The Aggressive Oversubscribing Scheduling for Interactive Jobs on a Supercomputing System

Shohei Minami
Prometech Softwere Inc. and
Tokyo Institute of Technology
Email: minami@prometech.co.jp

Toshio Endo
Tokyo Institute of Technology
Email: endo@is.titech.ac.jp

Akihiro Nomura
Tokyo Institute of Technology
Email: nomura@gsic.titech.ac.jp

***Abstract*—As interactive usages of supercomputing systems become popular, especially in the AI and machine learning (ML) fields, the systems are expected to provide resources in real time. As interactive jobs have different features from traditional batch jobs, the systems should be designed to accept both types of jobs efficiently. This paper shows that the aggressive oversubscribing scheduling, in which multiple jobs share computational resources regardless of job types, can effectively process hybrid jobs. This paper investigates behaviors of the real interactive jobs with fluctuating CPU utilization. And a simulation method is described, which combines existing workload trace data and data on CPU utilization. Through the evaluation, we demonstrate oversubscribing scheduling achieves a short response time for interactive jobs. Also our solution eliminates the necessity of configuring dedicated queues for job types and achieves robustness towards the change of demand of interactive jobs.**

***Index Terms*—Job scheduling, Simulator, Oversubscribing, Interactive Jobs, Supercomputing systems**

I. Introduction

Due to the widespread use of the AI and machine learning on supercomputing systems, providing computing resources to users in interactive fashions is more important. Thus modern supercomputing systems are expected to accommodate both traditional batch jobs, which tend to have longer running times, and interactive jobs, which may have shorter running times but interact with users in real time. However, satisfying these diverse demands is difficult on systems with traditional scheduling methods since job requests are queued and blocked until resources become available, even for interactive jobs.

To support hybrid types of jobs, some production systems assign several dedicated nodes for interactive jobs [1], [2]. Those nodes compose an interactive queue, separated from the normal batch queue. However, it is not trivial for administrators to configure the proper size of the interactive queue, since amounts of user demand change over time. Systems with a single queue that accept both types of jobs can avoid such a configuration, however, interactive users suffer from longer waiting time for other existing jobs.

From the above discussion, we focus on a method that we call *aggressive oversubscribing (AO) scheduling*, in which multiple jobs coexist on the same node and CPU cores. All jobs are submitted to a single queue and each of them is started on the assigned nodes and cores immediately with conditions described in Section III. Then the new job shares nodes and cores with the coexisting jobs. This approach is expected to reduce waiting time and improve responsiveness compared with the above-mentioned methods. This oversubscribing or time-sharing of CPU cores can be achieved with mechanisms of the conventional OS.

This paper demonstrates the feasibility of the AO scheduling through simulation. To make the simulation to be realistic, we conduct an analysis of the behaviors of interactive jobs by collecting data on a production supercomputer (Section II). Based on the analysis, we conduct the simulation to show the following results. First, responsiveness and waiting time are largely improved especially for interactive jobs. Also, slowdowns of jobs are mitigated not only for interactive jobs but batch jobs. Then we show that AO with the single queue configuration is robust to the change of user demands.

II. Understanding Interactive Jobs

A. Characteristics of Interactive Jobs

Interactive jobs include visualization, debugging of software, development loop of the AI/ML applications, and so on. Some usages can be via web browsers [3]–[6]. We view their characteristics and requirements are different from those of batch jobs as follows.

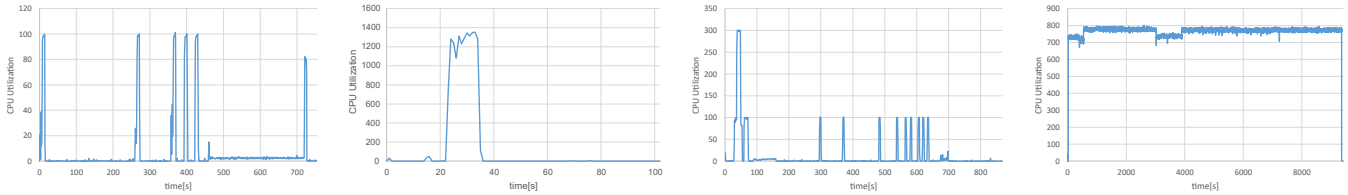
(1) Interactive jobs should be started immediately when users request them. Also the frequency of requests can be more frequent on weekdays and less on weekends, which shows the burstiness.

(2) During the execution, the actual utilization of resources, including CPU cores, tends to fluctuate. This is because users often interact with resources by executing some processes, seeing their results, and determining what to do next.

(3) Traditionally, interactive jobs like visualization tend to occupy only small resources. This tendency is, however, changing as interactive AI/ML usage becomes popular, where processes invoked interactively may require multiple cores and nodes. Thus fluctuation of actual resource utilization described in (2) becomes even larger.

B. Observation of Interactive Jobs

In order to confirm the characteristics described above, we analyze the behavior of actual interactive jobs. For this purpose, we collected time series data of CPU utilization during interactive jobs on the TSUBAME3.0 supercomputer



(a) Single-core usage (b) Multi-core usage (c) Single- and multi-core usage (d) Batch-like usage

Fig. 1. Instances of typical interactive job usages on TSUBAME3.0.

from 9th September 2022 to 1st November 2022. In this system, each interactive job is allowed to use up to 14 logical cores on a node, thus values of CPU utilization are from 0% to 1,400%. The sampling frequency of CPU utilization is five seconds. Excluding data with measurement failure, data of 193 jobs are collected.

Out of the collected data of 193 jobs, we pick up four jobs with different behaviors, and show their time series of CPU utilization in Figure 1. In Figures 1 (a), (b), and (c), we observe the characteristics (2) in the previous section; the CPU utilization fluctuates largely by repeating idle periods and busy periods. We consider that idle periods correspond to "users' thinking time". On the other hand, the utilization ratio in busy periods are different among jobs. While job (a) uses about 100% CPU (a single core), job (b) uses around 1400% (14 cores). The job (c) includes both single-core usages and multi-core usages. These observations support the characteristics (3); even interactive jobs require multiple cores for higher performance. Figure 1 (d) shows a different behavior from others; the CPU utilization is constantly kept high, although this job is invoked on an interactive node. We call such a case a batch-like usage.

In summary, while the actual interactive jobs basically obey characteristics described in the previous section, the utilization in busy periods or lengths of periods is largely different among jobs. Also, some interactive jobs behave like batch jobs.

C. Modeling Interactive jobs

In Figure 1, we have seen the divergent behaviors of interactive jobs. In order to understand jobs more generally, this section introduces a model, whose purposes are two-fold. One is to observe the statistics of collected data. The other is to process the collected time series data on TSUBAME3.0 based on this model and then embed them in the input of the simulation as described in Section IV.

In our model, an interactive job includes repeated busy periods and idle periods during its execution time as illustrated in Figure 2. During idle periods, CPU utilization ratio is almost zero and we consider they correspond to users' thinking time. During busy periods, CPU utilization ratio is higher for some computational processes.

Since CPU cores may be slightly used to react to users' commands or for OS jitters, we use a threshold to distinguish the two periods, which is 20% in our experiment. If the CPU utilization during each sampling period of five seconds is higher than this value, the period is regarded as a part of a busy period.

This model can express batch-like usage like in Figure 1 (d), which is modeled as a job with a single long busy period. On the other hand, it does not consider the difference in CPU utilization among busy periods in a job. Thus it does not distinguish multi-core usage and single-core usage in Figure 1 (c). In the future, the model could be more sophisticated to capture CPU utilization more in detail. This paper, however, uses this simple model for our purposes.

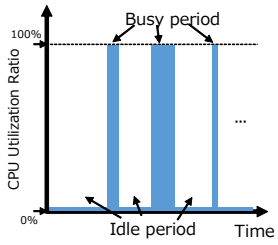


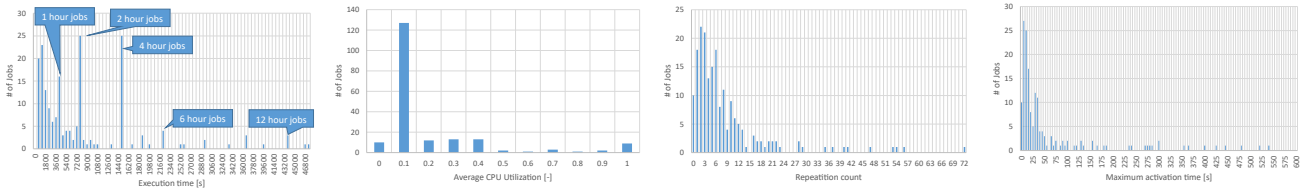
Fig. 2. A simplified behavior of resource utilization during an interactive job. Busy periods and idle periods are repeated.

D. Statistics of Measured Jobs

Based on our model, we have processed all collected data of 193 jobs and taken statistics as shown in Figure 3. In each histogram, the bar heights represent the number of jobs.

1) Job execution time: In Figure 3 (a), we see that short jobs less than 2 hours are popular. Also, there are peaks at 1, 2, 4, 6, and 12 hours and so on. We consider that this is because of users' requested time. While some users close the interactive sessions explicitly, others may leave them without closing, and the system forcibly terminates them. In total, the 90th percentile value is 5.2 hours.

2) Average CPU utilization: Figure 3 (b) corresponds to the average CPU utilization that counts both busy and idle periods in each job. In most jobs, it is quite small; for 71% of jobs, it is less than 10% (0.1 core). Also for 37% of jobs, it is less than 0.01 (it is too small and does not appear in the histogram). These observations reinforce our assertion that dedicating computing resources for interactive jobs degrade efficiency of system usage and the aggressive oversubscribing scheduling is important. On the other hand, we see several jobs with higher CPU utilization, which are considered batch-like jobs.



(a) Job execution time (b) Average CPU utilization (c) Repetition count ($\leq$ 99th percentile) (d) Longest busy period ($\leq$ 85th percentile)

Fig. 3. The primary statistics of interactive jobs collected on TSUBAME3.0. The sample size is 193 (jobs).

3) Repetition count and longest busy period: Figure 3 (c) shows the statistics of repetition counts of busy periods. Jobs with less than 10 busy periods are common, while the maximum count is 94. Figure 3 (d) summarizes the longest busy periods. While the 54th percentile is 30 seconds, the longest busy period in all the data is exceptionally long as 46,715 seconds.

From these observations, we can consider typical behaviors of interactive jobs, each of which repeats busy periods about 10 times and each length is less than 30 seconds. Also, there are jobs with batch-like usage. Our simulation method is designed so that we can reflect these characteristics.

III. Aggressive Oversubscribing Scheduling

This section discusses scheduling methods that accept both interactive jobs and batch jobs, and describes aggressive oversubscribing (AO) scheduling, whose basic idea has been described in authors' previous paper [7] [1].

Supercomputers that support interactive jobs should consider their characteristics described in SectionII-A; the interactive users require computing resources immediately, and the actual CPU utilization ratio during such jobs fluctuate largely. Preparing a single job queue with a traditional batch scheduling cannot satisfy users' demand since it introduces longer waiting time. Also, it degrades CPU utilization of the entire system.

To support two job types, some systems provide a dedicated job queue for interactive jobs in addition to a job queue for batch jobs [1], [2]. We call it multiple queue (MQ) method. With the separated queue, interactive jobs do not wait for the end of the running batch jobs, while they can wait for other interactive jobs. As a variant, systems may enable oversubscribing only for the interactive queue; multiple interactive jobs can share the same cores, which is adopted in TSUBAME3.0.

With the above method, the system administrators have to configure the number of interactive nodes statically. Hereafter R means the ratio of interactive nodes in the system [2]. While this approach is expected to reduce waiting times of interactive jobs, it is still conservative since it is hard to adapt to sudden changes in the amount of interactive jobs. Administrators can change R during the operation, however, it is hard to modify frequent changes, like changes between daytime and nighttime.

From the above discussion, we focus on AO method with a single queue (SQ), which accepts both interactive and batch jobs. To reduce the waiting time drastically, we adopt oversubscribing on all the nodes. Thus nodes and cores can be shared by jobs, regardless of job types. The job processes on the same core are executed on top of preemption mechanism of the OS, in a fine-grained fashion.

While the basic idea of this method is very simple, there are several issues caused by oversubscribing and thus we have proposed several strategies to mitigate them [7].

First, since jobs share CPU cores regardless of job types, even batch jobs may suffer from performance degradation by coexisting jobs. Especially, the performance of a parallel job can be largely degraded by a small single-core job, due to the effects of the synchronization among parallel processes or threads. To mitigate the degradation, the scheduler considers load balancing among nodes/cores as follows. The scheduler dynamically maintains *multiplicity* for each core, which is the number of processes assigned to that core. Here both busy processes and idle processes are counted. When the scheduler accepts a new job submission, it determines nodes/cores for that job so that the multiplicities of cores are kept fair.

Secondly, since the physical resource amount of a node is limited, we cannot invoke infinite processes on a node. Instead, the AO scheduler maintains memory amount allocated by the running jobs on each node. If the remaining memory amount is insufficient, new jobs cannot be executed on that node. Also, we introduce another system-level parameter, named maximum multiplicity M. Jobs are scheduled so that the multiplicity of any core does not exceed M, in order to mitigate overhead by OS preemption. From the above two mechanisms, a newly submitted job may experience waiting time even with AO method.

There are other ways to reduce waiting times. For example, gang scheduling supported by the Slurm scheduler [8] enables oversubscribing of nodes and cores, but it is based on coarse-grained timeslices unlike in AO. The timeslice length is configured by administrators, whose default value is 30 seconds. While it is expected to keep the performance of parallel jobs higher, in our context, the resource usage of the

[1]The previous paper has evaluated oversubscribing only with batch jobs, not considering interactive jobs with fluctuating CPU utilization

[2]For instance, TSUBAME3.0 supercomputer prepares four interactive nodes out of 540 nodes, thus $R = 0.74\%$ [1]

system tends to be degraded since timeslices are given even when processes are idle, which is common for interactive jobs. In AO method, when some processes become idle, OS can immediately activate other processes.

In Section V, we evaluate the AO method with the SQ configuration by simulation, focusing on the waiting times and performance overhead on jobs of both types. Simulation of job scheduling algorithms is common in the literature. However, it is not trivial when we consider interactive jobs whose CPU utilization fluctuates. Thus we discuss the simulation method in Section IV.

IV. Simulation Method with Interactive Jobs

The paper aims to demonstrate the effectiveness of aggressive oversubscribing (AO) scheduling considering both interactive jobs and batch jobs. In order to carry out a simulation of scheduling methods with oversubscribing, we have developed a simulator of job scheduling, named *node conscious oversubscribing scheduler simulator* (NCS) [9].

As the input of simulation, we use the standard work format (SWF), a well-known job trace format [10]. An SWF file consists of records of job information, each of which includes the submitted time point, the requested time, the (actual) execution time, the number of request cores, the request memory amount, and so on.

NCS takes an SWF file as input, and simulates the behavior of each job considering the overhead of oversubscribing. The original NCS, however, assumed the CPU cores are always kept busy during job execution; in other words, it considered only batch jobs [7]. In order to simulate interactive jobs, there are two issues to be solved.

- The SWF file does not include information on the changes of actual CPU utilization during each job.
- The overhead of oversubscribing in the simulation needs to be revised to reflect the fluctuation of CPU utilization.

A. Data Processing To Embed CPU Utilization

In our simulation, we use a workload trace named UniLu-Gaia-2014-1 [11] (UniLu hereafter) from Parallel Workloads Archive [12]. This data set contains three months data from the Gaia cluster at the University of Luxemburg. We choose it since the cluster consists of two queues, one of which is interactive queue (MQ configuration) and each job data has a flag of job type. As shown in Table I, 1,762 jobs (3.4%) are interactive jobs out of 51,871 jobs.

Unfortunately, the SWF format file has no information on CPU utilization. Instead, we embed information collected on TSUBAME3.0 (Section II-B) into data of interactive jobs in UniLu as follows. For each of UniLu interactive job (J_U), we see the execution time of it. And we pick one record from data set generated from TSUBAME interactive jobs (J_T), whose execution time is close to that of J_U. Then we embed data patterns of J_T that consist of busy and idle periods into J_U. To avoid using the same pattern too frequently, we also use randomness in choosing J_T. With this method, the generated behaviors include various types of interactive jobs as shown in Figure 3. For batch jobs, we assume CPU cores are always fully utilized.

TABLE I
The information of UniLu-Gaia-2014-1 trace

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

B. Simulated Overhead of Oversubscribing

In NCS simulator, we use the following assumptions on overhead of oversubscribing. (1) We consider slow down of each process depending on the number of coexisting processes on the same CPU core. (2) In a parallel job, the slowest process or threads in the job determines the progress of all the processes or threads in the job. More details are in [7].

In order to support idle jobs, we implemented the behavior of NCS simulation as follows. When a process is idle, it does not affect the progress of other coexisting processes. Also its progress is not affected by other coexisting processes. The latter rule is set since the lengths of idle periods are mainly determined by interactive users' behaviors, not by other jobs.

Here we mention the slow down of coexisting processes in (1) further. We use an assumption that the speed of each busy process is divided by $(m' \times C_o)$, where m' is the number of coexisting busy processes and C_o is a constant $= 1.2$. C_o is introduced considering overhead of OS preemption. However, we have observed the actual behavior is more complex due to cache pollution, synchronization methods, and so on [13]. Our future work includes evaluation with a variation of overhead.

V. Evaluation Results

This section shows results of the simulation that evaluate the effectiveness of the AO scheduling. The simulation used processed job data based on UniLu workload trace as in Section IV-A [3]. The size of the simulated system is the same as UniLu Gaia in Table I.

We compare the several scheduling methods and configurations described in Section III. First, in the SQ configuration, a single queue accepts both job types. Here a system parameter, the maximum multiplicity M, is configured. SQ with $M = 1$ is the simplest system, which does not allow oversubscribing, and is considered to impose long waiting times on jobs. SQ with $M > 1$ uses our proposed AO scheduling. As the comparison targets, we evaluate the MQ configuration that prepares a queue for batch jobs and one for interactive jobs.

[3]To evaluate the generality of our method, we also conducted a simulation using several modified data set, to achieve a similar tendency. They are omitted for page limitation

The ratio of interactive nodes is configured by the ratio R. Also, we allow oversubscribing on the interactive queue, with the maximum multiplicity of M_I.

A. Experiment 1: Effectiveness of Aggressive Oversubscribing

This experiment evaluates performance of job scheduling policies. To see effects of oversubscribing, we compare SQ configurations with $M = 1, 2, 3, 4$. $M = 1$ does not use oversubscribing. Also, we evaluate MQ configurations with $R = 1\%, 2\%, 3\%$, that correspond to different numbers of interactive nodes. On the interactive nodes in MQ, M_I is fixed at 4.

1) The responsiveness of interactive jobs: The purpose of the AO scheduling is to satisfy more interactive users' demand immediately, while restricting the number of coexisting processes. Table II shows the ratio of interactive jobs that suffer from waiting time. SQ configuration with $M = 1$ (no oversubscribing) imposes waiting time on 6.2% of jobs, while the rest (93.8% of jobs) are started immediately. With larger M using AO, the ratio is decreasing as expected, and when $M = 4$, all interactive jobs can be started immediately. We see MQ configuration also eliminates waiting times with $R \geq 2\%$. On the other hand, when $R = 1\%$, 8.3% experiences waiting time, which is larger than with "SQ, $M = 1$". In MQ, misconfiguration of the system (R value here) can have a large impact. Figure 4 (a) shows the longest waiting time among interactive jobs. We see as ratio in the Table II increases, the longest waiting time becomes worse.

TABLE II
RATIO OF INTERACTIVE JOBS THAT SUFFER FROM WAITING TIME.

Case	Ratio	Case	Raio
SQ, M=4	0%	MQ, R=3%	0%
SQ, M=3	0.7%	MQ, R=2%	0%
SQ, M=2	1.2%	MQ, R=1%	8.3%
SQ, M=1	6.2%		

2) Slowdown of jobs: With oversubscribing, the executions of both interactive jobs and batch jobs get slower. Also the waiting time extends the turnaround time visible to the users. To investigate these effects, we evaluate the "slowdown" metric, defined as $S = (T_w + T_r)/T_a$ for each job, where T_w is the waiting time. T_r and T_a are the execution time of the job; while T_a is the execution time if job were run alone (in our simulation, it is described in trace data), T_r includes overhead caused by oversubscribing (obtained from our simulator). $S = 1$ represents the ideal case.

Figure 4 (b) shows the largest slowdown among batch jobs and interactive jobs. With MQ, the largest slowdown of batch jobs is significantly larger than that of interactive jobs. Also, we see $MQ, R = 3\%$ rises 18,652x slowdown while it is 12,697x with $R = 2\%$, while the impact on interactive jobs is minor. Again we observe the sensitivity of configuration in MQ. In SQ, larger M contributes to a smaller slowdown both for batch jobs and interactive jobs. This property is expected to be helpful even for batch job users since it offers a shorter turnaround time.

In order to observe the tendency among all jobs, Figure 4 (c) shows the distribution of slowdown values among jobs. The X-axis is the cumulative ratio of jobs and the Y-axis is the slowdown value. Here interactive and batch jobs are mixed. We see "SQ, M=1" and MQ configurations show similar curves; while the slowdown of around 92% of jobs is one, the curve rises sharply. It means minor parts of jobs suffer from a very large slowdown. With SQ ($M \geq 2$), around 30% of jobs suffer from a slowdown, however, the slowdown ratio is smaller than 3 for most of such jobs. When $M = 4$, the largest slowdown is 4.8 among both job types. From this observation, we can say that SQ with AO scheduling can achieve fairer scheduling.

3) Efficiency of the system: Finally, we show the overall system efficiency. Table III shows makespan, the time until all the jobs are completed. SQ with $M \geq 3$ exhibits around 2.5% longer makespan, however, we observed this is due to a few jobs scheduled lastly in the duration. When we observe interim progresses, the time until 90% or 95% of jobs are completed, the results are similar among all the configurations.

TABLE III
SYSTEM EFFICIENCIES IN EXPERIMENT 1. MAKESPAN IS THE ENDING TIME OF THE LAST COMPLETED JOB. IT ALSO SHOWS THE TIMES WHEN 90% AND 95% OF JOBS ARE COMPLETED.

Case	Makespan [hour]	Finished time [hour]	
		90th	95th
SQ, M=4	2194	1859	2004
SQ, M=3	2192	1870	2016
SQ, M=2	2154	1883	2013
SQ, M=1	2138	1852	2015
MQ, R = 3%	2138	1851	2015
MQ, R = 2%	2138	1851	2015
MQ, R = 1%	2138	1850	2015

B. Experiment 2: Robustness to the ratio of interactive jobs

While the previous section used the fixed job data set, we consider that AO scheduling will have more advantages when interactive jobs are more dominant in HPC workloads. In order to confirm this expectation, we conduct the evaluation with artificial data sets with more interactive jobs. For this purpose, we simply modified the UniLu (original) trace; we pick some batch jobs with smaller execution time and convert them into interactive jobs to generate several sets, as shown in Table IV. Data set 6 includes 39,473 (76%) interactive jobs.

With SQ configuration, we fixed the maximum multiplicity $M = 4$, allowing oversubscribing. On the other hand, through the preliminary measurement with MQ, we observed it cannot follow the change of interactive jobs ratio at all. To give an advantage to MQ, we conducted an intensive parameter survey to determine the optimal system parameters for each data set. The parameters include R and M_I. Also now we allow oversubscribing even on the batch queue, with the maximum multiplicity of M_B. The found optimal parameters are shown in Table IV. For data set 6, the system needs to prepare 18% of dedicated nodes in the system for the interactive jobs for better performance.

Figure 5 shows the maximum slowdown for each case. Apparently, SQ ($M = 4$) achieves stable performance and

(a) Maximum waiting time (b) Maximum slowdown (c) Distribution of Slowdown

Fig. 4. The results of the experiment 1.

TABLE IV
THE GENERATED DATA SET AND OPTIMAL PARAMETERS IN MQ CONFIGURATION

	1.	2.	3.	4.	5.	6.
# of interactive jobs	1762	6500	15913	25336	34772	39473
ratio	0.034	0.125	0.307	0.488	0.670	0.761
Well-configured values for MQ						
R[%]	2	2	6	12	18	18
M_{I}	4	4	4	4	4	4
M_{B}	4	4	5	8	8	8

the slowdown values are less than 5 for all the jobs. With MQ configuration, we observe interactive jobs have a modest slowdown successfully, owing to optimized R for each data set. On the other hand, batch jobs suffer from a larger slowdown. These results are obtained even with the parameter survey. If the ratio of interactive jobs changed dynamically (such as the change between daytime and night), the performance with MQ would be much worse.

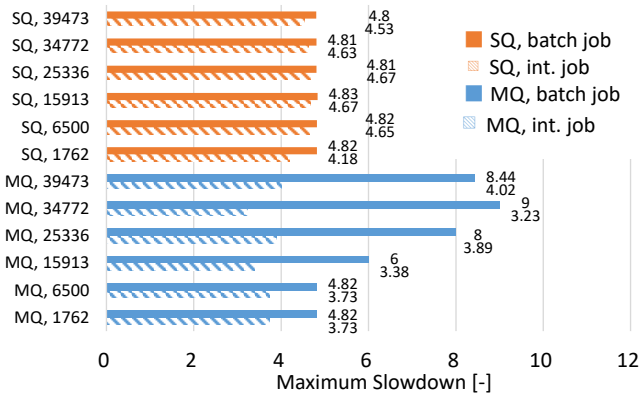


Fig. 5. The results of experiment 2; maximum slowdown with different trace data sets.

VI. Related work

Hofmeyr et al. have reported the performance of job scheduling with oversubscribing using their supercomputer's job data [14]. Their simulation assumes only batch jobs, with which processes are always busy. Also Slurm [8], a famous OSS job scheduler, already has the functions of oversubscribing and gang scheduling. As seen above cases, the idea of oversubscribing is well known for a long time, however, it is very rarely used in production supercomputers. We consider that this is largely because batch job users do not prefer preemption, which may incur unexpected issues. One of them has been dealt with by our previous work [7]; when job execution time gets longer, it may be killed by the system for time limit. We described a method to adjust time limit considering oversubscribing.

Interactive jobs have been well studied in system wide [15]–[20]. They built a system that immediately allocates computing resources to interactive jobs, after suspending batch jobs. Their main focus is the evaluation of the productivity of interactive users. On the other hand, this paper evaluates the impact of different job scheduling policies onto both interactive jobs and batch jobs.

VII. Conclusion and Future work

This paper evaluated the aggressive oversubscribing (AO) scheduling for supercomputers that support both interactive jobs and batch jobs. In order to conduct the evaluation with our NCS simulator, we collected information on real interactive jobs on a supercomputer and conducted an analysis. Also, we developed a simulation method that combines existing workload trace and CPU utilization data based on our collected data. From the evaluation, we confirmed the AO method is superior to the conservative (MQ) system configurations as follows. (1) It achieves high responsiveness for interactive jobs as a conservative system, (2) It alleviates the slowdown of batch jobs, (3) It does not need to configure the number of dedicated interactive nodes, and (4) It is more robust to the change of amounts of interactive jobs.

The future work includes:

- **Consideration of GPUs:** Many modern supercomputers have accelerators including GPUs. GPUs are important especially for interactive usage, not only visualization, but AI applications.
- **Development of Oversubscribing Scheduler System:** While our technique is evaluated only in simulation, we plan to develop system software. The current plan is to improve the OSS such as Slurm [8] or Open PBS [21].

ACKNOWLEDGMENT

This study is carried out using the TSUBAME3.0 supercomputer at Tokyo Institute of Technology, and is supported by Fujitsu Next Generation Computing Infrastructure Collaborative Research Cluster.

REFERENCES

[1] S. Matsuoka, T. Endo, A. Nukada, S. Miura, A. Nomura, H. Sato, H. Jitsumoto, and A. Drozd, "Overview of TSUBAME3.0, green cloud supercomputer for convergence of HPC, AI and big-data," *TSUBAME e-Science Journal*, vol. 16, pp. 2–9, 2017.

[2] M. Sato, Y. Ishikawa, H. Tomita, Y. Kodama, T. Odajima, M. Tsuji, H. Yashiro, M. Aoki, N. Shida, I. Miyoshi, K. Hirai, A. Furuya, A. Asato, K. Morita, and T. Shimizu, "Co-design for a64fx manycore processor and "fugaku"," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020, pp. 1–15.

[3] Jupyter Development Team, "Jupyter notebook," http://jupyter.org/, 2015.

[4] Google, "Google colaboratory," https://colab.research.google.com/, 2017.

[5] Amazon Web Services, "Amazon SageMaker," https://aws.amazon.com/sagemaker/, 2017.

[6] Microsoft, "Azure machine learning," https://azure.microsoft.com/services/machine-learning/, 2015.

[7] S. Minami, T. Endo, and A. Nomura, "Effectiveness of the oversubscribing scheduling on supercomputer systems," in *The International Conference on High Performance Computing in Asia-Pacific Region*, 2023.

[8] A. B. Yoo, M. A. Jette, and G. Mark, "Slurm: Simple linux utility for resource management," in *Workshop on job scheduling strategies for parallel processing*. Springer, 2003, pp. 44–60.

[9] S. Minami, "Node conscious ovesubscribing scheduler simulator." [Online]. Available: https://github.com/iwturnedaiw/NodeConsciousScheduler

[10] S. J. Chapin, W. Cirne, D. G. Feitelson, J. P. Jones, S. T. Leutenegger, U. Schwiegelshohn, W. Smith, and D. Talby, "Benchmarks and standards for the evaluation of parallel job schedulers," in *IPPS/SPDP '99/JSSPP '99*, 1999, p. 67–90.

[11] The University of Luxemburg, "The university of Luxemburg Gaia cluster log," https://www.cs.huji.ac.il/labs/parallel/workload/l_unilu_gaia/index.html.

[12] D. G. Feitelson, D. Tsafrir, and D. Krakov, "Experience with using the parallel workloads archive," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2967–2982, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731514001154

[13] S. Minami, T. Endo, and A. Nomura, "Measurement and modeling of performance of HPC applications towards overcommitting scheduling systems," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 2021, pp. 59–79.

[14] S. Hofmeyr, C. Iancu, J. Colmenares, E. Roman, and B. Austin, "Time-sharing redux for large-scale hpc systems," in *IEEE HPCC/SmartCity/DSS 2016*, 2016, pp. 301–308.

[15] A. Reuther, T. Currie, J. Kepner, H. Kim, A. McCabe, P. Michaleas, and N. Travinin, "Technology requirements for supporting on-demand interactive grid computing," in *2005 Users Group Conference (DOD-UGC'05)*, 2005, pp. 320–327.

[16] A. Reuther, J. Kepner, C. Byun, S. Samsi, W. Arcand, D. Bestor *et al.*, "Interactive supercomputing on 40,000 cores for machine learning and data analysis," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*, 2018, pp. 1–6.

[17] A. Reuther, C. Byun, W. Arcand, D. Bestor, B. Bergeron, M. Hubbell, M. Jones, P. Michaleas, A. Prout, A. Rosa, and J. Kepner, "Scalable system scheduling for hpc and big data," *Journal of Parallel and Distributed Computing*, vol. 111, pp. 76–92, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731517301983

[18] A. Reuther, J. Kepner, A. McCabe, J. Mullen, N. Bliss, and H. Kim, "Technical challenges of supporting interactive hpc," in *2007 DoD High Performance Computing Modernization Program Users Group Conference*, 2007, pp. 403–409.

[19] A. Reuther, C. Byun, W. Arcand, D. Bestor, B. Bergeron, M. Hubbell, M. Jones, P. Michaleas, A. Prout, A. Rosa, and J. Kepner, "Scheduler technologies in support of high performance data analysis," in *2016 IEEE High Performance Extreme Computing Conference (HPEC)*, 2016, pp. 1–6.

[20] C. Byun, J. Kepner, W. Arcand, D. Bestor, B. Bergeron, V. Gadepally, M. Houle, M. Hubbell, M. Jones, A. Kirby, A. Klein, P. Michaleas, L. Milechin, J. Mullen, A. Prout, A. Rosa, S. Samsi, C. Yee, and A. Reuther, "Best of both worlds: High performance interactive and batch launching," in *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, 2020, pp. 1–7.

[21] Altair Engineering, "OpenPBS Open Source Project." [Online]. Available: https://www.openpbs.org/