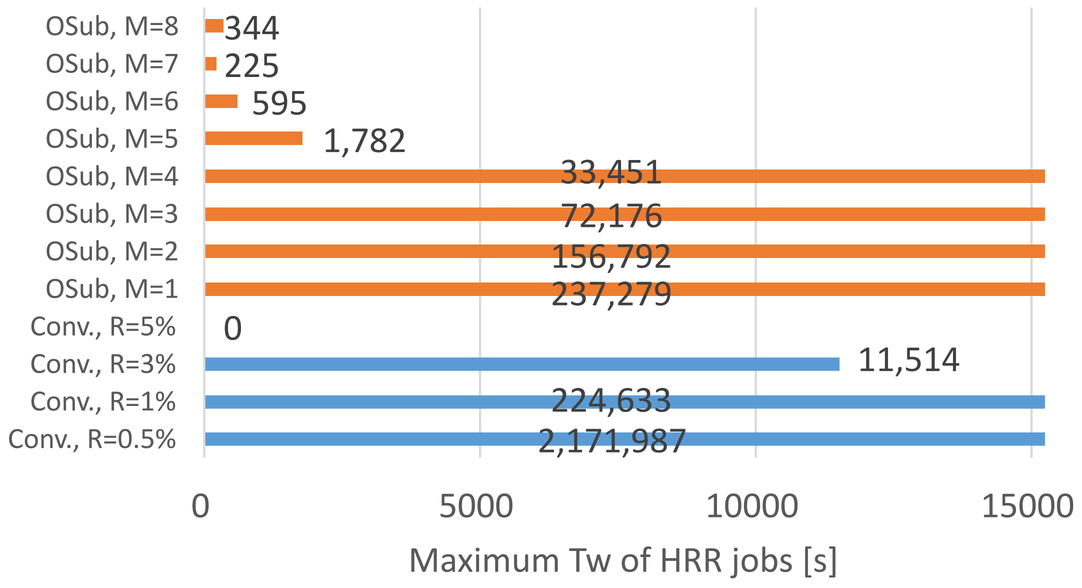


Fig. 5. Maximum waiting time for HRR jobs when conventional system and oversubscribing system with KIT-FH2-2016-1 workload trace. Since some cases exceed the upper bound of axis, the value for each case is included near the bar.

HRR jobs, binned by the size and length of jobs. From both figures, short jobs drastically improved, which corresponds to the elimination of waiting time by oversubscribing as shown in Figures 4 and 5. On the other hand, long jobs tend to be degraded. In this case, the turn around time of long jobs can be approximated to $T_r + Tw \sim Tr$. Thus, the effect of reduction of T_w by oversubscribing will not be expected as short jobs.

From the above discussion, we can conclude the contribution to the slowdown as follows: (1) Oversubscribing can suppress the maximum slowdown. There are no extremely late jobs. (2) Short jobs improve but long jobs do not. It depends on whether the running time is dominant to the turnaround time.

4.3.4 Investigation of Individual Jobs. So far we have evaluated different scheduling methods statistically. Contrarily, this section picks up some individual jobs with specific characteristics from the UniLu data set shown in Table 3.

Job 2819 is a massively parallel job with 200 cores, and the slowdown is largely improved from 11.0 to 2.00 by introducing oversubscribing, which is an example of how oversubscribing works effectively for massively parallel jobs. Job 918 has a short execution time. Although it is a normal job, we assume that the responsiveness of such jobs is important. In this case, there are no jobs with a quite painful slowdown in the conventional system accidentally. However, we expect oversubscribing can reduce the waiting time for this type of job. Job 9121 is a job with a long execution time. We observe its slowdown is improved from 7.07 to 2.00.

4.3.5 Evaluation of the overall system efficiency. Finally, we show the overall system efficiency. Figure 12 shows the makespan with UniLu trace. There is not a significant difference between cases. The increase in the worst case ("OSub, M=2") is very tiny, 2.5%. It is tolerable. Table 4 shows the 90 th and 95 th percentiles of computed jobs. The values are

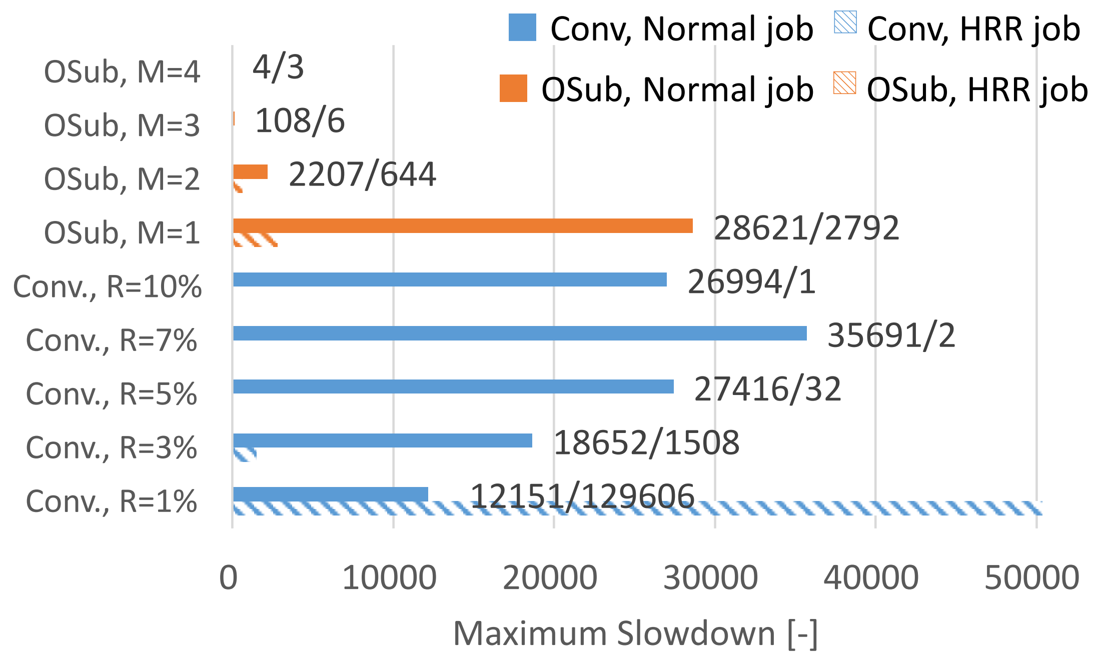


Fig. 6. Maximum Oversubscribing conscious slowdown for normal and HRR jobs when conventional system and oversubscribing system with UniLu-Gaia-2014-1 workload trace. Oversubscribing results with $M > 4$ are the same with $M = 4$. Filled bars represent normal jobs, striped bars HRR jobs. The label shows the value of normal job/HRR job. Since some cases exceed the upper bound of axis, the value for each case is included near the bar.

Table 3. Examples of Improved Jobs Under Oversubscribing. Conventional system is with $R = 5\%$ and Oversubscribing system is with $M = 4$. T_{spec} is the specified time and P is the parallelism. Rest variables are defined in Section 4.1.2

Job ID	T_a	Job Class	T_{spec}	P	T_r		T_w		S_{OSub}	
					Conv.	OSub	Conv.	OSub	Conv.	OSub
2819	2,938	normal	36,000	200	2,938	5,876	29,289	0	11.0	2.00
918	119	normal	300	4	119	119	0	0	1.00	1.00
9121	20,920	normal	36,000	12	20,920	41,840	12,702	0	7.07	2.00

the same in all cases. As mentioned in Section 3.2, although oversubscribing scheduling achieves high responsiveness it may invite a degredation of system efficiency. However, these results suggest that oversubscribing scheduling, if ever, hardly affects in terms of system efficiency.

With the KIT trace, a similar consideration can be derived with from Figure 13 and Table 5. The increase in makespan of the worst case ("OSub, M=8") is 1.2% and negligible. The percentiles are slightly degraded in oversubscribing systems. However, it is tolerable. We confirm the low impact on system efficiency also in the trace.

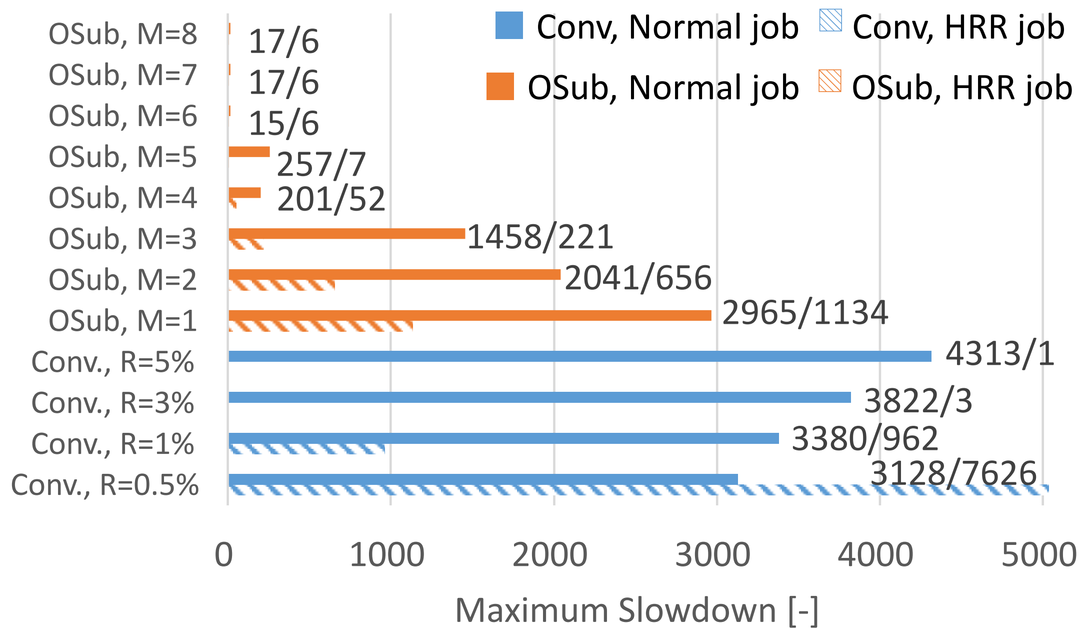


Fig. 7. Maximum Oversubscribing conscious slowdown for normal jobs when conventional system and oversubscribing system with KIT-FH2-2016-1 workload trace. Filled bars represent normal jobs, striped bars HRR jobs. The label shows the value of normal job/HRR job. Since some cases exceed the upper bound of axis, the value for each case is included near the bar.

Table 4. 90th and 95th percentiles of computed jobs when conventional system and oversubscribing system with UniLu-Gaia-2014-1 workload trace. Oversubscribing results with $M > 4$ are the same with $M = 4$. The values indicate the time when 51,871 * 0.90 = 46,684 th and 51,871 * 0.95 = 49,278 th job are finished.

	90th percentile [hour]	95th percentile [hour]
OSub, M=4	2094	2105
OSub, M=3	2094	2105
OSub, M=2	2094	2105
OSub, M=1	2094	2105
Conv, R=10%	2094	2105
Conv, R=7%	2094	2105
Conv, R=5%	2094	2105
Conv, R=3%	2094	2105
Conv, R=1%	2094	2105

4.4 Summary of Evaluation

In this section, we have evaluated oversubscribing scheduling system by comparing it with a conventional system with separated queues, using two actual workload traces. We demonstrated that oversubscribing largely decreases response time, which is critical for HRR jobs. It could be achieved on the conventional system, if it is configured with plentiful dedicated HRR nodes, however, it makes the slowdown of normal jobs worse. We observe that oversubscribing

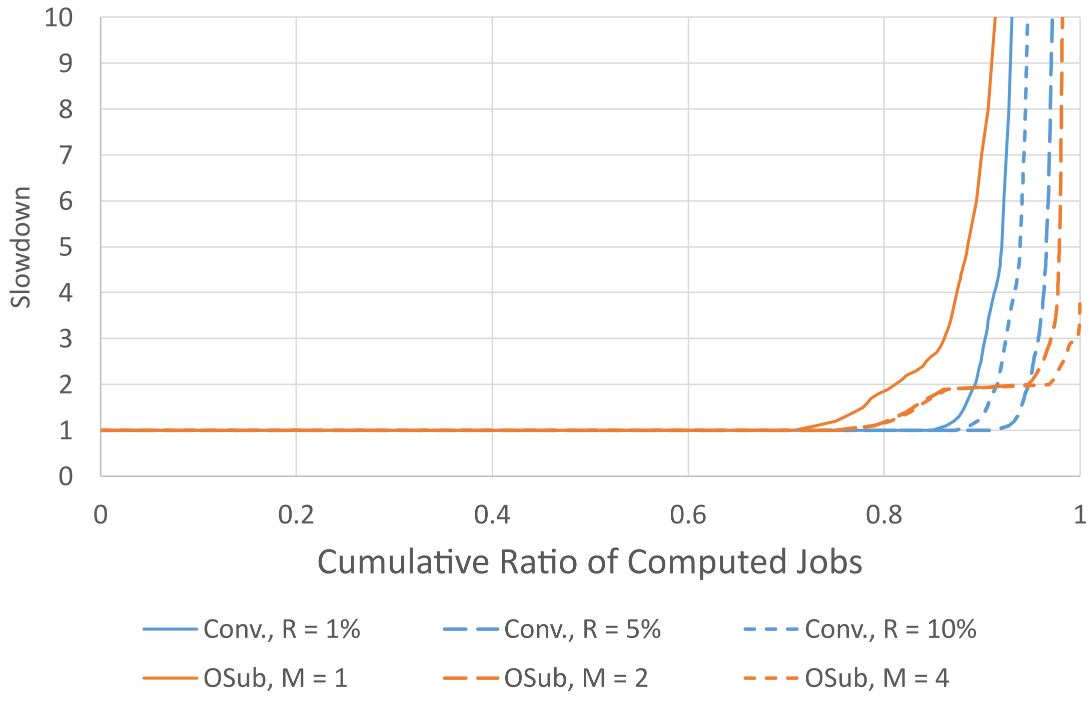


Fig. 8. Distribution chart of Slowdown for all jobs when conventional system with $R = 1\%, 5\%, 10\%$ and oversubscribing system with $M = 1, 2, 4$ with UniLu-Gaia-2014-1 workload trace.

Table 5. 90th and 95th percentiles of computed jobs when conventional system and oversubscribing system with KIT-FH2-2016-1 workload trace. The values indicate the time when 114,347 * 0.90 = 102,913 th and 114,347 * 0.95 = 108,630 th job are finished.

	90th percentile [hour]	95th percentile [hour]
OSub, M=8	13566	13728
OSub, M=7	13568	13728
OSub, M=6	13569	13728
OSub, M=5	13568	13728
OSub, M=4	13568	13728
OSub, M=3	13566	13727
OSub, M=2	13566	13726
OSub, M=1	13566	13726
Conv, R=5%	13570	13726
Conv, R=3%	13568	13726
Conv, R=1%	13566	13726
Conv, R=0.5%	13565	13726

improves turn around time, which is not seen in the conventional system. Thus the oversubscribing scheduling works efficiently both for HRR jobs and normal jobs.

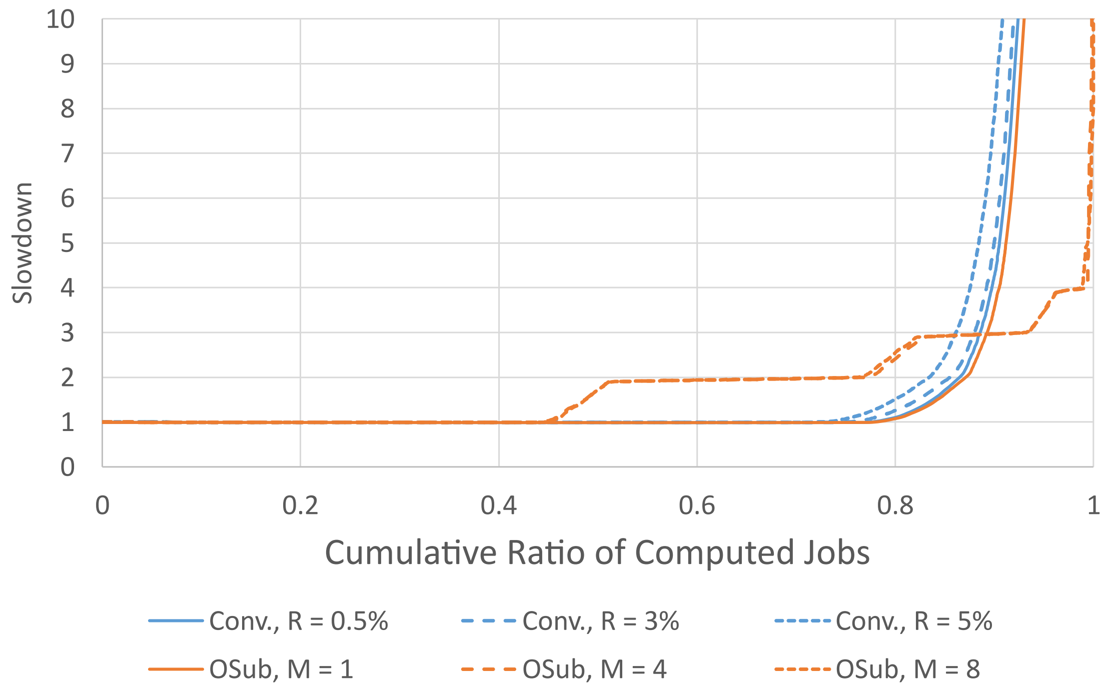


Fig. 9. Distribution chart of Slowdown for all jobs when conventional system with R = 1%, 3%, 5% and oversubscribing system with M = 1, 4, 8 with KIT-FH2-2016-1 workload trace.

Also, oversubscribing scheduling is robust to the changes in the number of HRR jobs. On conventional systems, administrators have to change the number of dedicated HRR nodes to support the fluctuation, however, it has disadvantages in the system CPU utilization. On the contrary, the oversubscribing system can adapt to the changes in the ratio of HRR jobs, both an increase and decrease without any configuration change.

5 DISCUSSIONS

5.1 Comparison to related work

Prior to this work, Hofmeyr et al. have reported the simulation results of time-sharing, oversubscribing scheduling[8]. Here we discuss relationship between their work and this work using Table 6, which summarizes the primary differences.

In order to show the superiority of job scheduling methods, it is important to use realistic and large scale job data sets. Hofmeyr's work uses data set collected on the NERSC Edison supercomputer, which is a large scale set with 2.4M jobs. On the other hand, our simulation uses two data sets (UniLu and KIT), each of which is smaller than theirs. On the other hand, since we use data sets collected on different supercomputers, it has been demonstrated that our method can improve slowdown and latency with job mixes with difference tendencies. Also the data sets we used are publicly available and our results are reproducible by other researchers.

For scheduling algorithm, both methods are based on FIFO + backfilling algorithm.

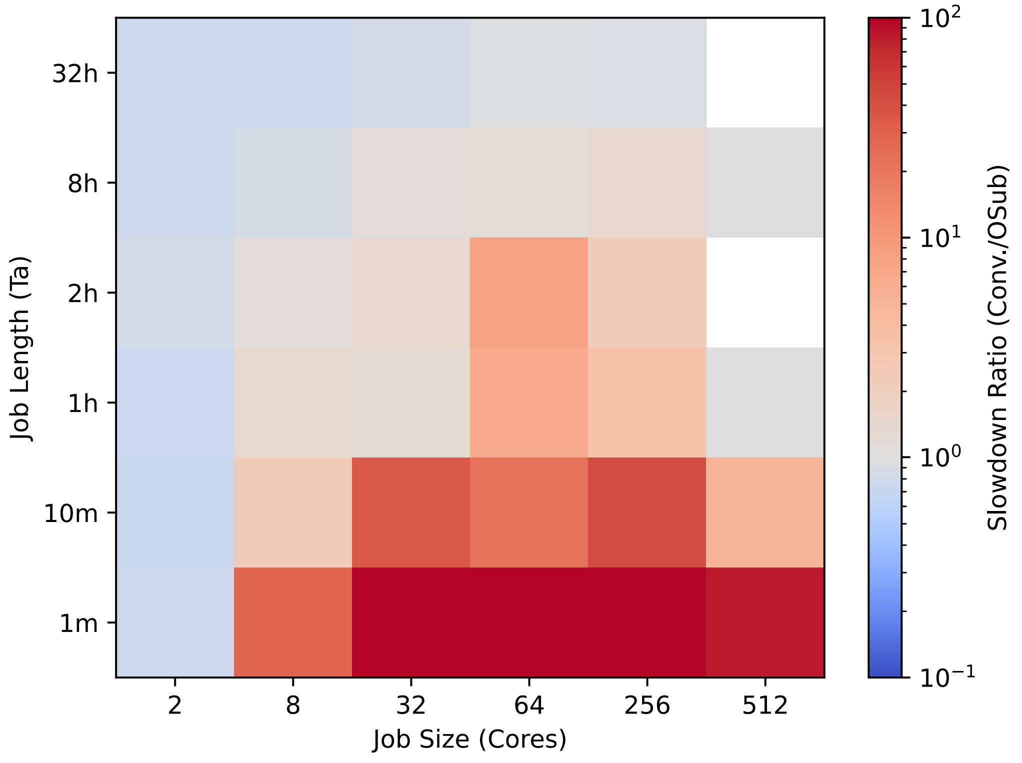


Fig. 10. Average slowdown ratio between conventional when $R = 7\%$ and oversubscribing when $M = 4$, binned by the size and length of jobs with UniLu-Gaia-2014-1 workload trace. Conventional and oversubscribing slowdowns are averaged(arithmetic mean) inside each bin and the ratio is calculated by dividing the values. White bin means no jobs there. The red bin means the oversubscribing is effective while the blue bin does it is ineffective.

The last row of the table shows that we use a more accurate model of performance degradation under time-sharing. Our scheduling algorithm and simulator consider the multiplicity in core-level; the speeds of jobs are affected by "how many threads are sharing each core". On the other hand, their work considers multiplicity in node-level, "how many jobs are sharing each node". Since modern systems consists of multi/many core CPUs, the "node-level multiplicity" may introduce misevaluation of actual performance evaluation, especially when some jobs occupy only subsets of CPU cores per node. Such jobs include interactive jobs discussed in Section 5.2. We will demonstrate advantages of our core-level approach quantitatively in the near future.

5.2 Discussion on Interactive Jobs

This paper has assumed that CPU utilization of each job is constant over time as shown in Figure 14 (a), while the number of allocated CPU cores may be different among jobs. We used the assumption even in evaluation with the UniLu data set (Section 4.2.1), which includes jobs marked as "interactive". The reason for this assumption is that the

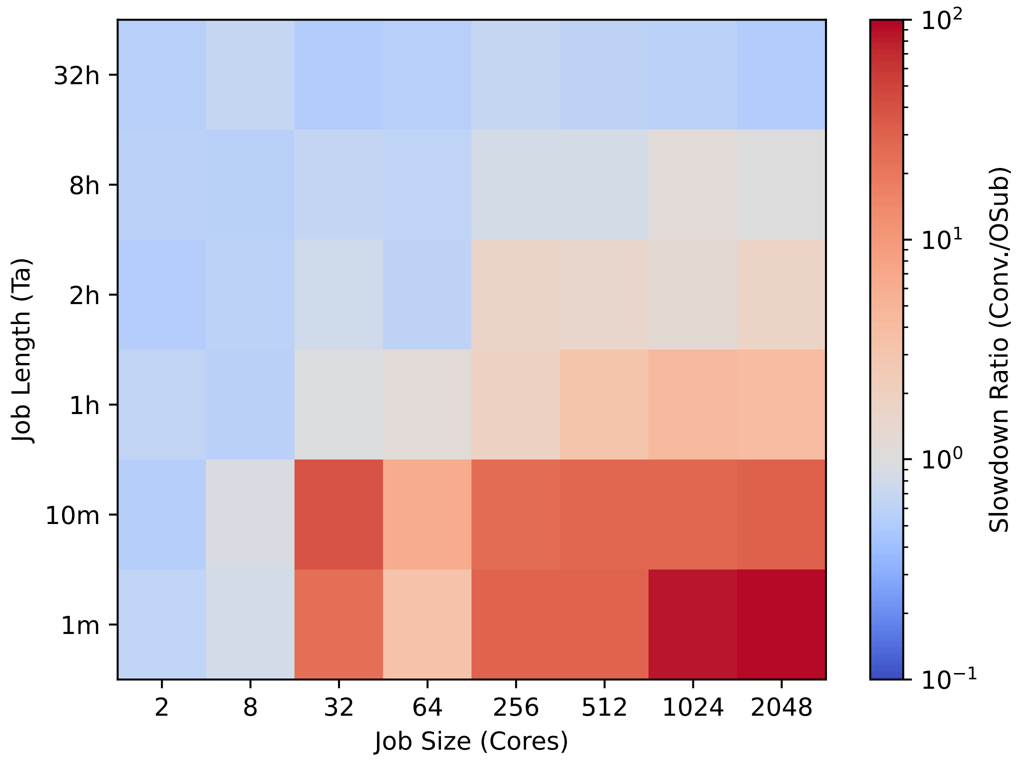


Fig. 11. Average slowdown ratio between conventional when $R = 3\%$ and oversubscribing when $M = 8$, binned by the size and length of jobs with KIT-FH2-2016-1 workload trace. Conventional and oversubscribing slowdowns are averaged(arithmetic mean) inside each bin and the ratio is calculated by dividing the values. The red bin means the oversubscribing is effective while the blue bin does it is ineffective.

Table 6. The primary differences in data and method between ours and related work[8]. The fourth column shows which is superior for each viewpoint. Its flag means more accurate or realistic time-sharing (oversubscribing) simulation in each viewpoint.

	Ours	**Hofmeyr's work**	**Superiority**
Size of data sets	50K jobs, over 3 months (UniLu) 110K jobs, over 19 months (KIT)	2.4M jobs, over 24 months	Hofmeyr's
Number of data sets	2	1	Ours
Scheduling algorithm	FIFO + Backfilling	FIFO + Backfilling	Fair
Degradation model under time-sharing	Multiplicity of cores	Multiplicity of nodes	Ours

SWF format does not provide information on the transition of core utilization during job execution. On the other hand, the actual interactive usage of supercomputers or cloud resources can introduce heavy fluctuation of CPU and resource utilization. As illustrated in Figure 14 (b), typical interactive users tend to repeat the following cycles: they execute some computation processes, and after seeing the results, they consider and determine what is executed next.

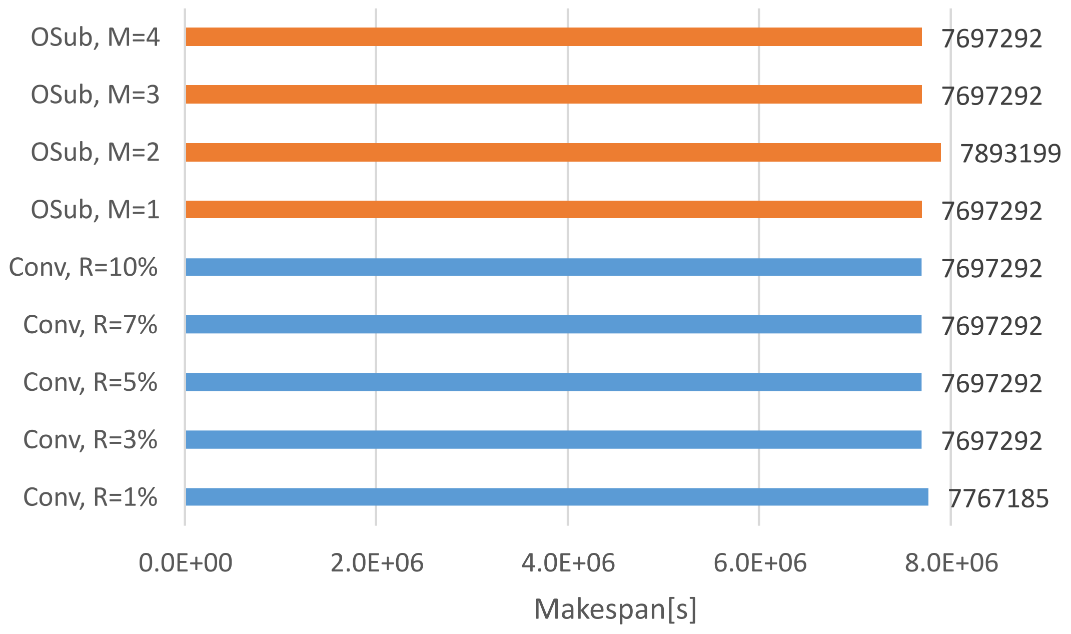


Fig. 12. Makespan, the finish time of final job, when conventional system and oversubscribing system with UniLu-Gaia-2014-1 workload trace. Oversubscribing results with $M > 4$ are the same with $M = 4$. The values are included on the right side.

From this discussion, our simulation with the above assumption tends to overestimate the costs introduced by oversubscribing, if there are jobs with fluctuating CPU utilization like interactive jobs. This overestimation would be improved if we could consider phase changes during the execution of interactive jobs. For this purpose, we plan to extend NCS based on a model shown in Figure 14 (c). Here NCS takes new modes of each job, active mode and idle mode. When a job is idle ("thinking" phase in Figure 14 (b)), it does not have an impact on the performance of other jobs sharing the same CPU cores. When a job is active, it works similarly in the current NCS; active jobs that are sharing cores get slowed down based on the multiplicity. The above discussion reinforces the importance of considering core-level oversubscribing.

While we expect this extension to NCS itself is simple and straightforward, there will be challenges in collecting job data including mode changes. As far as we have searched, there is no such data publicly available. Instead, we plan to model the behaviors of interactive users statistically to produce phase information and combine it with SWF data. For this purpose, we will obtain basic statistical data, including the average active rate, average and standard deviation of length of each phase, and so on. They can be obtained through the monitoring of interactive users on real systems.

As another topic, we discuss the improvement of scheduling policy. In the current oversubscribing scheduling, there is no difference in the treatment of batch (normal) and interactive (HRR) jobs; the classification is used only in the evaluation. One of the future directions is to modify the scheduling policy using job classification according to the preference of users and/or administrators. If they want to reduce the idle time of the CPU, the scheduler

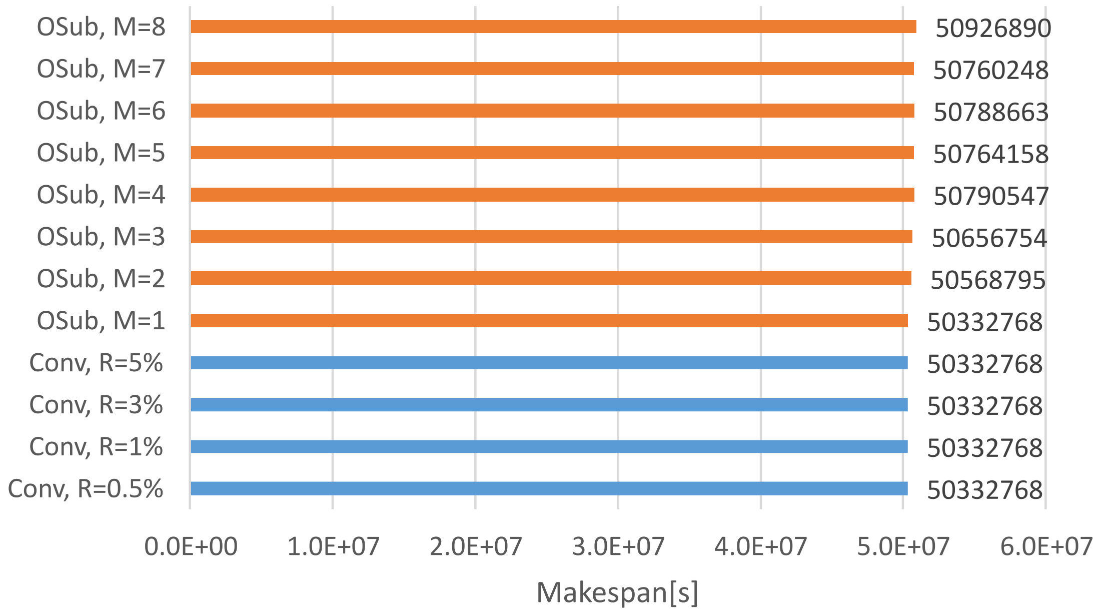


Fig. 13. Makespan, the finish time of final job, when conventional system and oversubscribing system with KIT-FH2-2016-1 workload trace. The values are included on the right side.

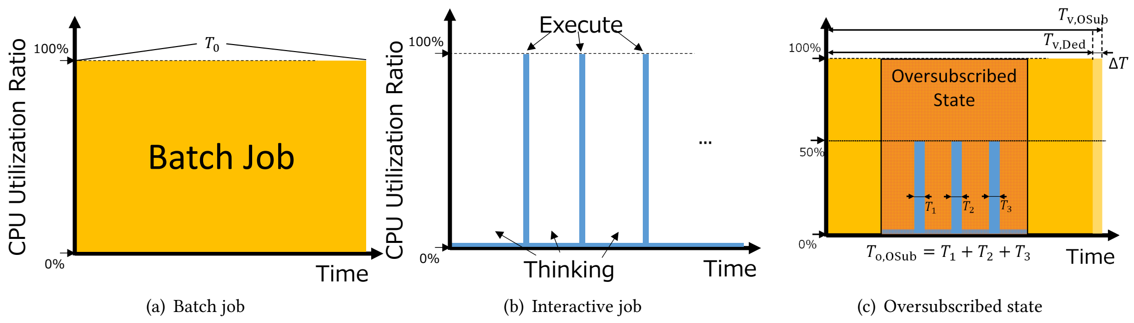


Fig. 14. The time-series of CPU utilization. The horizontal axis is time, the vertical is CPU utilization ratio.

should do oversubscribing aggressively. If they want to avoid performance degradation, the scheduler should suppress oversubscribing so that oversubscribing occurs only among interactive jobs with lower core utilization.

6 RELATED WORK

Hofmeyr et al. have reported the simulation of batch and time-sharing scheduling with their supercomputer's data[8]. We have mentioned the relationship between their work and ours in Section 5.1. Their research has covered various topics related to time-sharing on supercomputer systems, however, interactive jobs are not mentioned. We will focus on this point in the future.

Albert et al. have studied interactive jobs from a job scheduling viewpoint[13, 14]. They claim that high responsiveness promotes users' productivity, and in fact, report their on-demanded interactive system can change user experiences drastically.

Klusáček et al. have built a scheduling simulator for supercomputers and cluster systems, Alea[9]. It takes an SWF file as input and implements various scheduling algorithms. Unfortunately, Alea is not aware of node boundaries and allocates jobs to a huge node with NC cores, where N and C are the numbers of nodes and cores. Also, it is not implemented to manage core numbers bound to jobs. In our context, we want to consider oversubscribing per core, thus we need a simulator that can bind parallel jobs to desired nodes and cores. Since we could not find an existing simulator that satisfies this condition, we have developed NCS.

Slurm[16], the OSS job scheduler, is equipped with the function of oversubscribing. Administrators can configure the compute resources should be shared with multiple jobs. When we develop a real oversubscribing scheduling system in the future, we expect Slurm can be used for the implementation basis. Even with the basis, it is not trivial to develop scheduling algorithms, especially EASY Backfilling considering oversubscribing. Also, the scheduling algorithm requires an estimation mechanism of performance degradation as shown in Section 3.2.

7 CONCLUSION AND FUTURE WORK

We developed a scheduler simulator, NCS, that takes oversubscribing into account, and evaluated oversubscribing scheduling using actual supercomputer workload traces. NCS is equipped with scheduling algorithms, one of which is EASY backfilling algorithm adapted for oversubscribing.

Through the simulated evaluation, we confirmed the oversubscribing system is superior to the conventional system for the following factors. Oversubscribing reduces the waiting time of high-responsiveness-requesting jobs (HRR jobs) largely. While it can be achieved on the conventional system, the configuration designated for HRR jobs hurts the responsiveness of normal jobs heavily. Oversubscribing system is free from this trade-off and improves responsiveness both for HRR and normal jobs.

In this paper, we conducted the simulations with pessimistic assumptions, which overestimate the impact of performance degradation by oversubscribing. We have observed the advantages described above even with those assumptions, thus we expect the real oversubscribing system would produce more benefits both for interactive users and batch users.

The future works are as follows:

- **Accurate simulation of interactive jobs:** As discussed in Section 5.2, we do not take into account the fluctuation of CPU utilization of interactive jobs. While the modification is expected to emphasize the advantages of oversubscribing, we need a realistic model of behaviors of interactive jobs.
- **Consideration of accelerators:** Today's supercomputers have heterogeneous compute resources, such as GPU, FPGA, and so on. We assume that the next consideration of compute resources in the simulation should be GPU. Since some interactive jobs are used for the pre/post processing via GUI, oversubscription of GPU is promising. GPU load should be considered for realistic estimation of responsiveness of HRR(interactive) jobs.
- **Development of Oversubscribing Scheduler System:** While our current evaluation uses simulation, we plan to develop system software to realize oversubscribing on an actual system. The current plan is to improve

the OSS such as Slurm[16] or Open PBS[4]. However, we need development as described in Section 6 to harness the benefits of oversubscribing scheduling.

REFERENCES

[1] 2014. The University of Luxemburg Gaia Cluster log. Retrieved Octorber 4, 2022 from https://www.cs.huji.ac.il/labs/parallel/workload/l_unilu_gaia/index.html

[2] 2018. The KIT ForHLR II log. Retrieved Octorber 4, 2022 from https://www.cs.huji.ac.il/labs/parallel/workload/l_kit_fh2/index.html

[3] 2022. Node Conscious Shceduler simulator. Retrieved Octorber 4, 2022 from https://bitbucket.org/for-double-blinded-review/nodeconsciousscheduler

[4] Altair Engineering. 2022. OpenPBS Open Source Project. Retrieved Octorber 4, 2022 from https://www.openpbs.org/

[5] Steve J. Chapin, Walfredo Cirne, Dror G. Feitelson, James Patton Jones, Scott T. Leutenegger, Uwe Schwiegelshohn, Warren Smith, and David Talby. 1999. Benchmarks and Standards for the Evaluation of Parallel Job Schedulers. In *IPPS/SPDP '99/JSSPP '99*. 67–90.

[6] Dror G Feitelson, Larry Rudolph, Uwe Schwiegelshohn, Kenneth C Sevcik, and Parkson Wong. 1997. Theory and practice in parallel job scheduling. In *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 1–34.

[7] Dror G. Feitelson, Dan Tsafrir, and David Krakov. 2014. Experience with using the Parallel Workloads Archive. *J. Parallel and Distrib. Comput.* 74, 10 (2014), 2967–2982. https://doi.org/10.1016/j.jpdc.2014.06.013

[8] Steven Hofmeyr, Costin Iancu, Juan Colmenares, Eric Roman, and Brian Austin. 2016. Time-Sharing Redux for Large-Scale HPC Systems. In *IEEE HPCC/SmartCity/DSS 2016*. 301–308. https://doi.org/10.1109/HPCC-SmartCity-DSS.2016.0051

[9] Dalibor Klusáček, Mehmet Soysal, and Frédéric Suter. 2019. Alea -Complex Job Scheduling Simulator. In *13th International Conference on Parallel Processing and Applied Mathematics*. Bialystok, Poland. https://hal.archives-ouvertes.fr/hal-02329635

[10] David A. Lifka. 1995. The ANL/IBM SP scheduling system. In *Job Scheduling Strategies for Parallel Processing*, Dror G. Feitelson and Larry Rudolph (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 295–303.

[11] Satoshi Matsuoka, Toshio Endo, Akira Nukada, Shinichi Miura, Akihiro Nomura, Hitoshi Sato, Hideyuki Jitsumoto, and Aleksandr Drozd. 2017. Overview of TSUBAME3.0, Green Cloud Supercomputer for Convergence of HPC, AI and Big-Data. *TSUBAME e-Science Journal* 16 (2017), 2–9.

[12] Shohei Minami, Toshio Endo, and Akihiro Nomura. 2021. Measurement and Modeling of Performance of HPC Applications Towards Overcommitting Scheduling Systems. In *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 59–79. https://doi.org/10.1007/978-3-030-88224-2_4

[13] A.I. Reuther, T. Currie, J. Kepner, H.G. Kim, A. McCabe, P. Michaleas, and N. Travinin. 2005. Technology Requirements for Supporting On-Demand Interactive Grid Computing. In *2005 Users Group Conference (DOD-UGC'05)*. 320–327. https://doi.org/10.1109/DODUGC.2005.65

[14] Albert Reuther, Jeremy Kepner, Chansup Byun, Siddharth Samsi, William Arcand, David Bestor, et al. 2018. Interactive Supercomputing on 40,000 Cores for Machine Learning and Data Analysis. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*. 1–6. https://doi.org/10.1109/HPEC.2018.8547629

[15] John Shalf, George Michelogiannakis, Brian Austin, Taylor Groves, Manya Ghobadi, Larry Dennison, Tom Gray, Yiwen Shen, Min Yee Teh, Madeleine Glick, and Keren Bergman. 2020. Photonic Memory Disaggregation in Datacenters. In *Photonics in Switching and Computing 2020*. PsW1F.5.

[16] Andy B Yoo, Morris A. Jette, and Grondona Mark. 2003. Slurm: Simple Linux Utility for Resource Management. In *Workshop on job scheduling strategies for parallel processing*. Springer, 44–60.