applied sciences

Article

Log Analysis-Based Resource and Execution Time Improvement in HPC: A Case Study

JunWeon Yoon [1], TaeYoung Hong [1], ChanYeol Park [1], Seo-Young Noh [2] and HeonChang Yu [3],*

1 Supercomputing Center, KISTI, 245 Daehak-ro, Yuseong-Gu, Daejeon 34141, Korea; jwyoon@kisti.re.kr (J.Y.); tyhong@kisti.re.kr (T.H.); chan@kisti.re.kr (C.P.)
2 Department of Computer Science, Chungbuk National University, Chungbuk 28644, Korea; rsyoung@cbnu.ac.kr
3 Department of Computer Science and Engineering, Korea University, Seoul 02841, Korea
* Correspondence: yuhc@korea.ac.kr; Tel.: +82-2-3290-2392

Received: 28 February 2020; Accepted: 9 April 2020; Published: 10 April 2020

Abstract: High-performance computing (HPC) uses many distributed computing resources to solve large computational science problems through parallel computation. Such an approach can reduce overall job execution time and increase the capacity of solving large-scale and complex problems. In the supercomputer, the job scheduler, the HPC's flagship tool, is responsible for distributing and managing the resources of large systems. In this paper, we analyze the execution log of the job scheduler for a certain period of time and propose an optimization approach to reduce the idle time of jobs. In our experiment, it has been found that the main root cause of delayed job is highly related to resource waiting. The execution time of the entire job is affected and significantly delayed due to the increase in idle resources that must be ready when submitting the large-scale job. The backfilling algorithm can optimize the inefficiency of these idle resources and help to reduce the execution time of the job. Therefore, we propose the backfilling algorithm, which can be applied to the supercomputer. This experimental result shows that the overall execution time is reduced.

Keywords: HPC; supercomputer; parallel computing; job scheduling; backfilling

1. Introduction

A batch job scheduler periodically monitors the status of computing resources in a cluster and distributes jobs efficiently. Such a batch job scheduler is called, in general, the Distributed Resource Management System (DRMS) [1], Workload Management System (WMS) [2], or simply a job scheduler.

The basic role of the scheduler is to accurately reflect the resource status [3]. This includes the various computational resource, such as licenses, CPU, and memory, as well as many-core based acceleration systems, such as GPU and Intel PHI (many-core architecture) [4], which are of recent interest. It also meets requirements such as the fair-share policy [5] to ensure the fair distribution of resource distribution after awareness of resources, preemption support for high-priority jobs, resource scalability assurance, and support for various system environments. Most job scheduler software reflects the user job environment, from job submission to termination, as well as the state of the inventory and system status of the entire managed object. It also stores various pieces of information related to job execution, such as job scripts, environment variables, libraries, waiting, and the starting and ending times for a job. In order to configure the cluster environment and manage batch jobs, the scheduler is chosen to reflect the characteristics of the computation environment in software and hardware perspective [6].

This study analyzes the job execution logs performed in the batch scheduler of the Tachyon2 system, which has been actively operated in the National Supercomputing Center at Korea Institute of

Science and Technology Information (KISTI) [7]. The acquired log of the supercomputer contains rich pieces of information related to submitted jobs, such as user information, execution time, resource size, job exit status, and so on [8]. When a large-scale job is submitted to the batch scheduler, it waits without using the returned compute resource until the requested resource is prepared. This supercomputer log shows that idle resources are not used during this waiting time, causing inefficient resource waste. Therefore, the efficiency of resource allocation has to be evaluated by the scheduling algorithm [9]. It can be seen that optimization can be performed by applying the backfilling. In this paper, we will show that turnaround time can be reduced by optimizing the inefficiency of waiting resources.

The rest of this paper is organized as follows: In Section 2, we will introduce system configurations of Tachyon2, which is the supercomputer used for our analysis. In Sections 3 and 4, we will introduce the backfilling algorithm and our experimental results, respectively. We will conclude this paper in Section 5.

2. System Configuration

2.1. System Overview

This study was carried out using a Tachyon2 supercomputer that is currently in service. This system consists of various hardware and software stacks. On the hardware side, it is made of compute nodes, storage, backup archivers, and infrastructure nodes (login, data mover, scheduler, admin, etc.). In Tachyon2, each machine is connected by an InfiniBand interconnect [10]. For software, it works with a variety of software stacks, from the system OS to the application layer. As shown in Figure 1, KISTI's Tachyon2 supercomputer is a Linux-based cluster system with 3200 computing nodes. Each node is equipped with two quad-core Intel Xeon X5570 2.93GHz (Nehalem) CPUs, each of which has 8 cores. And DDR3 type of 24GB memory is installed in 3 channels. In this system, the Sun Grid Engine (SGE) is being used as a batch job scheduler [11].

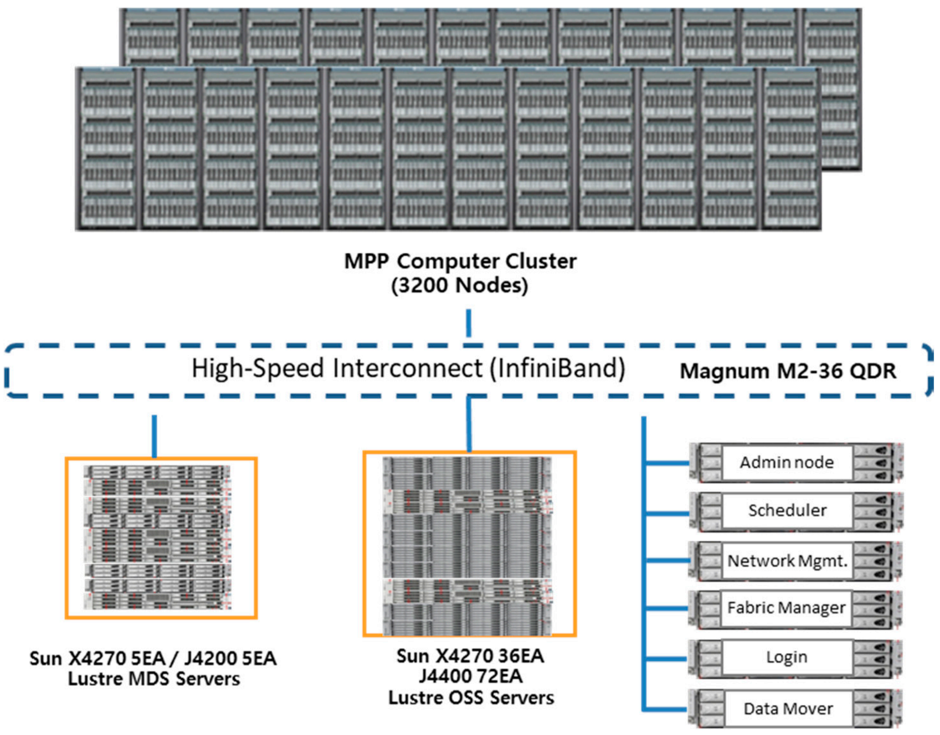


Figure 1. Diagram of KISTI's Tachyon2 supercomputer.

2.2. Batch Job Scheduler

Scheduler is a program that manages the order and schedule by giving priority to fairness of numerous submitted jobs. There are various solutions, such as a Sun Grid Engine (SGE), Slurm [12], LSF [13], Portable Batch System (PBS) [14], LoadLeveler [15], etc. Tachyon2 is using SGE to handle batch jobs. With such a scheduler, users can submit numerous jobs without concerning where and when the work will be assigned to computing resource and performed [16].

In order to execute an application on the supercomputer, the user writes a job specification script, as shown in Table 1, and submits the job script to the scheduler using the "qsub" command. When an available resource is acquired while waiting in the queue, the job is allocated to that resource and executed on that system. When assigning jobs, the policy for job assignment has to work. In Tachyon2, the exclusive node assignment policy is configured and applied. This means that only one job is assigned to a node, and interference can be eliminated when multiple jobs are executed simultaneously.

Table 1. Batch job script for submission.

#!/bin/bash	1	
#$ -V	2	Pass the environment variabe
#$ -cwd	3	Use the current working directory
#$ -w e	4	Verify options and abort if there is an error
#$ -N Job_name	5	Job name
#$ -pe mpi_4cpu 8	6	Parallel environment
#$ -q normal	7	Queue name to be execute
#$ -R yes	8	Resource reservation trigger
#$ -wd /scratch/$User/job_dir	9	Working directory
#$ -l h_rt=10:00:00	10	Maximum running time
#$ -l OMP_NUM_THREADS = 2	11	OpenMP threads per MPI process
#$ -M myEmailAddress	12	Email address to notify
mpirun -hostfile $TMPDIR/machines -np $NSLOTS userjob.exe	13	User application

Figure 2 shows the execution process of parallel application in the high-performance computing (HPC) batch-type environment. In general, all batch jobs are processed through the scheduler, the status of resources is continuously monitored, and abnormal resources are excluded from available resources. In this system, a job is not preempted by another until it is submitted to the scheduler, executed, and finished [17].

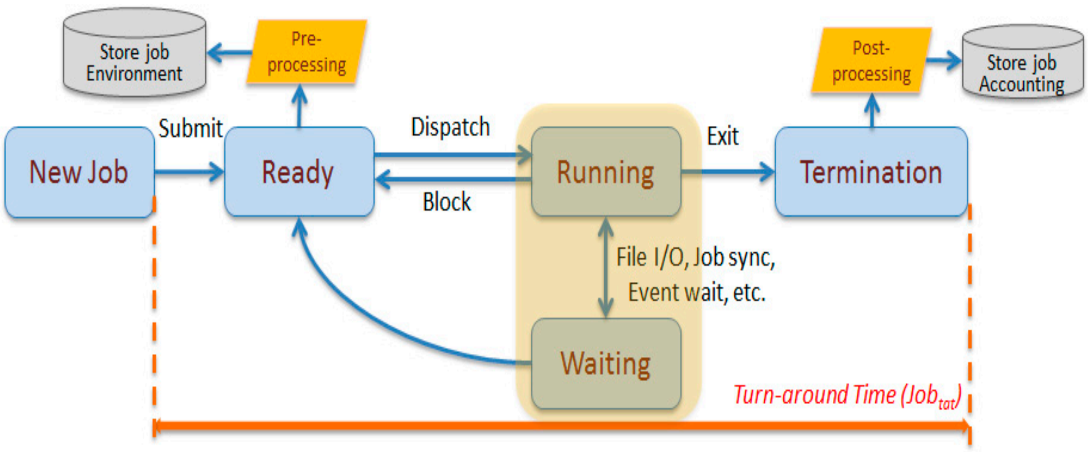


Figure 2. Procedure for the job running using a batch job scheduler.

The turnaround time of a job (Job_{tat}) is the time from when the user submits the job to get the results, which can be optimized according to the scheduling algorithm [18]. The steps of each process related to the work execution can be represented as follows:

$$Job_{tat} = T_{ready} + T_{running} + T_{processing\ (pre,\ post)} \tag{1}$$

$$T_{running} = T_{cal} + T_{comm} + T_{io} \tag{2}$$

- Job_{tat}: Amount of time from submission to termination of the job;
- T_{ready}: Waiting time for available resources in the ready queue;
- $T_{running}$: Real-time of user job;
- $T_{processing}$: Pre or post-processing time to store job information;
- T_{cal}: Job computation time (CPU bound);
- T_{comm}: Communication time (Network bound);
- T_{io}: I/O time between compute nodes and filesystem.

Ultimately, Job_{tat} consists of waiting time before submitting a job, real job running time, and preprocessing time. Scheduling optimization here can effectively reduce the latency of the job (T_{ready}). In this paper, we focus and simulate Job_{tat} reduction by optimizing T_{ready} through the backfilling algorithm.

2.3. Job Execution Logs

Tachyon2 stores information about job execution at the start and endpoint. When a job is submitted, it is queued while available resources are ready. When available resources are obtained, the job is assigned to the compute resources. At this time, the user's job description script is backed up, and the library, compiler, and node information required to run the program are stored. At this time, the user's job scripts, libraries and compilers required to run the program, and the allocated resource information are stored [19].

When the job is completed on the Tachyon2 system, the scheduler (SGE) leaves the job execution information, then regenerates this log and saves it again daily in the format shown in Table 2. This data is used in accounting, statistical extraction, user job tracking, and so on. This log contains the log recording time, unique job ID, job name and date information. In addition, such a log format contains several important pieces of date information that are related to the job. For example, Submit-Date, Start-Date, and End-Date are representing the submission, starting, and ending date information in the format of *yyyymmddhhmmss*. While such date information is provided in the perspective of jobs, there are pieces of resource utilization perspective information, such as Wait-Time (T_{ready}) and Execution-Time ($T_{running}$). Such time information provides how long the jobs were queued for waiting resources and utilized assigned computing resources. Finally, the log format also provides pieces status information, such as the number of used CPU, Exit-Code, Failed-Code, and the number of threads [6].

As mentioned earlier, Tachyon2 uses an exclusive policy that executes only one job per node. Even if the number of cores used by a job is 4 (CPUS), it is subtracted after multiplying by 8 (E-CPU), which is the total number of cores installed on the node. Resource usage is calculated for all nodes on which the job has been run. Each of the CPU USAGE (s), MEM USAGE (KB), and MAX VMEM (KB) is expressed as the sum of CPU time, memory usage, and virtual memory usage of all nodes where the job is executed. Normally, jobs in HPC are executed using multiple resources, and the scheduler recognizes the events of each processor at the end of the job and logs the status. Most of the job is normally completed and then terminated, but it can be abnormally terminated for reasons such as forced termination by the user, error in the job submission script, exceeding the wall time limit of the job scheduler, software and hardware problems, etc. At the end of the job, the scheduler leaves each signal code according to the above causes [20]. The code for this cause is logged in the EXIT-CODE

(#19) and FAILED (#20) fields. In the case of the OpenMP (Open Multi-Processing) code, the number of threads is stored in the OMP_NUM_THREADS (# 21) field.

Table 2. Job executed information of the converted Sun Grid Engine (SGE) log.

No.	Property	Value(Example)	No.	Property	Value(Example)
1	DATE	20181029	12	CPUS (cores)	128
2	JOBID	2310609	13	CPU USAGE (s)	2521538.58
3	GID	na0***	14	MEM USAGE (KB)	2269836.14
4	UID	r000***	15	MAX VMEM (KB)	345363.75
5	JOBNAME	mpi_solver	16	STATUS	D
6	QNAME	nomral	17	E-CPU(cores)	256
7	SUBMIT(DATE)	20181026111256	18	E-RUN(s)	11343872
8	START(DATE)	20181028191152	19	EXIT-CODE	0(128)
9	END(DATE)	20181029050924	20	FAILED	0(128)
10	WAIT(s)	444543	21	OMP_NUM_THREADS	1
11	RUN(s)	35852	22	TASK_NUM	0

This study confirms that the utilization of idle resources is effectively used when the large-scale job is submitted in Tachyon2. It extracts statistical data and analyzes patterns of large-scale jobs using thousands of cores or more from the job log of Tachyon2 stored in the form of Table 2.

2.4. Analysis of Job Execution Logs

This section describes the way to optimize resource utilization through the analysis of large-scale jobs. It was mentioned that the Tachyon2 system is composed of 3200 compute nodes and has 8 cores per node. In total, it consists of 25,600 cores. The scheduler divides resource groups through queues. In this system, queue groups are largely divided into two types: public queues and private queues [21]. The public queue is used by various researchers to share computational resources through competition, and the exclusive queue is assigned to groups for specific mission-oriented research. The size of the computational resources allocated to the public queue and the exclusive queue can be changed depending on the purpose of the study.

This experiment focused on log data from public queues used through resource competition and conducted experiments for statistical analysis of the data. The first approach extracted the number of large-scale jobs using more than 2000 cores over a two-year period on a monthly basis. As shown in Figure 3, the number of large-scale jobs is being gradually increased and more than 4000 cores were concentrated from November 2016 to March 2017. Figure 4 shows the trend of available resources during the same period. The *All Available Resources (core)* shows the size of the resources allocated to the public queue. As discussed in Section 2.3, the exclusive policy was used in the Tachyon2 system. Therefore, the difference between *Idle Resources (cores)* and *Avail Resources (Cores)* represents the sum of the total number of unused resources and the number of cores of unused nodes. The usage statistics of the Tachyon2 system during the period are as follows.
- Statistic results of the resource usage (January 2016–December 2017)

- Average waiting time (T_{ready}): 9.3 h
- Average turnaround time (T_{ready}): 15.3 h
- Average node usage rate: 85.69%

As shown in the summary, the average waiting time for public queues reaches 9.3 hours. However, the actual overall node utilization is only about 85%. This means that even though about 15% of the nodes were available, resources could not be used efficiently due to scheduling load. In other words, as shown in Figures 3 and 4, when a large-scale job is submitted, it waits until the required resources are available. In the meantime, even if a small job can be executed, it should wait if the priority is low. Therefore, even if a job requires a small number of resources and is performed shortly if the job priority is low, it must wait until the large-scale job is finished, even if there is an available resource.

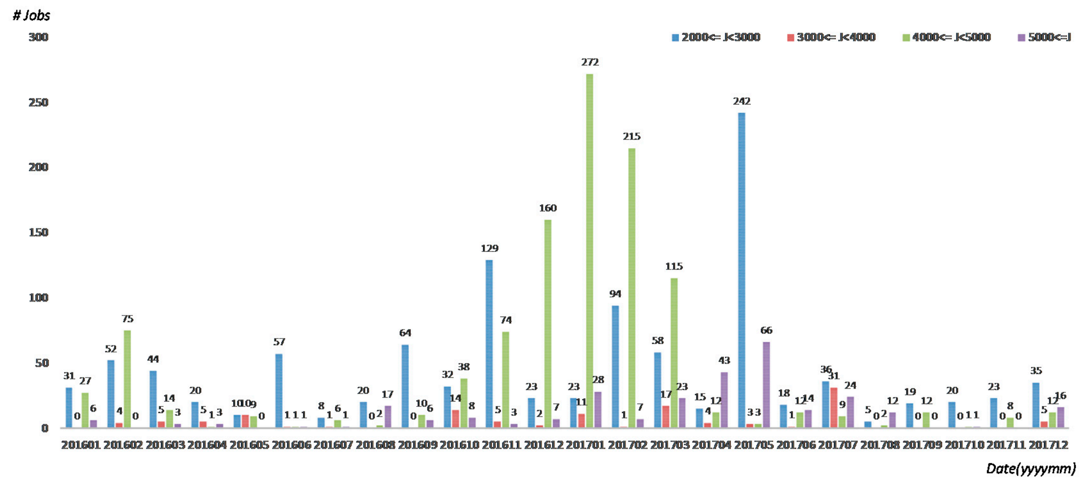


Figure 3. Statistics of the large-scale jobs counts.

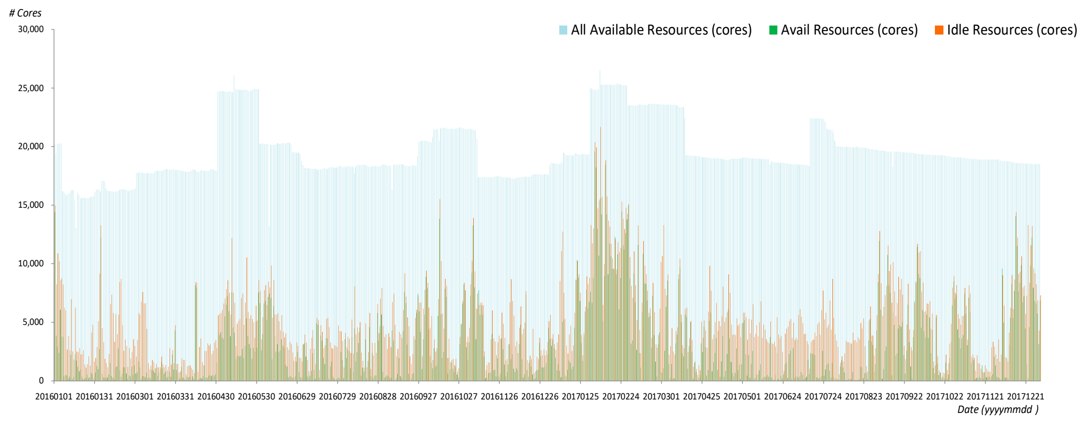


Figure 4. Statistics of the public queue available nodes.

The algorithm adopted by default in the scheduler is the First Come First Serve (FCFS) method, where the first arriving task in time is executed first. FCFS is the best way to ensure fairness of job order, but fragmentation occurs as the size of the computed resource increases, limiting the efficient use of resources. [22].

All jobs have their priorities. Such priorities are given, in general, based on arrival time. In addition, all jobs have different resource requirements. Therefore, it should be emphasized that actual available resources are not used if a high priority job holds those resources. Moreover, it is also noted that such available resources cannot be fragmented and shared for lower priority jobs. Figures 3 and 4 show the trend of increasing available resources as the number of large-scale operations increases towards the end of 2016 and early 2017.

The most extreme way to reduce resource fragmentation is to allocate the shortest job first, which is called the Shortest Job First (SJF) algorithm [23]. Such an algorithm prioritizes small jobs according to fragmented resources. This can improve resource utilization and improve overall performance, but it does not guarantee the fairness of job order. Therefore, it is worth noting that, in the worst case, the algorithm might result in the starvation problem for large-scale jobs. In that sense, the scheduling policy should be used in a way that combines FCFS and SJF considering resource size and job characteristics [24].

Most job scheduling policies simply use an FCFS method in which jobs submitted first to the queue are executed first. The backfilling algorithm is an excellent method for solving the resource fragmentation, but only simple and passive techniques are applied in a site-level service environment.

Nurion system, the next generation of the Tachyon2 system that was the background of this study, uses Portable Batch System (PBS) as a scheduler [25].

PBS uses the *backfill_depth* parameter, which is the number of backfill targets, for the top jobs with the highest priority among the queued job list [26]. PBS can backfill only a few lower priority jobs (*backfill_depth*) based on the highest priority parent job in the list of held jobs.

In addition, when a specific event occurs, the scheduler updates the resource status and the job profile, which is called a scheduling cycle [27]. The events that trigger the scheduling cycle include a certain period of time when submitting and terminating a job, when a scheduling server starts, and when a new configuration of a scheduler is applied, and so on. In the batch scheduler, the job size consists of the number of processors and the execution time, where the execution time is estimated by the user and specified in the job script and then submitted to the scheduler. Therefore, the actual job execution time can often be terminated earlier than the expected time. In this case, if the job ends earlier than expected, resource fragmentation occurs again and the backfill target (*backfill_depth*) job fills up. If this situation repeats and small jobs continue to be backfilled, eventually, the top job will not run and will lead to starvation. In general, to avoid this starvation, production level systems set the *backfill_depth* value very low to adjust the number of backfill operations. In the case of the Nurion system, the value is set to 3 or less. Because of this weakness, if the scheduler limits the number of backfill jobs to a setting like *backfill_depth*, it will not be able to take full advantage of fragmented resource utilization. Therefore, in this study, the simulation is performed by utilizing all fragmentation resources without limiting the number of jobs that can be backfilled as follows.

3. Backfilling Algorithm

Backfilling scheduling is a method of rearranging the order when a small job cannot be performed due to a relatively large predecessor job This aims to improve performance by placing small-scale jobs that can be executed first, while maintaining fairness. In order to apply this algorithm, the user must specify the size and execution time of the task.

This algorithm can improve performance with fairness. In backfilling scheduling, the execution time and size of each job must be specified. In this work, we traced and analyzed the job logs executed by the scheduler. As a result, we identify the idle resources that have not been used for a certain period of time and perform an experiment to optimize resource utilization by applying a backfilling algorithm [28].

The conservative backfilling algorithm is a basic version, and it adheres to the FCFS scheme, which is the basic principle of scheduling, and running first when a subordinate job satisfies a fragmented resource. In general, batch jobs submitted to the scheduler have attributes for required resources and execution time. The backfilling algorithm also has these two properties. For this mechanism, there are two data structures. The first is a list structure that stores jobs and execution time of the queue list and the second is the resource processor profile to be used.

This algorithm requires no latency for prior jobs due to subordinate jobs but cannot guarantee the planned sequence of jobs if the preceding job is terminated earlier than the expected time. Therefore, there is a more advanced algorithm for this, which is called EASY (the Extensible Argonne Scheduling System) backfilling [28,29]. However, the experiment in this study simulates the performance improvement through backfilling at a fixed time for a large-scale job and therefore excludes the case where the job ends before the expected time. Figure 5 and Algorithm 1 show the conservative backfilling algorithm and how it works, respectively.

The 1st priority queued job does not have enough resources to run, so it will be scheduled after the two running jobs (*a*) and (*b*) have ended. The 2nd priority queued job finds the available resource point (t_1), as indicated by the dotted line in Figure 5. Then, from this point t_1 to the end of the execution, it checks whether it is possible to free up resources. However, at t_2, the 1st priority queued job is waiting and the 2nd priority queued job will delay it. The 2nd priority queued job can potentially free up resources if only one predecessor is terminated. However, the execution time of this job delays the predecessor, so a new start point must be found. In other words, it is a mechanism that prevents future arrivals from delaying previous pending operations. Finally, the 3rd priority queued job is backfilling because there is enough gap to do the job.

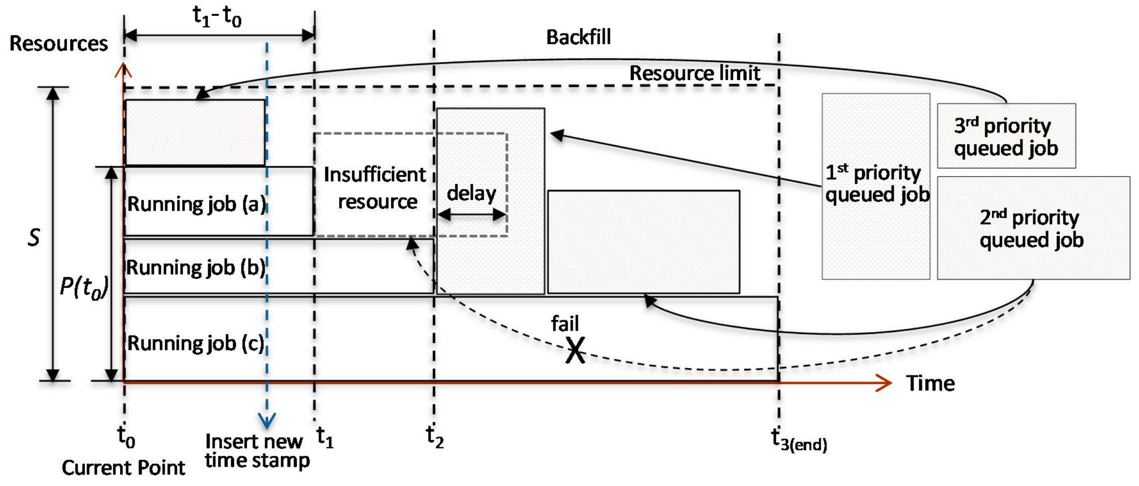


Figure 5. Conservative backfilling.

Algorithm 1: Conservative backfilling

```
data: ·Running job is $J_i$,
      Queued job $Q_j(p,t)$ – $Q_j(p)$ is required processors, $Q_j(t)$ is required time
      Total available resource of system is $S$
      Total used resource at time $t_i$ is $P(t_i)$

1  for queued job $Q_j \in Q$ do
2        Sort by execution time($\forall J$)
3     if $S\text{-}P(t_i) \geq Q_j\ (p)$ then
4        if $t_i - t_{i-1} > Q_j\ (t)$ then
5            JobAllocation();
6              $P(t_i) \leftarrow P(t_i) + Q_j\ (p)$;      // Resource update
7              $T_{i+1} \leftarrow T_i + Q_j\ (t)$;        // New time stamp insert
8              For time stamp index from $T_{i+1}$ to $T_{end}$ do
9                  IncreaseIndex($T_{i+1}$);
10             end for
11             $j \leftarrow j + 1$;
12       else
13             $i \leftarrow i + 1$;
14           next;
15       end if
16    else
17           $i \leftarrow i + 1$;
18       next;
19    end if
20 end for
```

4. Simulation and Results

In our experiment, we have simulated certain large-scale jobs that were run on Tachyon2. This system has a scheduling formula based on the job size weighting, but the backfilling algorithm is not applied. Figure 6 shows the scenario of the conservative backfilling algorithm applied to Tachyon2. As discussed, the queued job has two properties, which are the resource and time requirements [30]. Here, it is assumed that the priority is higher as the number is smaller. If large-scale job #0 is inserted into the queue list, it should wait until it has a suitable resource. The conservative backfilling can execute low priority jobs (box #1~#6) first during the waiting time of job #0. Job #6 has a lower priority than #5, but it runs first because the available resources are acquired first and do not delay predecessor #5.

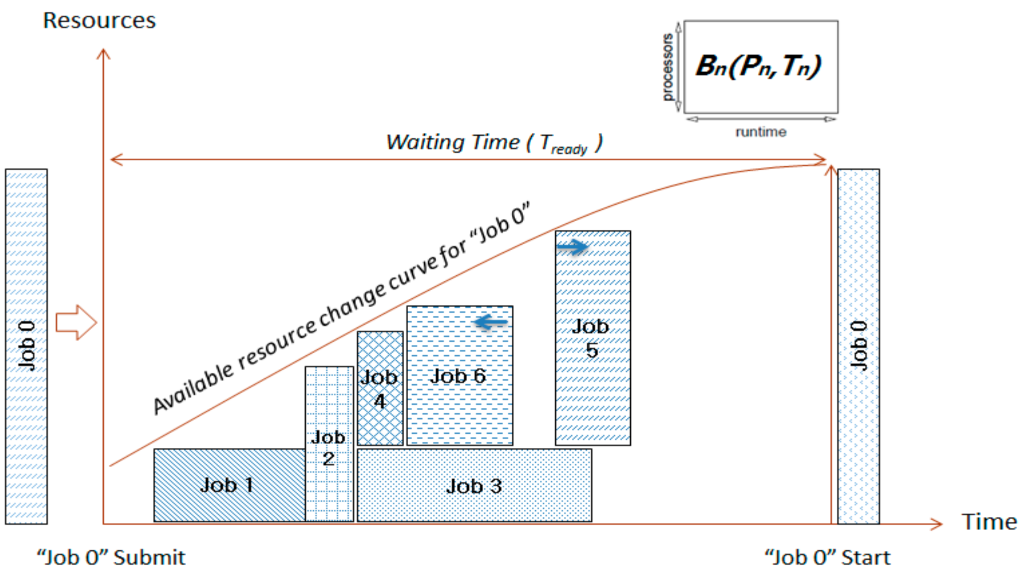


Figure 6. Applying backfilling to available resources before the large-scale job.

As shown in the graphs of Figures 3 and 4, the simulation is performed by selecting the job that were the main cause of the increase in the number of available nodes due to large-scale operation. This job was a large-scale job that required 2048 nodes (16,384 cores, 8 cores/node), and the job ran after waiting about 40 hours and 16 minutes. During the waiting time, the computing nodes are emptied to match the requested resources of the job, as shown in Figure 7.

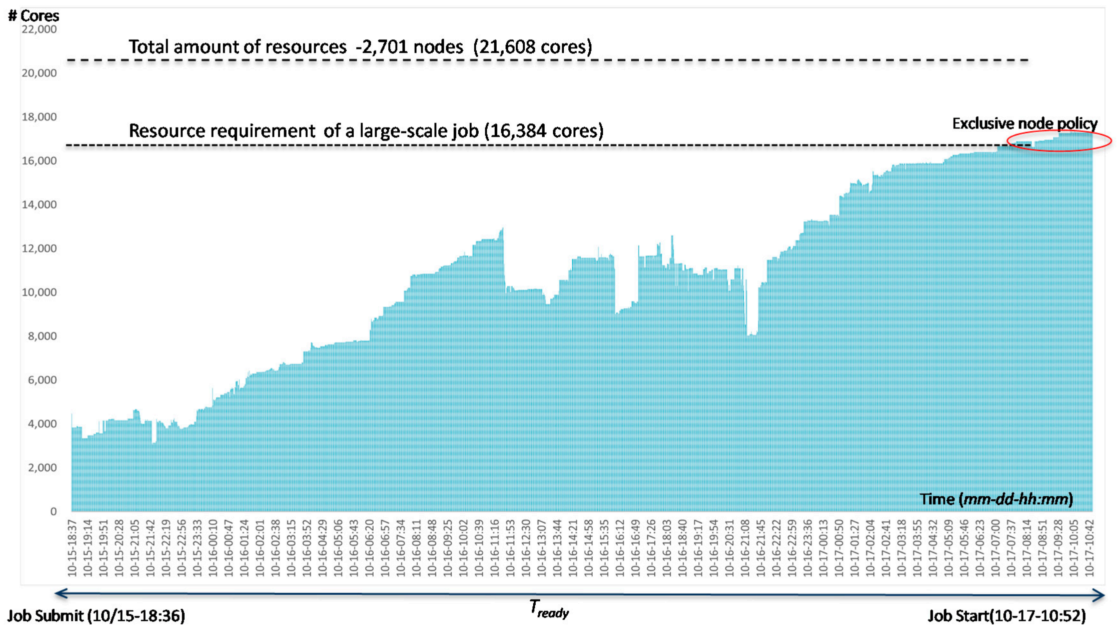


Figure 7. Increase in available resources for the resource allocation of large jobs.

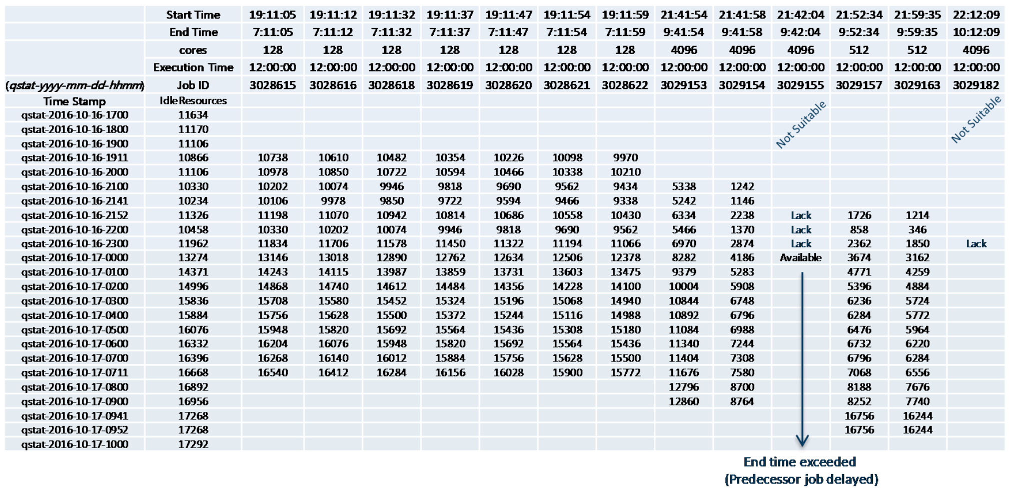
As a result, the latency of the entire subordinated job increases (Job_{tat}). If a small job with a low priority leapfrogs first for the resources that are emptied during the waiting time, it can use the resources efficiently. When a job is submitted to the scheduler, it has a unique job ID, submission time, resource requirements, execution, and queue status, as shown in Figure 8a. The queue status indicates whether the job is currently waiting (*qw*), running (*r*), pending (*h*), or terminating (*e*). This information can be checked with a command such as *qstat* provided by the scheduler. The simulation procedure is performed as follows:

① Extract jobs that do not exceed 2048 nodes between submission time and start time (Waiting time, T_{ready}) during subordinated jobs, as shown in Figure 8a;
② Backfilling is performed on the extracted jobs, as shown in Figure 8b. Each timestamp specifies the available idle processors (total number of cores);
③ Resource profile update

- Scans the profile (request resource) and finds the first point in the available processor. This operation is performed for the job to be backfilled in ①.
- Starting from this point, it continues to scan the processor to see if it is available as a profile (request resource) until the job is terminated as expected. Figure 8 shows the updated resources profile. Conservative backfill does not delay the start time of predecessors. Therefore, job 3029155 is excluded because it ends after the start time of the target job. Moreover, job 2019182 is out of resources due to the backfill operation in front. The calculation process is shown in Appendix A.

JobID	Priority	Q Status	Submit Time		Usage (Cores)	Execution Time	Expected End Time	
3028615	0.51397	qw	10/16/2016	19:11:05	128	12:00:00	10/17/2016	7:11:05
3028616	0.51397	qw	10/16/2016	19:11:12	128	12:00:00	10/17/2016	7:11:12
3028618	0.51396	qw	10/16/2016	19:11:32	128	12:00:00	10/17/2016	7:11:32
3028619	0.51396	qw	10/16/2016	19:11:37	128	12:00:00	10/17/2016	7:11:37
3028620	0.51396	qw	10/16/2016	19:11:47	128	12:00:00	10/17/2016	7:11:47
3028621	0.51396	qw	10/16/2016	19:11:54	128	12:00:00	10/17/2016	7:11:54
3028622	0.51396	qw	10/16/2016	19:11:59	128	12:00:00	10/17/2016	7:11:59
3029153	0.51419	qw	10/16/2016	21:41:54	4096	12:00:00	10/17/2016	9:41:54
3029154	0.51419	qw	10/16/2016	21:41:58	4096	12:00:00	10/17/2016	9:41:58
3029155	0.51419	qw	10/16/2016	21:42:04	4096	12:00:00	10/17/2016	9:42:04
3029157	0.51176	qw	10/16/2016	21:52:34	512	12:00:00	10/17/2016	9:52:34
3029163	0.51166	qw	10/16/2016	21:59:35	512	12:00:00	10/17/2016	9:59:35
3029182	0.51373	qw	10/16/2016	22:12:09	4096	12:00:00	10/17/2016	10:12:09

(a) Extraction of backfilling enabled jobs.



	Start Time	19:11:05	19:11:12	19:11:32	19:11:37	19:11:47	19:11:54	19:11:59	21:41:54	21:41:58	21:42:04	21:52:34	21:59:35	22:12:09
	End Time	7:11:05	7:11:12	7:11:32	7:11:37	7:11:47	7:11:54	7:11:59	9:41:54	9:41:58	9:42:04	9:52:34	9:59:35	10:12:09
	cores	128	128	128	128	128	128	128	4096	4096	4096	512	512	4096
	Execution Time	12:00:00	12:00:00	12:00:00	12:00:00	12:00:00	12:00:00	12:00:00	12:00:00	12:00:00	12:00:00	12:00:00	12:00:00	12:00:00
(*qstat-yyyy-mm-dd-hhmm*)	Job ID	3028615	3028616	3028618	3028619	3028620	3028621	3028622	3029153	3029154	3029155	3029157	3029163	3029182
Time Stamp	Idle Resources													
qstat-2016-10-16-1700	11634													
qstat-2016-10-16-1800	11170										Not Suitable			Not Suitable
qstat-2016-10-16-1900	11106													
qstat-2016-10-16-1911	10866	10738	10610	10482	10354	10226	10098	9970						
qstat-2016-10-16-2000	11106	10978	10850	10722	10594	10466	10338	10210						
qstat-2016-10-16-2100	10330	10202	10074	9946	9818	9690	9562	9434	5338	1242				
qstat-2016-10-16-2141	10234	10106	9978	9850	9722	9594	9466	9338	5242	1146				
qstat-2016-10-16-2152	11326	11198	11070	10942	10814	10686	10558	10430	6334	2238	Lack	1726	1214	
qstat-2016-10-16-2200	10458	10330	10202	10074	9946	9818	9690	9562	5466	1370	Lack	858	346	
qstat-2016-10-16-2300	11962	11834	11706	11578	11450	11322	11194	11066	6970	2874	Lack	2362	1850	Lack
qstat-2016-10-17-0000	13274	13146	13018	12890	12762	12634	12506	12378	8282	4186	Available	3674	3162	
qstat-2016-10-17-0100	14371	14243	14115	13987	13859	13731	13603	13475	9379	5283		4771	4259	
qstat-2016-10-17-0200	14996	14868	14740	14612	14484	14356	14228	14100	10004	5908		5396	4884	
qstat-2016-10-17-0300	15836	15708	15580	15452	15324	15196	15068	14940	10844	6748		6236	5724	
qstat-2016-10-17-0400	15884	15756	15628	15500	15372	15244	15116	14988	10892	6796		6284	5772	
qstat-2016-10-17-0500	16076	15948	15820	15692	15564	15436	15308	15180	11084	6988		6476	5964	
qstat-2016-10-17-0600	16332	16204	16076	15948	15820	15692	15564	15436	11340	7244		6732	6220	
qstat-2016-10-17-0700	16396	16268	16140	16012	15884	15756	15628	15500	11404	7308		6796	6284	
qstat-2016-10-17-0711	16668	16540	16412	16284	16156	16028	15900	15772	11676	7580		7068	6556	
qstat-2016-10-17-0800	16892								12796	8700		8188	7676	
qstat-2016-10-17-0900	16956								12860	8764		8252	7740	
qstat-2016-10-17-0941	17268											16756	16244	
qstat-2016-10-17-0952	17268											16756	16244	
qstat-2016-10-17-1000	17292													

(b) Job profile (resource, time), update, and exception handling of out of resource jobs.

Figure 8. Applying backfilling for a large-scale job.

Resource Efficiency can be obtained as Equations (3)–(5). The backfilling job has two properties (Resource usage (P) and Runtime (T)), as shown in Figure 5. The resource efficiency by backfilling scheduling is measured as follows:

$$\text{Backfilling jobs: } B_1(P_1, T_1), \cdots, B_n(P_n, T_n) \tag{3}$$

$$\text{Resource Efficiency} = \sum_{i=1}^{n} P_i T_i \tag{4}$$

$$\begin{aligned} \text{R}(128\text{cores} \times 12\text{hours}) \times 7\text{jobs} + (4,096\text{cores} \times 12\text{hours}) \times 2\text{jobs} \\ +(512\text{cores} * 12\text{hours}) \times 2\text{jobs} \\ = 72,192\text{seconds (Reduced overall job execution time)} \end{aligned} \tag{5}$$

Backfill scheduling recognizes the status of jobs and resources every scheduling cycle event covered in Section 2.4 and attempts scheduling policy. This is because, as mentioned above, the status of the job changes continuously depending on the user's intention or the state of the system. In this study, the simulation was performed while reflecting the job profile for each timestamp. The target job of the experiment was shortened by about 20 hours through a backfill compared to FCFS. If this study is applied to other similar large-scale jobs, the resource efficiency is much higher and the overall job execution time (Job_{tat}) can be greatly reduced.

5. Conclusions and Future Work

Parallel computing is a suitable solution for solving large-scale problems. The main content of this study is to optimize the order of jobs executed in the Tachyon2 system to efficiently use the entire available resources. This experiment analyzed the actual logs of job execution using the converted log in the Tachyon2 system. Note that the average utilization of the Tachyon2 system is about 85.6% and the average wait time is about 9.3 hours. This can be seen as a fragmentation of resources that occur during scheduling. Of course, it is not possible to use all the resources perfectly to reflect both usage and execution time. However, it is possible to minimize resource fragmentation occurring during job scheduling and to make resource utilization more efficient by analyzing the jobs performed on the computational resources. Such utilization can be more improved by studying and applying the appropriate scheduling algorithms. One way is to analyze the statistics of large-scale operations, which are gradually increasing in the Tachyon2 system, to grasp the user's success rate, execution time, and resource size. Based on the statistical results, the backfill scheduling algorithm is studied and simulated to reduce resource fragmentation in Tachyon2. This study effectively used available resources and reduced turnaround time for backfilled tasks. In the end, we improved the performance of the overall scheduler.

This paper focuses on the simulation of the backfill scheduling algorithm that analyzes the supercomputer's job statistics and targets cases of inefficient resource use. Although it is useful to understand how to improve the utilization of an actively working system, it is necessary for us to apply the various scheduling algorithms to the current system and analyze the work execution history continuously and repeatedly as future work. In that sense, research is needed to apply advanced algorithms that reflect when a job is done earlier than expected or when available resources are suddenly added. Such research will optimize resource utilization and reduce overall user latency. In this experiment, our main target system was the Tacyon2 supercomputer. As of writing, the next generation of KISTI's 5th supercomputer Nurion consists of 8400 compute nodes based on many-core architectures and production services launched in December 2018. Therefore, it is worth evaluating newly delivered supercomputers with the same approach to find the resource underutilization problem, ultimately providing more resources to scientists for better and faster scientific outcomes.

Author Contributions: Conceptualization, J.Y.; Methodology, J.Y., S.-Y.N. and C.P.; Data curation, J.Y.; Software, J.Y.; Validation, J.Y., S.-Y.N. and H.Y.; Investigation, J.Y. and C.P.; Writing—Original Draft Preparation, J.Y.; Writing—Review & Editing, J.Y. and S.-Y.N.; Supervision, H.Y.; Project administration, T.H.; Funding acquisition, T.H. All authors have read and agreed to the published version of the manuscript.

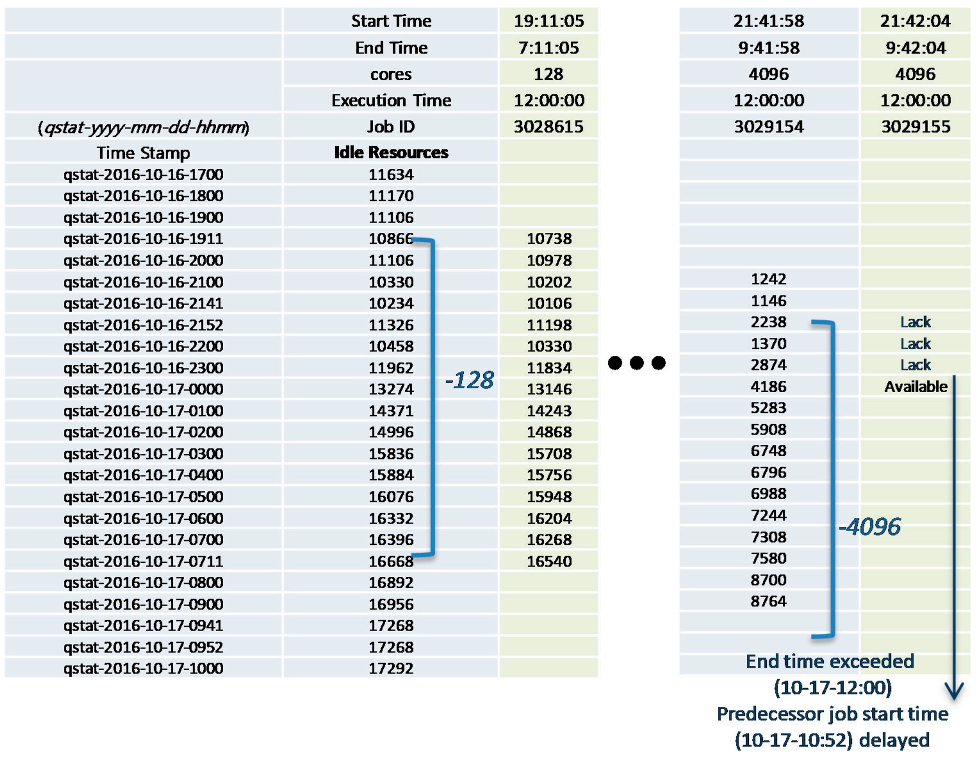
Funding: This research has been performed as a subproject of Project No. K-19-L02-C01-S01 (The National Supercomputing Infrastructure Construction and Service) supported by the KOREA INSTITUTE of SCIENCE and TECHNOLOGY INFORMATION(KISTI).

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

Jobs with a lower priority than the larger job being simulated in the queue are listed in Figure 8a. These jobs are targeted for backfilling jobs and should not delay the start time of the original job. This is the basic principle of a conservative algorithm. Each job has its own resource requirements, such as the number of processes and the execution time. The scheduler calculates resource usage at regular intervals (time stamp), and this time is performed for each scheduling cycle mentioned in the previous text. *Idle Resources* in Figure A1 shows the total resources available when resources are allocated or released by a task.

For example, when the first backfilling job (*Job ID: 3028615*) requests 128 cores, it allocates resources from the start time to the end time and update *Idle Resources*. In the case of the backfilling job (Job ID: 3029154), the job is submitted at 21:42:04 and requests 4096 resources, but, at that point, there is not enough resources. After about 2 hours, the available resources are released at 00:00, but this job cannot be backfilled because the 12-hour execution time delays the original job.


-128
●●●
-4096
End time exceeded
(10-17-12:00)
Predecessor job start time
(10-17-10:52) delayed

	Start Time	19:11:05	21:41:58	21:42:04
	End Time	7:11:05	9:41:58	9:42:04
	cores	128	4096	4096
	Execution Time	12:00:00	12:00:00	12:00:00
(*qstat-yyyy-mm-dd-hhmm*)	Job ID	3028615	3029154	3029155
Time Stamp	**Idle Resources**			
qstat-2016-10-16-1700	11634			
qstat-2016-10-16-1800	11170			
qstat-2016-10-16-1900	11106			
qstat-2016-10-16-1911	10866	10738		
qstat-2016-10-16-2000	11106	10978		
qstat-2016-10-16-2100	10330	10202	1242	
qstat-2016-10-16-2141	10234	10106	1146	
qstat-2016-10-16-2152	11326	11198	2238	Lack
qstat-2016-10-16-2200	10458	10330	1370	Lack
qstat-2016-10-16-2300	11962	11834	2874	Lack
qstat-2016-10-17-0000	13274	13146	4186	Available
qstat-2016-10-17-0100	14371	14243	5283	
qstat-2016-10-17-0200	14996	14868	5908	
qstat-2016-10-17-0300	15836	15708	6748	
qstat-2016-10-17-0400	15884	15756	6796	
qstat-2016-10-17-0500	16076	15948	6988	
qstat-2016-10-17-0600	16332	16204	7244	
qstat-2016-10-17-0700	16396	16268	7308	
qstat-2016-10-17-0711	16668	16540	7580	
qstat-2016-10-17-0800	16892		8700	
qstat-2016-10-17-0900	16956		8764	
qstat-2016-10-17-0941	17268			
qstat-2016-10-17-0952	17268			
qstat-2016-10-17-1000	17292			

Figure A1. Selecting backfill jobs and updating job profiles.

References

1. Troger, P.; Rajic, H.; Haas, A.; Domagalski, P. Standardization of an API for Distributed Resource Management Systems. In Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07), Rio De Janeiro, Brazil, 14–17 May 2007; pp. 619–626.
2. Marco, C.; Fabio, C.; Alvise, D.; Antonia, G.; Alessio, G.; Francesco, G.; Luca, P. The glite workload management system. In *Journal of Physics*; Conference Series; IOP Publishing: Bristol, UK, 2010; Volume 219, p. 062039.
3. Abawajy, J.H. An efficient adaptive scheduling policy for high-performance computing. *Future Gener. Comput. Syst.* **2009**, *25*, 364–370. [CrossRef]
4. Sodani, A. Knights landing (knl): 2nd generation intel®xeon phi processor. In Proceedings of the IEEE Hot Chips 27 Symposium (HCS), Cupertino, CA, USA, 22–25 August 2015; pp. 1–24.
5. Östberg, P.O.; Espling, D.; Elmroth, E. Decentralized scalable fairshare scheduling. *Future Gener. Comput. Syst.* **2013**, *29*, 130–143. [CrossRef]
6. Yoon, J.; Hong, T.; Kong, K.; Park, C. Improving the job success rate through analysis of user logs in HPC. *J. Digit. Contents Soc.* **2015**, *16*, 691–697. [CrossRef]
7. National Institute of Supercomputing and Networking, KISTI. Available online: http://www.ksc.re.kr (accessed on 11 February 2020).
8. Das, A.; Mueller, F.; Hargrove, P.; Roman, E.; Baden, S. Doomsday: Predicting which node will fail when on supercomputers. In Proceedings of the SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, Dallas, TX, USA, 11–16 November 2018; pp. 108–121.
9. Pallickara, S.; Pierce, M. Swarm: Scheduling large-scale jobs over the loosely-coupled hpc clusters. In Proceedings of the 2008 IEEE Fourth International Conference on eScience, Indianapolis, IN, USA, 7–12 December 2008; pp. 285–292.
10. Shanley, T. *InfiniBand Network Architecture*; Addison-Wesley Professional: Boston, MA, USA, 2003.
11. Chaubal, C. *Scheduler Policies for Job Prioritization in the Sun N1 Grid Engine 6 System*; Technical Report; Sun Microsystems, Inc.: Santa Clara, CA, USA, 2005.
12. Lipari, D. The SLURM Scheduler Design. In Proceedings of the SLURM User Group Meeting, Barcelona, Spain, 9 Octeber 2012; Volume 52, p. 52.
13. Lumb, I.; Smith, C. Scheduling attributes and Platform LSF. In *Grid Resource Management*; Springer: Boston, MA, USA, 2004; pp. 171–182.
14. Bode, B.; Halstead, D.M.; Kendall, R.; Lei, Z.; Jackson, D. The Portable Batch Scheduler and the Maui Scheduler on Linux Clusters. In Proceedings of the Annual Linux Showcase & Conference, Atlanta, GA, USA, 10–14 October 2000.
15. Kannan, S.; Roberts, M.; Mayes, P.; Brelsford, D.; Skovira, J.F. *Workload Management with Loadleveler*; IBM Redbooks: Armonk, NY, USA, 2001; Volume 2, p. 58.
16. Iqbal, S.; Gupta, R.; Fang, Y.C. Planning considerations for job scheduling in HPC clusters. *Dell Power Solution.* **2005**, 133–136.
17. Kettimuthu, R.; Subramani, V.; Srinivasan, S.; Gopalasamy, T.; Panda, D.K.; Sadayappan, P. Selective preemption strategies for parallel job scheduling. In Proceedings of the International Conference on Parallel Processing, IEEE, Vancouver, BC, Canada, 21 August 2002; pp. 602–610.
18. Sabin, G.; Kochhar, G.; Sadayappan, P. Job fairness in non-preemptive job scheduling. In Proceedings of the International Conference on Parallel Processing, ICPP 2004, Montreal, QC, Canada, 15–18 August 2004; pp. 186–194.
19. Naik, V.K.; Squillante, M.S.; Setia, S.K. Performance analysis of job scheduling policies in parallel supercomputing environments. In Proceedings of the 1993 ACM/IEEE conference on Supercomputing, Portland, OR, USA, 19 December 1993; pp. 824–833.
20. Yoon, J.; Hong, T.; Park, C.; Yu, H. Analysis of Batch Job log to improve the success rate in HPC Environment. In Proceedings of the International Conference on Convergence Technology, Seoul, Korea, 10–12 July 2013; pp. 209–210.
21. Rasley, J.; Karanasos, K.; Kandula, S.; Fonseca, R.; Vojnovic, M.; Rao, S. Efficient queue management for cluster scheduling. In Proceedings of the Eleventh European Conference on Computer Systems, London, UK, 18–21 April 2016; pp. 1–15.

22. Schwiegelshohn, U.; Yahyapour, R. Analysis of first-come-first-serve parallel job scheduling. In Proceedings of the SODA, San Francisco, CA, USA, 25–27 January 1998; Volume 98, pp. 629–638.
23. Hovestadt, M.; Kao, O.; Keller, A.; Streit, A. Scheduling in HPC resource management systems: Queuing vs. planning. In *Workshop on Job Scheduling Strategies for Parallel Processing*; Springer: Berlin, Germany, 2003; pp. 1–20.
24. Yuan, Y.; Wu, Y.; Zheng, W.; Li, K. Guarantee strict fairness and utilize prediction better in parallel job scheduling. *IEEE Trans. Parallel Distrib. Syst.* **2014**, *25*, 971–981. [CrossRef]
25. Etsion, Y.; Tsafrir, D. *A Short Survey of Commercial Cluster Batch Schedulers*; School of Computer Science and Engineering, The Hebrew University of Jerusalem: Jerusalem, Israel, 2005; Volume 44221, pp. 2005–2013.
26. Klusáček, D.; Chlumský, V. Planning and metaheuristic optimization in production job scheduler. In *Job Scheduling Strategies for Parallel Processing*; Springer: Cham, Switzerland, 2015; pp. 198–216.
27. Henderson, R.L. Job scheduling under the portable batch system. In *Workshop on Job Scheduling Strategies for Parallel Processing*; Springer: Berlin, Germany, 1995; pp. 279–294.
28. Feitelson, D.G.; Weil, A.M.A. Utilization and predictability in scheduling the IBM SP2 with backfilling. In Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing, Orlando, FL, USA, 30 March–3 April 1998; pp. 542–546.
29. Srinivasan, S.; Kettimuthu, R.; Subramani, V.; Sadayappan, P. Characterization of backfilling strategies for parallel job scheduling. In Proceedings of the International Conference on Parallel Processing Workshop, Vancouver, BC, Canada, 21 August 2002; pp. 514–519.
30. Lucarelli, G.; Mendonca, F.; Trystram, D.; Wagner, F. Contiguity and locality in backfilling scheduling. In Proceedings of the 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Shenzhen, China, 4–7 May 2015; pp. 586–595.