Hybrid Job Scheduling Mechanism Using a Backfill-based Multi-queue Strategy in Distributed Grid Computing

Kiejin Park † Changhoon Kang †† and Sungsook Kim †††

†Ajou University, Suwon, Gyeonggi-do, South Korea
†† Gangdong College, Icheon, Gyeonggi-do, South Korea
††† Anyang University, Anyang, Gyeonggi-do, South Korea

Summary
In a distributed computing environment, effective job scheduling is a critical challenge. In this paper, a hybrid job scheduling mechanism is proposed that considers both the meta-scheduling scheme for distributing jobs to overall nodes a**n**d the local job scheduling scheme for assigning jobs within a local node at a simultaneous time. Depending on the number of required processors and the expected execution time of jobs, the order of priorities is established. Jobs with high priority are then allocated to a job queue whereas those with low priority are assigned to a backfill queue with remote jobs that are sent from other nodes. Experiments conducted to evaluate the performance of the proposed mechanism show that the utilization of a grid computing system becomes more efficient and waiting times are considerably reduced.

Key words:
Meta-scheduling, Job scheduling, Backfill scheduling, Distributed Grid computing

1. Introduction

As modern technologies advance, high-performance computing power is more desired to solve complex and large-scale computation problems. To meet this need, a significant amount of research has been conducted to exploit grid computing, which binds up heterogeneous computing resources into a single high performance system [1]. Recently, a variety of grid tools have been developed to support the basic grid services such as 1) grid resource discovery service to locate available resources that are connected to grids [2], 2) grid job scheduling service to distribute and schedule jobs for an efficient processing, 3) grid security service for the system protection, and 4) grid accounting service to assess costs incurred by using the computing resources [3]. In addition, the research on the grid middleware like Globus [4] and Legion [5] have made implementations of grid application programs much easier.
In a grid computing environment, effective job scheduling is a critical challenge. The functions of the scheduler in a grid computing environment can basically be classified into two categories that include: 1) meta-scheduling function regarding the distributed processing of the complex problem among nodes and 2) the local job scheduling function within a local node [6]. The meta-scheduling can improve system utilization by dispatching jobs to local nodes in an efficient manner, and the local job scheduling can reduce the slowdown of job processing by scheduling jobs within a node. So far, a majority of researchers have considered these two types of functions separately. Therefore, it is desirable to develop a hybrid job scheduling mechanism that considers both the meta-scheduling scheme and the local job scheduling scheme in order to improve the overall performance of the grid computing system. For example, Lawson et. al.[7] presented a scheduling for both remote jobs that is migrated from another node and local jobs by use of the multi-queue concept in a distributed parallel processing environment.
Although a variety of distributed computing scheduling algorithms have been studied [8], they tend to consider only the problem’s complexity, and they may be unsuitable to be directly used in a geographically distributed grid computing environment. A backfill algorithm [9] that fits for a heterogeneous distributed parallel computing environment was introduced to achieve a faster response time in grid computing. However, there exist trade-offs among performance metrics because the algorithm may yield different job dispatching results depending on the characteristics and priorities of jobs. More specifically, priorities of a job that are determined to increase the system utilization can incur job slowdown, and as a consequence may result in a decrease of system utilization. It is well known that NP-completeness results when the utilization of the grid computing system and job slowdown is simultaneously improved by adjusting the priority of jobs based on job characteristics [10].
Motivated by this, we propose a hybrid job scheduling mechanism that can exploit both the meta-scheduling scheme and the local job scheduling scheme. This paper is organized as follows. In section 2, related work is presented, and the hybrid job scheduling mechanism is then presented in section 3. Section 4 discusses the

Manuscript received September 5, 2012
Manuscript revised September 20, 2012

backfill-based multi-queues scheduling scheme. In section 5, the experiments and the results are provided to illustrate the performance of the proposed mechanism, and the research summary and future directions for possible research are discussed in Section 6.

2. Related Work

The meta-scheduling schemes can be classified into centralized scheduling and de-centralized scheduling. In the centralized scheduling, all grid jobs are submitted to the meta-scheduler that distributes them to nodes based on the status of the nodes in which this status information is managed by the scheduler. Although this approach can be satisfactory, it may not be scalable because the meta-scheduler needs to manage all information and distribute the jobs. On the other hand, the de-centralized scheduling does not require a separate meta-scheduler since all the jobs are submitted directly to the schedulers of the relevant local node. Depending on a scheduling policy, the scheduler of each node then assigns jobs to itself or sends them to another available node. Although high scalability can be realized by this approach, this scheme is considerably more difficult to implement, and synchronization is hard to achieve [11]. Most of the research on grid scheduling is conducted based on the de-centralized scheduling scheme where submitted jobs in a certain node are assigned to other nodes when the jobs cannot be processed in that node [12]-[14].

With regard to the job processing types, there are three types: batch processing, interactive processing, and parallel processing. Since the batch processing and the interactive processing are executed in a single node, submitted jobs are usually allocated to the least-loaded node. If a node is overloaded, the jobs being executed in that node are sent to another node. On the other hand, the parallel processing requires multiple concurrent processors to process jobs. Therefore, heavy delays in communicating among nodes or long job execution times can be incurred if the jobs that are being executed on multiple nodes do not run in a synchronous manner [15].

Variable partitioning scheme is one of the most frequently used scheduling schemes for parallel processing job scheduling. In this scheme, assignments of jobs can be described as rectangular shapes in a graph with the required resource space on the x-axis and the job execution time on the y-axis. As Fig. 1 shows, the occupied area represents the system utilization of a parallel system.

It is worth noting that a job with a low priority cannot be executed even when resources are available since the variable partitioning follows the first-come first-served (FCFS) policy. As shown in Fig. 1, it is obvious that job (j4) that arrives after job (j3) has been completed cannot be executed since the resource is not enough due to the two jobs (j1 and j2) that are being currently processed. Nor can job (j5) be executed even though there exists sufficient available resources to process it. This is because job (j5) has a lower priority than job (j4) according to FCFS policy. This causes fragmentation that decreases the system performance as a result of being unable to use idle resources. Therefore, it can be inferred that the response time for job (j5) can be reduced if the job can be assigned without influencing job (j4).

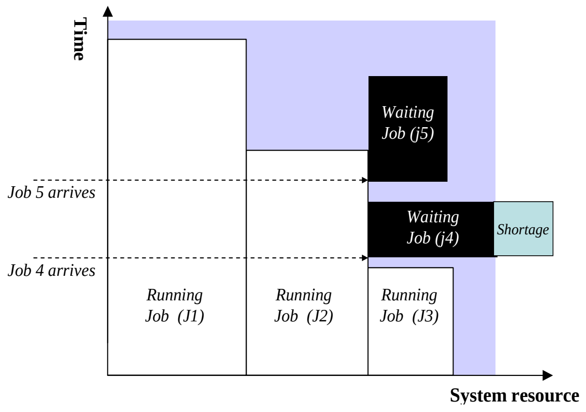

Fig. 1 Variable partitioning scheme.

The backfill scheme has been proposed to address the fragmentation problem [9]. This scheme changes the priority of grid jobs in the job queue in such a way that a job can be moved to the top of the job queue if it can be executed by currently available resources without delaying the processing of higher priority jobs. A number of schemes based on backfill scheduling have been developed to enhance performance by: 1) reducing job slowdown by assigning a high priority to a job with short execution time [16], 2) reducing job slowdown by sorting job orders according to the waiting time since their submission [17], and 3) increasing system utilization by allowing job slowdowns incurred by jobs with short execution time or low priority [18]. In spite of the research, the trade-off between system utilization and job slowdowns remains to be addressed [19]. Although gang scheduling [20] and dynamic partition [21] have also been proposed for the purpose of addressing the fragmentation problem, several limitations such as communication overhead or clock synchronization in a node have restricted their extensive use.

In the backfill scheduling schemes, the conservative backfill schemes allow execution of a low priority job only when it does not cause any slowdowns of *all* the other jobs with higher priority. The EASY backfill schemes execute low priority jobs in a queue when the slowdown of only the first job in the queue is not caused. The EASY backfill can increase the system utilization of the entire system by backfilling more jobs than the conservative backfill. However, it may not be able to guarantee job completion times to users due to possible unbounded delays caused by the job slowdowns.

In this paper, a hybrid job scheduling mechanism is presented to increase system utilization that is not easy to achieve by the conservative backfill and to prevent the unbounded delay incurred by the EASY backfill. The mechanism distributes high-priority jobs to the job queue where reserved jobs are stored. Also, it allocates jobs with low priority or remote jobs that are sent from other nodes to the backfill queues. Several experiments are conducted to evaluate the proposed mechanism with real data, and the results show that the utilization of a grid computing system increases while job slowdown decreases at the same time.

3. Structure of Distributed Grid Computing System

Fig. 2 shows the architecture of the grid computing system that enables the hybrid job scheduling mechanism. In the architecture, grid workers (i.e. computers) constituting a grid computing node are linked as a single logical group through the Internet, and they operate as a single computer system. Each node consists of grid workers, a meta-scheduler, and a local job scheduler. The hybrid job scheduling mechanism proposed in this paper supports the interactions between a meta-scheduler and a local job scheduler to minimize the scheduling load within a local node and to facilitate cooperation among nodes. As such, it can improve the efficiency and the availability of a grid computing system by allowing both the meta-scheduling and the local job scheduling to be simultaneously utilized.

When a user submits a job to a node (0. Initiate Job), the scheduler of the node either allocates the job to the job queue for the submitted job to be processed by a local job scheduling policy, or sends it to one of the backfill queues where the job waits to be assigned to the job queue (1. Request Job). If it is found to be impossible to execute the job at the node due to excessive load, the scheduler requests other nodes for processing it (2. Request Remote Job). If resources are available in other node(s) by broadcasting and replying among nodes (3. Discover & 4. Volunteer), the job is sent to the other node(s) (5. Negotiate & 6. Request Job). At this time, the local scheduler in the receiving node communicates with the meta-scheduler that keeps track of processing requests for remote jobs, and it then allocates the job in the job queue if the meta-scheduling policy is not violated (7. Allocate Job). After the receiving node completes the job in accordance with a local job scheduling strategy, it notifies the user (8. Reply Job).

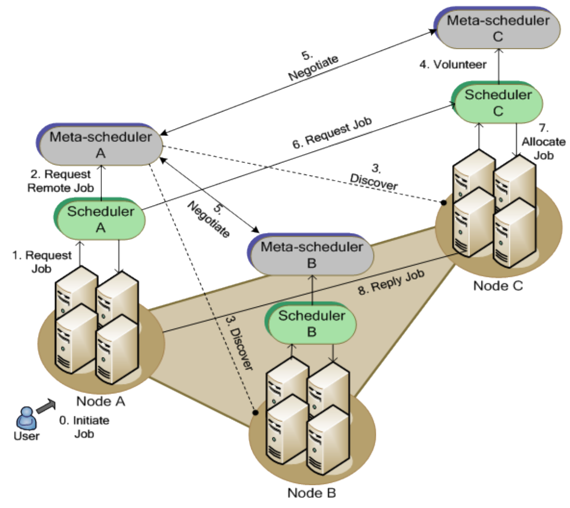

Fig. 2 Architecture of distributed grid computing system.

To put it another way, the proposed architecture for the hybrid job scheduling can incorporate centralized hierarchical scheduling strategies that can be exercised by meta-schedulers and de-centralized scheduling policies implemented by local job schedulers simultaneously. Thus, communication bottlenecks, SPOF (Single Point of Failure) problems, or scalability problems can be addressed with the grid computing architecture.

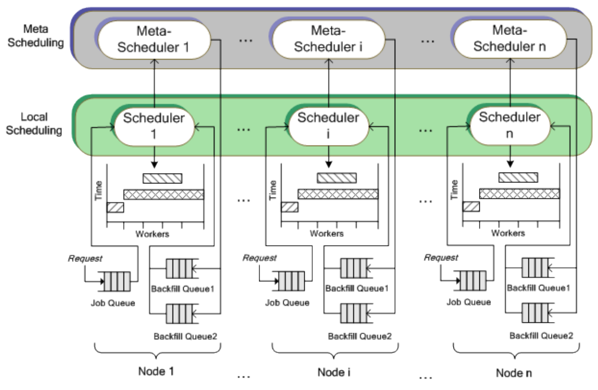

Fig. 3 Multi-queue scheduler of computing nodes.

Fig. 3 shows the structure of the multi-queue scheduler of local nodes in a distributed grid computing system. Each local node consists of workers participating in the grid computing and a scheduler that supports both the meta-scheduling and the local job scheduling by means of the multi-queue backfill mechanism. The scheduler again consists of two types of queues: The first is the job queue that stores local jobs that are executed by FCFS policy, and the second are the two backfill queues where jobs are waiting for backfilling.

It should be noted that the scheduler considers both the states of all the workers and the job characteristics when it employs the scheduling strategy. In other words, a job that is submitted to an arbitrary node is assigned to one of the three queues within the node by the local job scheduler. To which queue a job is assigned is determined based on the number of required processors and the expected execution time of the job. Jobs that require a large number of processors are assigned to the job queue where other jobs are reserved depending on a job priority. On the other hand, local jobs requiring a small number of processors and jobs that are received from other nodes are sent to the backfill queues. If the node is overloaded, it is sent to other node with the least-loaded backfill queues by the meta-scheduler. The job is then stored in one of the backfill queues of the receiving node.

4. Multi-queue Scheduling Mechanism

4.1 Multi-queue Management Strategy

The multi-queue management strategy (MMS) is related to the process that allocates submitted jobs to either the job queue or the backfill queues based on their characteristics. In backfill scheduling, it is important to determine how many jobs are to be backfilled in an efficient way. Jobs that require a small number of processors may have high possibility of being backfilled since it is easier for them to acquire available processors within a node. This approach can increase the likelihood of improving the utilization of the processors. In addition, while jobs in the job queue are executed in FCFS fashion, jobs in the backfill queue can be backfilled whenever a backfill is possible even when a job in the job queue is being executed. Thus, MMS strategy can be applied when 1) a job is submitted to a local node, 2) a job on the local node is completed, or 3) a remote job is dispatched from other nodes.

It is worth remarking that priorities of remote jobs are set to be lower than those of the local jobs. This is applied by allocating them to the backfill queues rather than to the job queue. It can contribute to guaranteeing the completion time of local jobs that are submitted directly to a local node. More specifically, in order to prevent a local job submitted to a certain node from being slowed down by remote jobs, flags are given to jobs of specified job types such as a local job or a remote job.

4.1.1 Job Classification and Priority Assignment Strategies for Local Jobs

For the meta-schedulers and the local job schedulers to apply MMS, they reference basic information that includes job number (i), job arrival time (TA_i), and node number (k) on the grid. In addition, jobs submitted to a certain local node contain the parameters with regard to the number of the required processors (P_i) for execution and the expected execution time (TES_i). Thus, a job can be represented by the following expression (1).

$$JOB_i(TA_i, TES_i, P_i, k) \tag{1}$$

Based on information about jobs, MMS classifies and prioritizes jobs through the following steps. In step 1, jobs are classified into three groups depending on the total number of required processors in the node to which they are directly submitted or remotely dispatched. In step 2, each group is further classified into two subgroups according to the expected execution time. For those jobs that are classified in one of six groups that are formed through these steps, MMS is applied in such a way that 1) jobs requiring a large number of processors are allocated to the job queue for fast completion, and 2) those requiring a small number of processors are assigned to the backfill queues for the higher system utilization through more frequent backfills (refer to Figure 4). In step 3, priorities of jobs are assigned.

```
n.multi.distribute (j)
    if (j.req <= n.processor * 1/3)
        n.multi.enqueue(j, backfill2);
    else if (j.req <= n.processor * 2/3)
        n.multi.enqueue(j, backfill1);
    else n.multi.enqueue(j, jobqueue);

n.multi.migrate(n', j);
    n'.multi.enqueue(j, backfill1);
```

Fig. 4 Multi-queue management strategy pseudo-code.

▌ Step 1 – Job classification by the number of processors

In this step, jobs are classified into three groups based on the number of required processors as expressed by conditions (2) – (4), where TP_j is the number of processors in node *j*. After being classified, jobs are allocated to the queues in a node. Large jobs are stored in the job queue, and medium jobs and small jobs are stored in the first backfill queue and the second backfill queue, respectively.

$$L(Large\ Job): \left\lfloor TP_j \times \tfrac{2}{3} \right\rfloor \le P_i \le TP_j \qquad (2)$$

$$M(Medium\ Job): \left\lfloor TP_j \times \tfrac{1}{3} \right\rfloor \le P_i < \left\lfloor TP_j \times \tfrac{2}{3} \right\rfloor \qquad (3)$$

$$S(Small\ Job): 1 \le P_i < \left\lfloor TP_j \times \tfrac{1}{3} \right\rfloor \qquad (4)$$

■ Step 2 – Job classification by expected execution time

In this step, jobs in each group formed by the step 1 are further divided into two types based on the expected execution time. If the expected execution time of a job is longer than the average execution time of jobs that have been completed in the receiving node up to a current point of time, the job is classified as L-type (Long Job). Otherwise, it is characterized as S-type (Short Job). This classification is expressed in conditions (5)-(6):

$$L(Long\ Job): TES_i \ge TMEAN_j \qquad (5)$$

$$S(Short\ Job): TES_i < TMEAN_j \qquad (6)$$

where TES_i is the expected execution time of the job *i*, and $TMEAN_j$ is the average execution time of jobs that have been executed on the receiving node *j*.

It should be noted that the average execution time in a node varies as the node continues to process multiple jobs. Therefore, the classification criterion changes in a dynamical manner.

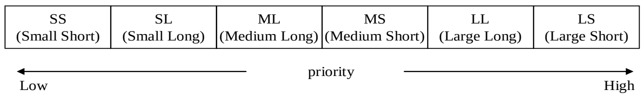

Fig. 5 Priority order of local jobs.

■ Step 3- Priority assignment by job class

A job type that is obtained through the above mentioned job classification steps is used to determine the priority of job processing in the job queue and the sequence of the backfill in the backfill queues. More specifically, S-type jobs have higher priorities in processing or backfilling than L-type jobs. Fig. 5 shows the priorities of local jobs that are determined by the two steps described above. Jobs in the rightmost cells have the highest priority, and those in the leftmost have lowest priority.

4.1.2 Job Classification and Priority Assignment Strategies for Remote Jobs

As aforementioned, remote jobs sent from other nodes are allocated to the backfill queues of a receiving node. The second part in Fig. 4 shows the step that processes the remote jobs. Any remote jobs are assigned to the backfill queue (e.g. Backfill1) of a receiving node by the distribution strategy for remote jobs. This is in order to give the local jobs higher priority over remote jobs so that they can be processed earlier than the remote jobs.

The classification of the remote jobs is done in a similar way as the local jobs. In the first step, remote jobs are classified according to the number of the required processors (P_i) for execution, and they are labeled as either Remote Large (RL) jobs or Remote Small (RS) jobs depending on conditions (7) – (8).

$$RL\ (Remote\ Large\ Job): \left\lfloor TP_j \times \tfrac{1}{2} \right\rfloor \le P_i \le TP_j \qquad (7)$$

$$RS\ (Remote\ Small\ Job): 1 \le P_i < \left\lfloor TP_j \times \tfrac{1}{2} \right\rfloor \qquad (8)$$

In the second step, they are further classified based on the execution time as in conditions (9)-(10). When a job is sent from another node, the receiving node references information concerning job execution time and compares it with the average execution time up to a current point of time to determine the type of the remote job.

$$RL\ (Remote\ Long\ Job): TES_i \ge TMEAN_j \qquad (9)$$

$$RS\ (Remote\ Short\ Job): TES_i < TMEAN_j \qquad (10)$$

The Fig. 6 shows the priority orders of remote jobs that are determined by the two steps mentioned above.

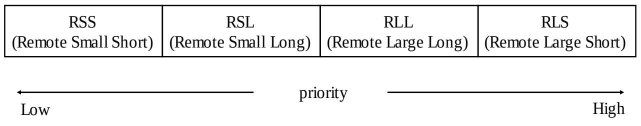

Fig. 6 Priority order of remote jobs.

4.2 Hybrid Job Scheduling Strategy

The meta-scheduling strategy aims at improving the system utilization of the entire grid system and the local job scheduling strategy helps achieve an efficient job distribution within a node. Jobs in the backfill queues can be delayed unboundedly when required processors for execution of a job are not obtained, and the jobs are not reserved in the backfill queue. In particular, the probability of unbounded delay of remote jobs can be larger than that of local jobs. The drawbacks of both the conservative and the EASY backfill mechanisms that were described in section 2 can be alleviated by the mechanism proposed in this paper, which prevents unbounded delays and improves system utilization.

```
n.global.discover (j)
V = volunteer list with idle resources;
for each v ∈ V
v.global.negotiate(n, j);

n.global.negotiate (n′, j)
runnable = n.local.create(j);
if (runnable) n′.global.commit(n, j);
else n′.global.wait(n, j);

n.global.commit (n′, j)
n.multi.migrate(n′, j);

n.global.wait (n′, j)
if (n.multi.j.resv_counterG++ > Threshold){
  fix job j's priority into the urgent level;
  n.global.reserve(j);
}

(a) meta scheduler
```

```
n.local.create (j)
span = some runs during an admitted time;
if (n.avail >= j.req) n.local.execute (j, span);
else {
q = n.multi.distribute(j);
n.local.wait(j);
  }

n.local.remote (j)
n.global.discover(j);

n.local.execute (j, span)
j.met = j.met – span;
if (j.met == ∅) n.multi.depart(j);
else {
span = some runs during an admitted time;
n.local.execute(j, span);
  }

n.local.wait ( j)
if (n.multi.j.rcounterL++ > Threshold){
  fix job j's priority into the urgent level;
  n.local.reserve(j);
  }

n.local.reserve (j)
span = the reserved execution slots for j.req;
  n.local.backfill(j, span);

n.local.backfill (j, span)
j.met = j.met – span;
if (j.met == ∅) n.multi.depart(j);
else n.local.execute();

n.local.volunteer ()
newly = current idle resources;
n.avail = n.avail + newly;

(b) local scheduler
```

Fig. 7 The meta- and the local job scheduling procedure.

Meta-scheduling Strategy:

When a job is submitted to a node, the meta-scheduler determines whether the job is to be processed in the receiving node or to be sent to another node based on the status of receiving node and grid system. In this stage, the meta-scheduler negotiates with meta-schedulers of other nodes for better job allocation. Fig. 7 (a) shows the procedures of the meta- scheduling. When a job *(j)* is submitted to a node *(n)*, the meta-scheduler of node *(n)* searches for volunteer nodes for processing a job *(j)* (*n*. global.discover (*j*)). At this time, the meta-scheduler of a volunteer node (e.g., node *v*) communicates with the local job scheduler to check resource availability for processing the job *(j)*. If a resource is available, node *(v)* responds with a commit message (*n*.global.commit (*v*, *j*)). Then, job *(j)* is migrated from node *(n)* to node *(v)* (*n*.global.migrate (*v*, *j*)).

Local Job scheduling Strategy:

The local job schedulers can manage information about a node status regarding the number of available processors, the system utilization (memory usage, network usage, etc.), system slowdowns, and expected response time. Based on this information, the scheduler makes a decision with regard to job execution and dispatching to other node (refer to Fig. 7 (b)).

To prevent unbounded delay of jobs in the job queue, a certain number of jobs are reserved to be executed so that they are not slowed down due to other jobs in the backfill queue. By doing this, the system utilization can also be increased more than the EASY backfill whereas only the first job is reserved in the EASY backfill mechanism. The number of the reserved jobs (J_r) in the job queue is determined by (11):

$$J_r = \alpha \cdot J_n,\ 0.1 \le \alpha \le 0.5 \qquad (11)$$

where J_n is the number of queued jobs in the job queue, and α is the threshold value used to adjust the number of the reserved jobs.

To prevent unbounded delays of jobs in the backfill queue, a reservation technique is also applied to jobs in the backfill queue. More specifically, when the number of jobs that are not backfilled (J_{NB}) is found to be larger than a threshold value (β) at a certain point of time, those jobs are dispatched to the job queue, or sent to other node if they can be executable in that node. It can be expressed as (12).

$$J_{NB} \ge \beta \qquad (12)$$

When the slowdowns of jobs in the backfill queues occur even if the jobs are reserved by condition 12, those jobs need to be sent to another node. In this case, those jobs become remote jobs in the other node that receives them. To determine which node the jobs are to be sent to, the mechanism proposed in this paper checks whether the jobs can be executed in the receiving node in consideration of execution and transfer time of the jobs. By doing this, the influence of remote jobs in a receiving node can be minimized. The number of reserved jobs plays a key role in the proposed mechanism. It is set to greater than in the EASY backfill and less than in the conservative backfill.

5. Performance Evaluation

In order to evaluate the performance of the hybrid job scheduling mechanism proposed in this paper, several experiments were conducted. In this type of experiment, input data play a critical role in evaluating the performance of different mechanisms because of their influence on the job load. According to the following two steps, the input data are generated. In the first step, parameters of independent variables (e.g., mean arrival time, mean estimated execution time, mean number of processors, mean width, etc.) are generated from the set of workload logs of Feitelson Archive [22]. Table 1 shows the parameters that are generated from CTC trace (CTC: the Cornell Theory Center 512-node IBM SP2, 79,296 jobs) and Feitelson workload. The nodes of the CTC computer are not all identical and they differ in type and memory.

Table 1: Workload data

Month	Total	A Type	B Type
July	7950	7933	7897
Aug.	7273	7279	7234
Sep.	6167	6180	6106
Oct.	7257	7277	7270
Nov.	7917	7841	7816
Dec.	7896	7893	7888
Jan.	7519	7538	7506
Feb.	8189	8177	8159
Mar.	6915	6933	6909
Apr.	6124	6102	6085
May.	6082	5984	5962
Average	7208.09	7194.27	7166.55

A Type: Jobs requiring at most 256 nodes, B Type: Jobs requiring at most 128 nodes

In the second step, with the parameters that are generated in the first step, other parameters for the experiments are then obtained based on the probability functions which are described as follows. The average job arrival, the job request time, and the average number of required processors are assumed to follow an exponential distribution with the rate of 0.167/min. (≈ 7200 jobs/month), an exponential distribution with the average of 100 min, and a uniform distribution from 1 to 64, respectively [9]. Finally, a grid system with 8 nodes of 64 processors each is used.

The backfilling ratios that is the ratio of the number of backfilled jobs to the number of queued jobs with regard to the system utilization are measured, and the impact of the hybrid multi-queue and the remote job processing is assessed in terms of the slowdown ratio of the jobs (*SlowdownRatio*, refer to equation 14). In addition, the slowdowns caused by different scheduling strategies are compared by means of the average response time metrics. The average slowdown time of the job can be obtained using (13): [16]

$$AverageSlowdownTime = \frac{\sum_{i=1}^{N} \frac{resp(i)}{width(i) * \max(S, exec(i))}}{N} \tag{13}$$

where N is the total number of jobs, S is the processing time of the job with the shortest execution time, $resp(i)$ is the time elapsed between job submission and completion, $exec(i)$ is the execution time of the job, and $width(i)$ is the number of the processors required for the job to be executed.

To compare the effectiveness of the multi-queue ($AverageSlowdownTime_m$) and the single queue ($AverageSlowdownTime_1$) strategies, the slowdown ratio is used.

$$SlowdownRatio = \frac{AverageSlowdownTime_1 - AverageSlowdownTime_m}{\min(AverageSlowdownTime_1, AverageSlowdownTime_m)} \tag{14}$$

As can be observed in (14), a positive value of *SlowdownRatio* means that the average slowdown of the single queue strategy is greater than that of the multi-queue strategy. On the other hand, negative value of *SlowdownRatio* indicates that the multi-queue strategy causes greater slowdowns due to the management overheads.

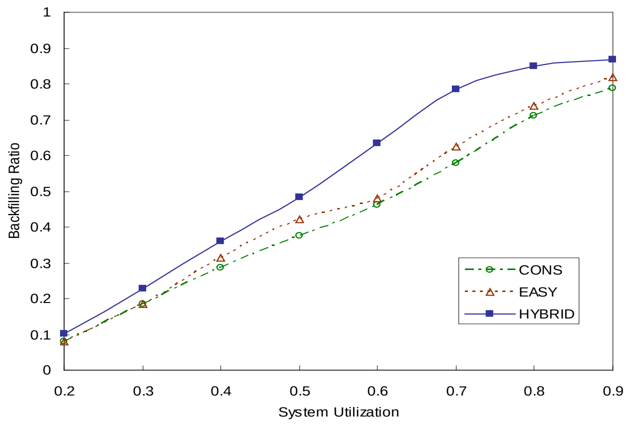

Fig. 8 Backfilling ratios depending on the system utilization.

Fig. 8 shows the changes of the backfilling ratios of three different strategies depending on the system utilization that is calculated by dividing the job arrival time by the average processing time. Therefore, more frequent job requests can increase the system utilization. As can be observed in Fig. 8, the EASY backfill method has more backfills than the conservative backfill method. Furthermore, backfills occur the most frequently when the hybrid multi-queue scheduling mechanism is applied, which means that hybrid multi-queue scheduling mechanism improves the system utilization of the entire grid system.

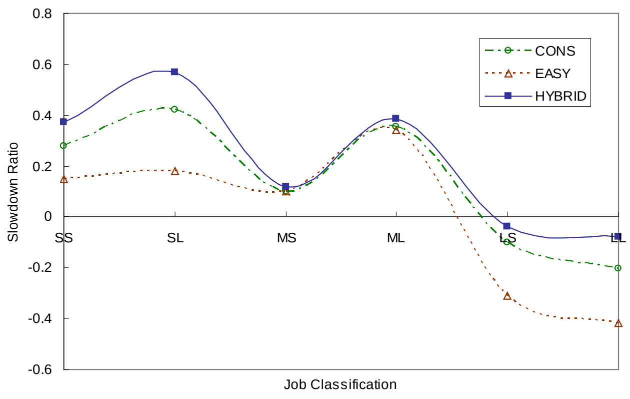

Fig. 9 The changes of the slowdown ratio.

Fig. 9 shows the changes of the slowdown ratio of jobs according to the number of required processors and the expected execution time for which both are used job classification. It can be observed that the hybrid multi-queue is more effective to decrease the job slowdown ratio than the single queue regardless of scheduling strategy. In addition, the slowdowns occur for the jobs (LS and LL) that require a large number of processors due to the backfills of the jobs with low priority. However, the hybrid multi-queue scheduling mechanism causes fewer slowdowns than both the conservative and the EASY backfill methods even if a single queue is used. From this observation, it can be inferred that the hybrid job scheduling method can have advantages that include the unbounded delay prevention and the increased backfill ratio which are the drawbacks of the conservative backfill method and of the EASY backfill methods, respectively.

Fig. 10 shows the changes of the average response time with regard to the system utilization. In case of the EASY backfill method, higher system utilization incurs a steep increase in the average response time due to increased number of backfills. In the case of the hybrid method proposed in this paper, the average response time is greater than the conservative method in spite of smaller job slowdown. This is because the hybrid method dispatches jobs to other nodes more frequently than the conservative method. Therefore, it can be inferred that the overhead of dispatching jobs to remote nodes plays an important role in improving the response time of job executions if a dedicated network is not used to connect nodes of a grid. Note that the increased system utilization does not always mean reduced job response time.

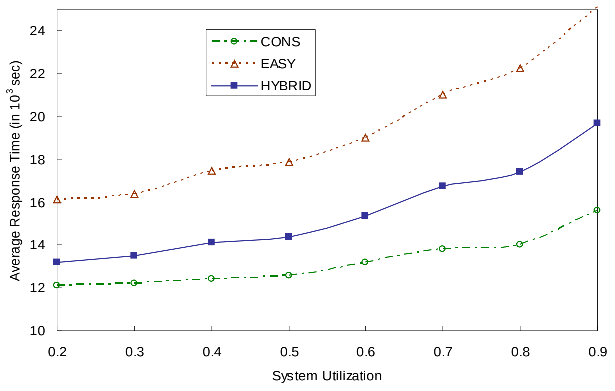

Fig. 10 Average response time according to system utilization.

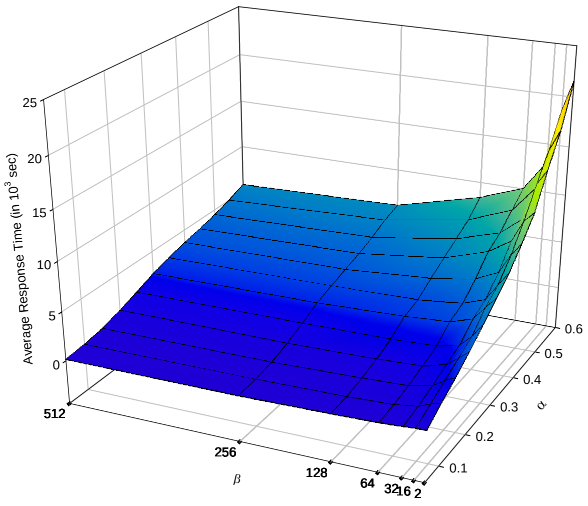

Fig. 11 Average response time with parameters (α , β) used in the reservation policy.

Fig. 11 shows the graph of the average response times according to the value of α (the threshold value that is used to adjust the ratio of the reserved jobs to queued jobs in job queue) and β (the threshold value for those jobs that are backfilled for execution) that are introduced to implement the reservation policy of the conservative and the EASY backfilll methods and to prevent unbounded delay of the EASY backfill method, respectively. As α increases, the average response time also increases. This is because more jobs are moved to the job queue and backfilled, resulting in longer waiting time of jobs that are not backfilled. Also, the decrease of the threshold value of β that restrains the backfill of jobs that can otherwise be backfilled causes the increase in the average response time. However, the impact of the value of β on the average response time is observed to decrease for somewhat large values of β (e.g., $\beta \geq 64$). This is more obvious in case of small values of α. In particular, because there can be more available processors if each node has more processors, the probability of the slowdowns caused by the backfills decreases.

6. Conclusion

Although a significant amount of research has been conducted on the developments of scheduling mechanisms to reduce job slowdown, improve the response time and the system utilization of a grid system, most of the research has been pursued in a separate manner to solve problems of the meta-scheduling methods that distribute jobs to each node and the local job scheduling methods within a node. In order to address this situation, we have proposed the hybrid job scheduling mechanism that considers both the meta-scheduling and the local job scheduling approaches at the same time.

Jobs that are submitted to grid computing nodes are classified based on the number of required processors and the expected job execution time. This classification schema is then used to determine whether jobs are processed in a local node or dispatched to other node(s) according to the job load. As the backfilling ratio and job slowdowns are improved with the hybrid multi-queue scheduling mechanism, the utilization of a grid computing system becomes more efficient and waiting times are reduced. Future work will include the investigation of the impact of the network communication overhead on overall performance of a grid system and will be checking for the implementation issues with various input data sets.

Acknowledgments

This work (Grants No. C0023120) was supported by Business for Cooperative R&D between Industry, Academy, and Research Institute funded Korea Small and Medium Business Administration in 2012 .

References

[1] I. Foster, et. al., “Grid Services for Distributed System Integration," Computer, Vol. 35, No. 6, pp. 37-46, 2002.

[2] Kiejin Park, et. al., “Credible Worker Selection Mechanism for Grid Computing,” IEICE Transactions on Information and Systems, Vol. E89-D, No.2, pp. 605-611, Feb. 2006.

[3] K. Krauter, et. al., “A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing,” Software Practice and Experience Journal, Vol. 32, No. 2, pp. 135-164, Feb. 2002. Summary: Grid computing is concerned with the sharing and coordinated use of diverse resources in distributed "virtual organizations." The dynamic and multiinstitutional nature of these environments introduces challenging security issues that demand new techn.....

[4] I. Foster, et. al., “Globus: A Metacomputing Infrastructure Toolkit,” The International Journal of Supercomputer Applications and Performance Computing, Vol. 11, No. 2, pp. 115-128, Oct. 1997

[5] A. Grimshaw, et. al.,”Legion: Lessons Learned Building a Grid Operating System ,” Proceedings of the IEEE, Vol. 93, Issue 3, pp. 589 – 603, Mar. 2005.

[6] H. Shan, et. al., “Job Superscheduler Architecture and Performance in Computational Grid Environments,” In SC 2003 Conference, 2003.

[7] B. Lawson, et. al., "Multiple-queue Backfilling Scheduling with Priorities and Reservations for Parallel Systems," The 8th International Workshop, JSSPP 2002 Edinburgh, Scotland, UK, pp. 72-87, July 2002.

[8] T. Braun, et. al., “A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems,” Journal of Parallel and Distributed Computing, Vol. 61, pp. 810-837, 2001.

[9] A. Mualem, et. al., “Utilization, Predictability, Workloads and User Run time Estimates in Scheduling the IBM SP2 with Backfilling,” IEEE Trans. Parallel and Distributed System, Vol. 12, No. 6, pp. 529-543, June 2001.

[10] O. Ibarra, et. al., “Heuristic Algorithm for Scheduling Independent Tasks on Nonidentical Processors,” Journal of ACM, Vol. 24, No. 2, pp. 280-289, Apr. 1977.

[11] V. Hamscher, et. al., "Evaluation of Job-Scheduling Strategies for Grid Computing," The 1st IEEE/ACM International Workshop on Grid Computing (Grid 2000) at the 7th International Conference on High Performance Computing (HiPC-2000), LNCS 1971, pp. 191-202, 2000.

[12] V. Subramani, et. al., “Distributed Job Scheduling on Computational Grids Using Multiple Simultaneous Requests,” The 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11 2002), pp. 359-368, July 2002.

[13] Q. Wang, et. al., “De-centralized Job Scheduling on Computational Grids Using Distributed Backfilling,” Grid and Cooperative Computing - GCC 2004: Third International Conference, LNCS 3251, pp. 285-292, Oct. 2004.

[14] K. Li, “Job Scheduling for Grid Computing on Metacomputers,” The 19th International Parallel and Distributed Processing Symposium (IPDPS 2005), Abstracts Proceedings, Apr. 2005.

[15] D. Feitelson, et. al., “Theory and Practice in Parallel Job Scheduling,” Job Scheduling Strategies for Parallel Processing, IPPS'97 Workshop, Geneva, Switzerland, LNCS 1291, pp. 1-34, Apr. 5, 1997.

[16] D. Zotkin, et al., “Job-Length Estimation and Performance in Backfilling Schedulers,” The 8th IEEE International

Symposium on High Performance Distributed Computing (HPDC'99), Aug. 1999.

[17] S. Srinivasan, et al., “Characterization of Backfilling Strategies for Parallel Jobs Scheduling,” 31st International Conference on Parallel Processing Workshops (ICPP 2002 Workshops), pp. 514-522, Aug. 2002.

[18] W. A. Ward Jr., et al., “Scheduling Jobs on Parallel Systems Using a Relaxed Backfill Strategy,” , 8th International Workshop on Job Scheduling Strategies for Parallel Processing(JSSPP’2002), pp. 88-102, July 2002.

[19] B. G. Lawson, et al., “Self-Adapting Backfilling Scheduling for Parallel Systems,” 31st International Conference on Parallel Processing (ICPP 2002), pp. 583-592, Aug. 2002.

[20] D. Feitelson, et al., “Improved Utilization and Responsiveness with Gang Scheduling,” Job Scheduling Strategies for Parallel Processing, IPPS'97 Workshop, Geneva, Switzerland, LNCS 1291, pp. 238-261, Apr. 1997.

[21] R. McCann, et. al., “A Dynamic Processor Allocation Policy for Multiprogrammed Sharedmemory Multiprocessors,” ACM Trans. on Computer System, Vol. 11, No. 2, pp. 146-178, May 1993.

[22] D. Feitelson, “Logs of Real Parallel Workloads from Production Systems,” Available: http://www.cs.huji.ac.il/labs/parallel/workload/logs.html.

Kiejin Park received the B.S. degree in industrial engineering from Hanyang University, Seoul, Korea, in 1989, the M.S. degree in industrial engineering from Pohang University of Science and Technology, Pohang, Korea, in 1991, and the Ph.D. degree from the Department of Computer Engineering, Graduate School, Ajou University, Suwon, Korea, in 2001. From 1991 to 1997, he was with the Software Research and Development Center, Samsung Electronics Company Ltd., Suwon, as a Senior Researcher. From 2001 to 2002, he was with the Network Equipment Test Center, Electronics and Telecommunications Research Institute, Daejeon, Korea, as a Senior Researcher. From 2002 to 2004, he was with the Department of Computer Engineering, Anyang University, Anyang, Korea, as a Professor. Since 2004, he has been an Associate Professor with the Division of Industrial and Information Systems Engineering, Ajou University. From 2010 to 2011, he was with Rutgers, The State University of New Jersey, Piscataway, as a Visiting Professor. His research interests include in-vehicle network, fault-tolerant computing, and cloud computing.

Changhoon Kang was born in Daejeon, Korea. He received the B.S. and M.S. degrees in computer science from Chungnam National University, Korea, in 1986 and 1988, respectively, and Ph. D. degree in Department of Computer Engineering, Graduate School of Ajou University in Korea in 2006. He is currently an Associate Professor in Department of Visual Broadcasting Media, Kangdong College. From 1990 to 1993, he worked in the Computer Center of Chungnam National University, Korea, as an Assistant Teacher. His research interests include cluster computing, and grid computing.

Sung sook Kim received the B.S degree in Educational technology from Hanyang University, Seoul, Korea, in 1991, the M.S. degree in E-Learning from Ajou University, Suwon, Korea, in 2009, respectively, where she is currently working toward the Ph.D. degree in the Department of Computer Science & Engineering in Anyang University. Her research interests include real-time data mining from mobile log data and task scheduling for MapReduce software framework in cloud computing.