마켓 기반 계산 그리드를 위한 SLA 인지형 스케줄링 기법

한영주**, Ren Ye*, 윤찬현*
*한국과학기술원 정보통신공학과
**한국과학기술원 전기및전자공학과
e-mail : y.han@kaist.ac.kr

SLA-Aware Scheduling Scheme for Market-based Computational Grid

Youngjoo Han**, Ren Ye*, Chan-Hyun Youn*
**Dept. of Info. and Comm. Engineering, Korea Advanced Institute of Science and Technology
*Dept. of Electrical Engineering, Korea Advanced Institute of Science and Technology

Abstract

For successfully commercialized grid systems, it is required to provide an efficient scheduling scheme which is able to optimize benefits for three participants such as consumers, brokers, and providers so that every participant has sufficient benefit to maintain a sustainable market. In this paper, we define this job scheduling problem as an objective optimization problem for three participants. The three objectives are to maximize the success rate of job execution, total achieved profit, and the system utilization. To address the scheduling problem, we propose heuristics referred to as SLA-aware scheduling scheme (SA) for optimal resource allocation. The simulation results show that the improvement and the effectiveness of the proposed scheme and the proposed scheme can outperform well-known scheduling schemes such as first come first serve (FCFS), shortest job first (SJF), and earliest deadline first (EDF).

1. 서론

Commercial vendors are currently developing aggressively towards releasing service market where users only pay for what they use. In this market, commercial users will not use a grid service for critical computing jobs, if the service is being provided on the best-effort approach only [1]. Also, providers and brokers will not stay and provide their services, if they could not achieve larger utilization and bigger profit, respectively. For successfully commercialized grid systems, it is needed to provide an efficient scheduling scheme which is able to optimize benefits for all the participants such as consumers, brokers, and providers.

Many studies in the literature have addressed scheduling issues in the grid. Most traditional scheduling schemes decide when to execute which job on which resource by the grid system and adopt system-centric approaches that maximize execution performance and optimize management cost [2]. However, these approaches have been found by many researchers to be lacking in supporting service-on-demand computing [3]. These techniques fail to capture the more delicate requirements of users such as QoS constraints. Accordingly, researchers have been examining the appropriateness of user-centric approaches such as market-based techniques [4] for satisfying QoS constraints and ensuring that users are treated fairly. In a market-based grid, scheduling decisions are made by three parties (resource provider, broker, and consumer), but all the individual participants utilize some market instruments such as price to achieve their objectives. In particular, every participant will not stay and perform their own role, if sufficient benefit for them is being provided on the best-effort approach only. So, there is a scheduling problem that is the issue of optimizing benefits for three parties so that every participant has sufficient benefit to maintain a sustainable market.

The motivation for our work stems from the challenges in scheduling jobs of consumers to resource providers to optimize benefits for three parties. The practical constraints as the benefit for each party include maximizing (i) the successful execution rate of jobs for consumers, (ii) service profits for brokers, and (iii) the system utilization for providers to increase their profit. The objective of this study is to schedule and process the job with achieving the practical constraints mentioned above. However, our objectives could not be realized by an almighty scheduler since it is required more time to compute for searching appropriate providers when increasing the number of providers [5]. Thus, each provider makes scheduling decisions on his own behalf, and the individual economic behaviors of all providers work together to accomplish resource scheduling.

In this paper, we formulate the scheduling problem outlined above and design an efficient scheduling scheme to solve the problem. Since optimal scheduling algorithm for dynamic systems with more than one processor, and/or tasks that have mutual exclusion constraints is a NP-hard problem [6], we propose the SLA-aware scheduling scheme (SA) using heuristics to find appropriate resources based on SLA. Through the simulation, we demonstrate the improvement and the effectiveness of the proposed SA scheme and show that the SA scheme outperforms well-known schemes such as FCFS, SJF, and EDF in terms of optimizing benefits.

2. SLA-Aware Scheduling Scheme

An essential benefit for consumers is that the grid system

should have a high successful execution rate of jobs, where it means that a job is completed without SLA violation. Thus, we choose the successful execution rate (σ) of the grid system as the benefit for consumers. A primary benefit for brokers is a profit that is calculated by subtracting penalty cost from the reward for the broker. Thus, we choose total achieved profit (ζ) based on the profit policy discussed in section 2-B as the benefit for brokers. In addition, each provider wants to fully utilize all the available resource to accept more jobs, and finally it can increase profits from the viewpoint of the entire grid. In order to capture this intuition, we choose a metric called utilization (δ) of the provider. Therefore, the scheduling problem can be described as follows: Find an efficient scheduling scheme that schedules a set of jobs $J=\{J_0, J_1, ..., J_n\}$ to a set of providers $R=\{R_0, R_1, ..., J_m\}$ to maximize σ, ζ, and δ.

Now we propose the SLA-aware scheduling scheme (SA) with heuristics to solve the scheduling problem define above.

A. Eligibility Check Algorithm

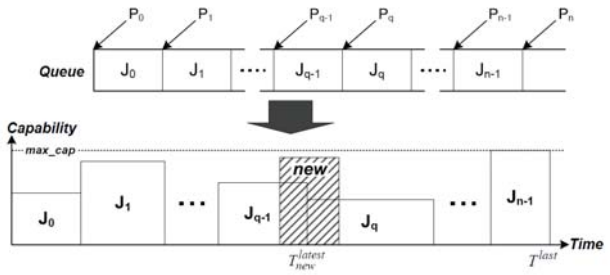


Fig. 1 Waiting jobs and job queue in a resource provider

Every time a resource provider receives a job announcement, it starts the eligibility check algorithm for job scheduling. As shown in Fig. 1, job scheduling can be considered in terms of a 2D chart with time along one axis and the computing capability along the other axis. In addition, each job is represented as a rectangle whose width is the required time slot and height is the required computing capability to complete consumer job within deadline. Thus, when the provider receives the job announcement, the eligibility check algorithm estimates whether it is able to provide the space for the rectangle requested. There are *n* jobs in the job queue in Fig. 1. If we call the potential new job as new, P_0, P_1, …, P_n and P_n represent *n+1* possible places for new to be inserted into. T^{last}, the last time of execution for current jobs in the job queue, is the time instance when all the jobs in the job queue are completed. T_{new}^{latest} is the latest time, which is calculated by subtracting the time duration to execute new with minimum required capability from T_{new}^{target}, to begin the execution of *new* if the provider cannot miss its deadline.

Algorithm 1 shows how the algorithm decides whether to accept *new* based on T_{new}^{target}, req_cap_{new}, and current job queue. The algorithm is based on the principle that inserting a new job can never incur new deadline missing. The time complexity of this algorithm is O(n^2), where n is the number of jobs in the job queue.

Algorithm 1 Eligibility check algorithm

```
1: if T^latest ≥ T^last then
2:   can_satisfy ← true
3:   reordered ← false
4:   avail_place ← P_n and calculate usage price
5: else
6:   for j = 0 to j < n do
7:     if min_cap_new ≤ avail_cap during T_new^run then
8:       Calculate usage price and add P_k into avail_place
9:     else if T_j^start ≤ T^latest and min_cap_new ≤
       max_cap_j then
10:      for k = j to k < n do
11:        if T^end of all jobs in job queue will not miss its
           T^target then
12:          Calculate usage price and add P_k into
             avail_place
13:        end if
14:      end for
15:    end if
16:  end for
17:  if total number of available places in avail_place > 0
     then
18:    can_satisfy ← true
19:    reordered ← true
20:    Choose the cheapest time slot from avail_place and
       reserve it
21:  else
22:    can_satisfy ← false;
23:  end if
24: end if
25: if can_satisfy = true then
26:   Send a bid with the usage price and the capabilities of
      R_j
27: end if
```

(a) Eligibility check algorithm in a resource

Algorithm 2 Price adjustment algorithm

```
1: if P_j^unit = NULL then
2:   Set P_j^unit as the P^M {When join a group}
3: else
4:   F_job ← J_length_j / J_length_total
5:   F_cap ← C_j / Σ_{0≤k<m} C_k
6:   if F_job > F_cap and P_j^unit ≤ P^M then
7:     P_j^unit ← α · P_j^unit
8:   else if F^job < F^cap and P_j^unit ≥ P^M then
9:     P_j^unit ← β · P_j^unit
10:  else
11:    Set current P_j^unit as the unit price
12:  end if
13: end if
14: P_j^usage(T_{i,j}^run) ← P_j^unit · J_length_i · x(J_i)
```

(b) Price adjustment algorithm

Fig. 2 Algorithms for SLA-Aware Scheduling Scheme

B. Price Adjustment Algorithm

The price adjustment scheme is represented in Algorithm 2. Providers use this algorithm to ensure that more profit is achieved, instead of less profit due to failure to meet their deadlines by accepting too many jobs or imbalanced job distribution. When receiving a job, providers start this algorithm, and its time complexity is O(1). C_j and

$\sum_{0\le k<m} C_k$ are the capability of R_j and the aggregated capability of all the providers, respectively. Also, α is a decimal above 1, and β is a positive decimal under 1. This price adjustment algorithm makes the price fluctuate around the market price and differentiates the chances of providers to be chosen. This algorithm ultimately realizes load balancing on the job distribution, meaning that it can increase system utilization. Once P_j^{unit} is set, we can calculate the usage price of resource provider R_j which is defined as follows:

$$P_{i,j}^{usage} = P_j^{unit} \cdot J\_length \cdot x(J_i), \quad (1)$$

Where P_j^{unit} is given by

$$x(J_i) = \begin{cases} \psi & \text{if reordered} = \text{true} \\ d & \text{if reordered} = \text{false}, \end{cases} \quad (2)$$

where ψ denoted a decimal slightly greater than 1. As discussed in 2-A, because the deadlines of some jobs are somewhat tight, these jobs need to be inserted into the job queue ahead of foregoing jobs and should be given higher priority. Thus, it is reasonable to charge more for them. After checking the eligibility specification of J_i, if *reordered* is set to true, providers raise the usage price ψ times more to reduce the chance of being chosen to some extent.

Now we define the profit policies for the brokers and the consumers. When a job submitted is completed with satisfying SLA_l, consumers have to pay reward for the service provided. In particular, when the broker completes job execution earlier than its deadline as the consumer set on the SLA_1, it can receive an incentive (additional reward). Owing to incorrect scheduling decision of the broker or system failure of the providers, however, the grid system fails to meet the deadline of the job. Hence, the penalty cost is given to the broker to compensate the consumer for failure to satisfy SLA_l of the job. Accordingly, when θ_i is deadline of job *i*, the profit policy of the broker *l* is defined as follows:

$$profit_l(J_i) = \begin{cases} P_l^{usage} + C_l^{inc} \cdot (\theta_i - T_i^{\text{end}}) & \text{if } T_i^{end} \le \theta_i \\ P_l^{usage} - C_l^{pen} \cdot (T_i^{\text{end}} - \theta_i) & \text{if } T_i^{end} > \theta_i \end{cases} \quad (3)$$

where P_l^{usage} as a reward is the usage price to use brokering service of the broker *l* to execute job *i*. Also, C_l^{inc} and C_l^{pen} are an incentive cost factor for early completion and a penalty cost factor for violation of SLA_1 in broker *l*, respectively.

C. Divisible Job Scheduling

Though the broker tries to find suitable providers to execute consumer job, when finish time of all the providers exceeded the deadline of a job, it could not guarantee consumer requirements. Especially for divisible jobs, dividing a job into sub-jobs can be a solution to reduce finish time of the job. We discussed this divisible job scheduling scheme in [5]. Thus, in this paper, we do not discuss it in more detail.

Table 1 Simulation parameters and their range

Parameter	Range of Values
Number of Grid Resource	50, 100, and 150
System Load	0.1 - 1.5
Provider Performance	5 - 15
Job Size (Type 1 / Type 2)	(100 - 150) / (200 - 250)
Deadline (Type 1 / Type 2)	(30 - 45) / (60 - 75) units
Transmitted Data Size	50 - 200 [Mbit]
Budget (Type 1 / Type 2)	(5000 - 7500) / (10000 - 12500)
Market Price P_M	10
Control Factor μ	0.5, and 1.0
α / β / ψ	1.2 / 0.8 / 1.1
incentive cost factor C^{inc}	2.5
penalty cost factor C^{pen}	10

3. Performance Evaluation

All simulation runs sufficiently long: 50,000 units in simulation time. All the simulation parameters and their respective ranges are given in Table I. These are varied randomly following uniform distributions while the job arrival rates follow Poisson distribution. The system load varies from 0.1 to 1.5, with a step of 0.1. We consider the following performance metrics, which are of direct relevance to this study. The *successful execution rate* (σ) is defined as the number of jobs completed without SLA violation out of total number of jobs submitted. We also define the total achieved profit (ζ) based on the profit policy and a metric (δ) that quantifies the system utilization of the provider. Furthermore, an important factor we need to consider when selecting a provider is load balancing. Thus, to quantify load balancing efficiency among providers, when J_length_j denotes the number of successfully executed jobs in R_j, we define *balance deviation* (η):

$$\eta \triangleq std\_dev(\Delta_0, ..., \Delta_{m-1}),$$
$$\text{where } \Delta_j = \frac{J\_length_j / \sum_{0\le l<m} J\_length_l}{C_j / \sum_{0\le l<m} C_l}$$

Fig. 3 and 4 show that the behavior of $\sigma, \zeta, \delta,$ and η when we employ the SLA-aware scheduling scheme in a system with 100 resources with respect to the system load. For study the effectiveness of the price adjustment algorithm and the divisible job scheduling algorithm. We work out the simulation results of the SLA- aware scheduling scheme without the price adjustment and divisible job scheduling separately, and we compare them with result of the complete SLA-aware scheduling scheme. From the plots of σ, δ, and η in Fig. 3, it is obvious that two algorithms are complemented each other and indispensable. When disabling the divisible job scheduling algorithm and the system load is high, the SA achieves poor performance. It can be explained that the providers can no longer continue to accept the newly arrived jobs unless the deadline requirements of already accepted jobs together with the new jobs being considered could be satisfied because of heavy system load. A higher deadline missing rate as well as higher job rejection rate results in more penalty and worse load balancing, and finally lower successful execution rate.

For showing the effectiveness of the SLA-aware scheduling scheme, Fig. 4 shows four performance metrics with the SA and three representative scheduling schemes that are deployed in decentralized scheduling frameworks. Three of representative scheduling schemes we choose are FCFS, SJF, and EDF. In Fig. 4(a), for the heavy system load, although all the lines representing four schemes starts falling

steeply, values of σ in the case of the SA is highest in all cases. In Fig. 4(b), it is observed that for the low system load the overall achieved profit (ζ) delivered to the broker is increasing since all the schemes are affordable to accept and process all the jobs arrived. However, as the system load further increases, the achieved profit tends to stop growing and to saturate at a certain level. The profit in case of the SA is highest. From these situations we infer that the proposed SA is seen to be more efficient than three representative scheduling schemes. From the plots of η in Fig. 4(c), we see that when applying the SLA-aware scheduling algorithm, the variation is smaller and better in terms of stability. In other schemes, though some resources complete the tasks earlier, and other resources take a longer time for processing their load. This means that the proposed SA has an effect on load balancing, and it gives us an approximate way to improve system performance.

4. Performance Evaluation

We formulated and discussed the job scheduling in a sustainable market-based computational grid as a three-objective optimization problem to optimize benefits for three parties: consumers, brokers, and resource providers. As the problem formulated is a NP-hard, we proposed the SLA-aware scheduling scheme (SA) with heuristics to address the problem. The simulation results showed the effectiveness of the proposed SA with the SLA negotiation strategy on the various environments. Furthermore, we showed that the SA can outperform three well-known schemes such as FCFS, SJF, and EDF and achieved the three objectives better in terms of optimizing benefits for three participants.

본 연구는 2010 년도 정부(교육과학기술부)의 재원으로 한국연구재단-미래기반기술개발사업(첨단융복합분야)의 지원을 받아 수행된 연구임[N01100428].

Reference

[1] Future for European Grids: GRIDs and Service Oriented Knowldege Utilities, Next Generation GRIDs Expert Group Report 3

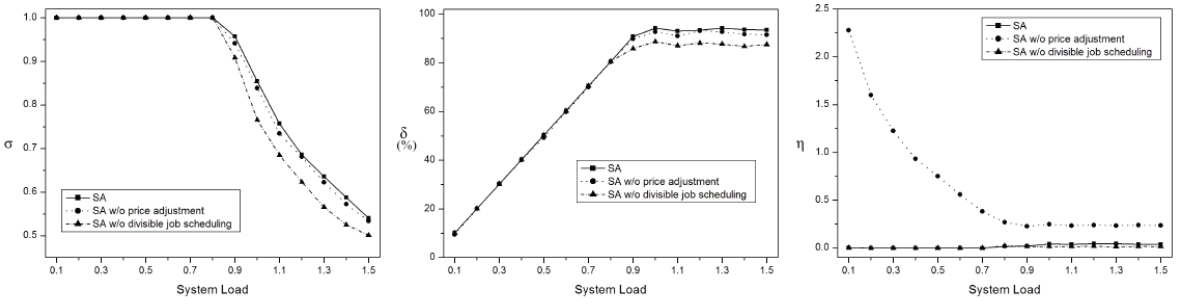
[2] G. Borges et al., Sun Grid Engine, a new scheduler for EGEE middleware, in: Proc. Iberian Grid Infrastructure Conf., May 2007.

[3] R. Wolski et al., Analyzing market-based resource allocation strategies for the computational Grid, in: Proc. Int'l Parallel and Distributed Processing Symposium, Aug. 2001.

[4] R. Buyya, D. Abramson, and S. Venugopal, The Grid Economy, Proceedings of the IEEE 93(3) (2005) 698-714.

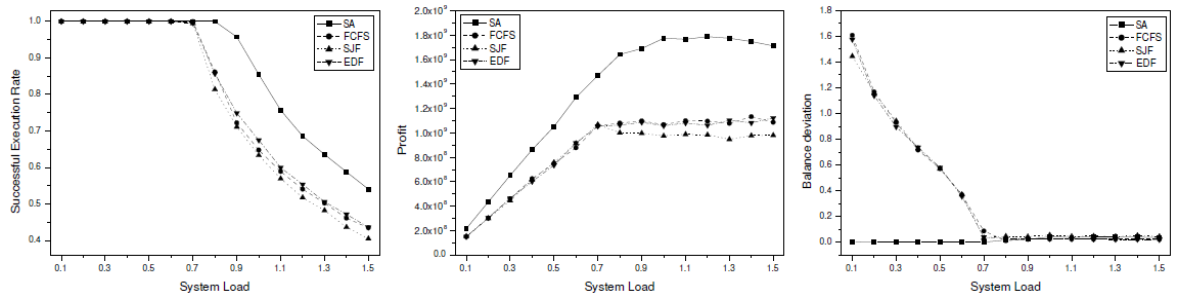
[5] Youngjoo Han and Chan-Hyun Youn, "A new grid resource management mechanism with resource-aware policy administrator for SLA-constrained applications," Future Generation Computer Systems 25(7) (2009) 768-778.

[6] K. Ramamritham, J.A. Stankovic, P.-F. Shiah, Efficient scheduling algorithms for real-time multiprocessor systems, IEEE Trans. on Parallel and Distributed Systems 1(2) (1990) 184-194.



(a) Successful execution rate (σ) (b) Utilization (δ) (c) Balance deviation (η)

Fig. 3 Analysis of SA on the performance objectives



(a) Successful execution rate (σ) (b) Profit (ζ) (c) Balance deviation (η)

Fig. 4 Simulation results for comparison of the SLA-aware scheduling scheme with other algorithms

Acknowledgement