Using Slack Reservation Strategy to Improve User QoS in Computational Grids

Peng Xiao* and Ning Han

College of Computer and Information Science, Hunan Institute of Engineering
**xpeng4623@gmail.com; linghan81@126.com*

Abstract

Advance reservation is an effective technique to realize end-to-end quality of service (QoS) guarantees for grid applications. However, excessive advance reservation will result in low resource utilization and high rejection rate on receiving new requests, which in turn reduces user's QoS satisfaction. In this paper, a slack reservation strategy is proposed, with aiming to mitigate the negative effects brought about by conventional reservation service. The strategy is based on the observation that applications tend to overestimate reservation deadline to ensure their completion. It allows accepting new reservation requests that overlapping with existing ones under certain conditions. Both the benefits and risks of the proposed policy are presented theoretically. Experimental results show that the strategy can bring about remarkably higher resource utilization and lower rejection rate when the system is in presence of heavy reservation workload.

Keywords: *Grid Computing; Quality of Service; Reservation Violation; Task Scheduling*

1. Introduction

Grid computing has emerged as a promising platform for parallel and distributed high-end applications, which often requires end-to-end QoS provision [1]. Advance reservation, as an effective technique to support end-to-end QoS, has been incorporated into many grid platforms, such as Globus [2], Legion [3] and etc. It allows applications to gain concurrent access to adequate resources, and guarantees the availability of resources at required times.

Although advance reservation has been proven effective in many situations, it also brings many negative effects on resources sharing and task scheduling in grid computing. Studies in [4-10] have shown that fixed-capability reservation will result in low resource utilization, and excessive reservation can lead to high rejection rate. These negative effects inevitably have influences on the performance of utility-based grid systems [11-12], where providers wish to fully utilize their resources to obtain maximal profits. More importantly, the negative effects will become very significant when the target grid system is in presence of heavy reservation workload [7-9]. Therefore, how to mitigate the negative effects brought about by conventional reservation service becomes an important issue needed to be addressed.

Typically, advance reservation is defined as a process of requesting resources for use at a specific time in the future [13]. Two key attributes of an advance reservation are start time and deadline. For many grid applications, precise estimation of these two parameters is difficult, due to the availability and the performance of resources are unpredictable in large-scale grid systems. As a result, grid applications tend to overestimate these parameters, especially the deadline, to ensure their successful completion [4]. This behavior results in high rejection rate and low resource utilization. Motivated by these facts, in this paper, we propose a slack reservation strategy, namely SRS, which allows reservations overlapping each other under certain conditions. The objective of SRS is to increase resource utilization

and reduce rejection rate when using advance reservation in grid computing environments. In this way, the user's QoS satisfaction will be improved consequently when the target system is overloaded with plenty of reservation requests.

The rest of this paper is organized as follows. Section 2 presents the related work. In Section 3, we introduce the slack reservation policy and model. In Section 4, theoretical analysis of the policy is presented. In Section 5, extensive experiments are conducted to verify the performance of the proposed strategy. Finally, Section 6 concludes the paper with a brief discussion of future work.

2. Related Work

The effects of advance reservation on system performance have been widely studied in many works. In [5], Smith and Foster investigate the impacts of advance reservation on performance of Grid scheduler in terms of Mean Waiting Time (MWT), Mean Offset Time (MOT), and Request Rejection Rate (RRR). Their experimental results show that MOT will be reduced when the reservation rate is low, however, in face of high reservation rate all the three metrics will increase. These conclusions are repeatedly confirmed in [7-8, 10]. In [9], Wu and Sun study the effects of advance reservation on remote jobs and local jobs in non-dedicated environments. They use queuing system to model resources, and prove that excessive reservation will prolong the response time of remote jobs as well as local jobs.

Many techniques have also been proposed to overcome the limitations of advance reservation. Foster, et al., [4] propose that incorporating adaptive mechanism into advance reservation to provide more flexible reservation policy. An extended GARA [14] architecture is developed, in which reservation mechanism are enhanced with intelligent decision model and performance sensors. In this way, the system can deliver more reliable QoS for Grid applications. However, this adaptive reservation architecture is designed for high-end applications with high-bandwidth and dynamic flows, which means it only adapts to bandwidth reservation for bulk-data transfer. Furthermore, it relies on performance sensors to predict resource performance fluctuations. For many existing systems, incorporating such mechanism is difficult to realize in practice.

To increase resource utilization for reservation-based scheduling, backfilling technique [15] is widely used to improve the scheduling performance. In [6], backfilling-based gang scheduling is incorporated into SCOJO scheduler [16] for co-reservation in multi-cluster grid environment. The simulation results show that it can mitigate the negative effects of advance reservation for the applications with high ratio of computing to communication. In [17], Sulistio et al. use several techniques, including re-arranging subtask, interweaving task graphs and backfilling, into advance reservation based scheduling to improve resource utilization. Although backfilling is effective to improve resource utilization and throughput, it still has the problem of high rejection rate when the system is in presence of high reservation rate, because only those non-reservation requests in waiting queue can be backfilled.

To solve the problem of high rejection rate, Kaushik, et al., [18] propose a flexible reservation window scheme. By conducting extensive simulations, Kaushik conclude that when the size of reservation window is equal to the average waiting time in on-demand queue, the reservation blocking probability (also called rejection rate), can be minimized near to zero. However, Kaushik does not address the issue of low resource utilization brought by advance reservation. Wu, et al., [19] study the bandwidth reservation for Grid applications, and propose a adaptive mechanism for malleable bandwidth reservation requests to reduce rejection rate of advance reservation. Unfortunately, Wu's study only adapts to the systems that support malleable reservation requests [23], and it only can be used for bandwidth reservation.

3. Slack Reservation Model

In resource reservation model described in [2], resources are managed by *Reservation Manager* (RM), which performs admission control and tracks the reservations on all resources that under its control. All reservation requests are sent to RM. In this work, we characterize a reservation request by a 3–tuple $< t_i, d_i, c_i >$, where t_i is reservation start time, d_i is reservation deadline, c_i refers to resource demands.

On receiving a request, RM tries to allocate a free slot, which can meet the request's requirements. If a feasible time-slot is available, RM will respond the request with a confirmative response. Then, it is said that a *reservation contract* has been successfully signed between the request and RM. Otherwise, it is said that the reservation request is *rejected*. Given a reservation contract, if RM does not make the resource accessible for the request at time t_i, or fail to keep the resource accessible until time d_i, it is said that a *reservation violation* occurs.

Given current time is t_0, a time-slot table example is shown in Figure 1. The existing reservations are illustrated by rectangles with texture. Considering a reservation request r_i arrives, RM finds that no available free slot can strictly meet the requirements of r_i. However, RM notices that a free slot between r_2 and r_5 seems to be a good candidate, except that the start time and the deadline of r_i are slightly overlapping with other existing reservations. As shown in Figure 1, if RM reserves this time-slot to r_i, then the start time of r_i can not be guaranteed because of r_2 and r_3. Meanwhile, r_i will interfere the start time of r_5 and r_6.

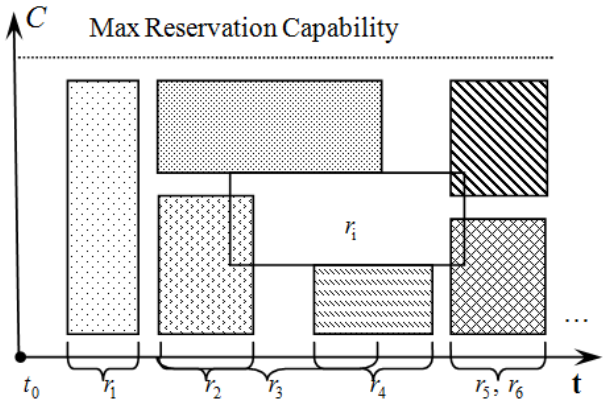


Figure 1. The Time-Slot Illustration for Slack Reservation Strategy

As mentioned before, reservation requests usually tend to overestimate deadline to ensure their successful completion. If the actual deadlines of r_2 and r_3 are earlier than the start time of r_i, and the actual deadline of r_i is earlier than the start time of r_5 and r_6, then, this time slot is feasible for r_i in fact. So, our strategy takes such overestimation into account, and tries to accept some requests, whose reservation requirements can not be met in conventional way. Obviously, this overlapped strategy will take some risks of reservation violation. In the following sections, we will analyze the risks and benefits of this strategy and verify its effectiveness by experiments.

4. Analysis on Slack Reservation Strategy

4.1 Block Set

Consider that reservation request r_i, denoted as $<t_i,d_i,c_i>$, arrivals to the RM. To meet the requirements of r_i, RM should find a free slot which satisfying $C_{free} \geq c_i$ during the period $[t_i,d_i]$, where C_{free} is the amount of free capacity at the time t_i. It is clear that there are two types of existing reservations that will block request r_i. For convenient representation, we define them as two kinds of set:

Definition 1. The start time blocking set of r_i is those existing reservations that their deadlines overlap with r_i, denoted as $S_i^s = \{r_j \mid \forall j \in [1...i-1],\ t_j < t_i < d_j\}$.

Definition 2. The deadline blocking set of r_i is those existing reservations that their start time overlap with r_i, denoted as $S_i^d = \{r_j \mid \forall j \in [1...i-1],\ t_j < d_i < d_j\}$.

Clearly, if $S_i^s = S_i^d = \varnothing$ and $C_{free} \geq c_i$, then request r_i can be guaranteed by the system. This is also the criterion of conventional reservation mechanism while accepting reservation requests. Our SRS follows a more relaxed policy, which tries to accept some reservation requests that do not obey the strict criterion of conventional reservation mechanism. As shown in Figure 1, if the RM decides to accept r_i, then the start time blocking set of r_i is $S_i^s = \{r_2, r_3\}$, and the deadline blocking set is $S_i^d = \{r_5, r_6\}$.

4.2. Risk Analysis

Given there has been $i-1$ existing reservations when r_i arrivals. To know the risk of accepting r_i, we should figure out the probability of reservation violation if it is accepted. Let random event $\mathbf{E_i}$ represent that no reservation violation occurs on r_i, random event $\mathbf{E_i^s}$ represent that all the reservations in S_i^s do not incur the violation of r_i, and random event $\mathbf{E_i^d}$ represent that r_i do not incur any violation on all the reservations in S_i^d. We assume that all reservation requests are independent, thus the probability of $\mathbf{E_i}$ can be expressed as

$$\Pr\{\mathbf{E_i}\} = \Pr\{\mathbf{E_i^s}\} \cdot \Pr\{\mathbf{E_i^d}\} \tag{1}$$

Let random variable $TE_j (j \in [1...i])$ represent the actual deadline of reservation request r_i. For each reservation in S_i^s, if its actual deadline is earlier than the start time of r_i, this reservation will not block r_i actually. Furthermore, if there are a set of reservations in S_i^s whose actual deadlines are earlier than the start time of r_i, then there will be sufficient free resources for r_i to guarantee its start time at time t_i. So the probability that S_i^s do not incur the violation of r_i can be calculated as

$$\Pr\{\mathbf{E_i^s}\} = \Pr\{TE_{j_1} \leq t_i \bigcap TE_{j_2} \leq t_i \bigcap \ldots \bigcap TE_{j_n} \leq t_i \mid C_{free} + \sum_{j \in J} c_j > c_i\} \tag{2}$$

where set $J = \{j_1, j_2 \ldots j_n\} \subseteq S_i^s$. Clearly, we hope to get an optimal set J^* that can

maximize $\Pr\{\mathbf{E_i^s}\}$. So, getting J^* can be expressed as a programming problem that

$$\begin{aligned} &\max \ \Pr\{\mathbf{E_i^s}\} \\ &s.t. \ C_{free} + \sum_{j \in J^*} c_j \geq c_i \\ &\quad J^* \subseteq S_i^s \end{aligned}$$

The approach to find J^* is very similar to 0-1 knapsack algorithm [20], and the details of calculating $\Pr\{\mathbf{E_i^s}\}$ are shown as following.

Algorithm 1: Calculating $\Pr\{\mathbf{E_i^s}\}$

Begin
1. $\Pr\{\mathbf{E_i^s}\} \leftarrow 0$; $J^* \leftarrow \varnothing$
2. **For** each $r \in S_i^s$
3. $\quad V[r] \leftarrow \Pr\{TE_r \leq ts_i\}$;
4. **If** $\Pr\{\mathbf{E_i^s}\} < V[r]$ **Then**
5. $\quad \Pr\{\mathbf{E_i^s}\} \leftarrow V[r]$; $J^* \leftarrow \{r\}$;
6. **End** For
7. **If** $C_{free} + res_{J^*} \geq res_i$ **Then**
8. $\quad$ **Return** $\Pr\{\mathbf{E_i^s}\}$ and J^* ;
9. **Else**
10. $\quad$ **For** each $r \in S_i^s$
11. $\quad\quad$ **If** $r \in J^*$ **Then** Continue;
12. $\quad\quad$ **Else**
13. $\quad\quad\quad J^* \leftarrow J^* + \{r\}$;
14. $\quad\quad\quad$ **For** each $r' \in J^*$
15. $\quad\quad\quad\quad V[r'] \leftarrow V[r'] \cdot \Pr\{TE_{r'} \leq ts_i\}$;
16. $\quad\quad\quad$ **If** $C_{free} + \sum_{j \in J^*} res_j \geq res_i$ **Then**
17. $\quad\quad\quad\quad$ **Return** $\Pr\{\mathbf{E_i^s}\}$ and J^* ;
18. $\quad\quad\quad$ **Else** $J^* \leftarrow J^* - \{r\}$;
19. $\quad$ **End** For

End

To obtain $\Pr\{\mathbf{E_i^d}\}$, we first sort all the $r_k < t_k, d_k, c_k > (r_k \in S_i^d)$ in ascending order of t_k, and store them in a ordered set $\{r_{k_1}, r_{k_2}, \ldots, r_{k_m}\}$. Then, we should find k_l that satisfying $\sum_{j=k_1}^{k_{l-1}} c_j \leq C_{\max} - c_i < \sum_{j=k_1}^{k_l} c_j$, where $C_{\max}$ is the total capacity of resources. As long as the actual deadline of r_i is earlier than t_{k_l}, we can ensure that r_i will not incur any violation on all the reservation in S_i^d. Therefore, we can get

$$\Pr\{\mathbf{E_i^d}\} = \Pr\{TE_i \leq t_{k_l}\} \tag{3}$$

Without loss of generality, we assume that the service time of requests following exponential distribution with rate μ, then its cumulative distribution function can be expressed as $F(t)=1-e^{-u\cdot t}$. So, we can rewrite (2) and (3) as following

$$\Pr\{\mathbf{E_i^s}\}=\prod_{j\in J^*}(1-e^{(t_i-t_j)\cdot\mu}) \tag{4}$$

$$\Pr\{\mathbf{E_i^d}\}=1-e^{(t_{k_l}-t_i)\cdot\mu} \tag{5}$$

Summary the above analysis, we list the steps to calculate the probability that no violation occurs on r_i if accepting it as following. It is clear that the risk of accepting r_i is $1-\Pr\{\mathbf{E_i}\}$.

Algorithm 2: Calculating $\Pr\{\mathbf{E_i}\}$

Begin

1. Get sets S_i^s and S_i^d.
2. Using 0-1 knapsack algorithm to find J^*.
3. Calculate $\Pr\{\mathbf{E_i^s}\}$ by calling algorithm 1.
4. Sort all the $r_k \in S_i^d$ in ascending order of t_k.
5. Find k_l that satisfying $\sum_{j=k_1}^{k_{l-1}} c_j \le C_{\max}-c_i < \sum_{j=k_1}^{k_l} c_j$.
6. Calculate $\Pr\{\mathbf{E_i^d}\}$ by formula (5).
7. Return $\Pr\{\mathbf{E_i^s}\}\cdot\Pr\{\mathbf{E_i^d}\}$

End

4.3. Benefit Analysis

After knowing $\Pr\{\mathbf{E_i}\}$, we can also calculate the expected benefits of accepting r_i. Given p is the resource price under condition that no reservation violation occurs, q is penalty price that RM should pay if reservation violation occurs. If r_i is accepted, the expected benefits b_i can be estimated as

$$\mathrm{E}(b_i)=p\cdot\Pr\{\mathbf{E_i}\}-q\cdot(1-\Pr\{\mathbf{E_i}\}) \tag{6}$$

In Figure 2, we describe the relationship between $\Pr\{\mathbf{E_i}\}$ and $\mathrm{E}(b_i)$. As Shown in Figure 2, if $\Pr\{\mathbf{E_i}\}>v$, then the expected benefits $\mathrm{E}(b_i)$ is positive with maximal value p, which means the acceptance of r_i may be profitable. Hence, a RM may simply use v as its threshold to decide whether accepts r_i or not. However, $\Pr\{\mathbf{E_i}\}<v$ only means that accepting r_i is risky instead of non-profitable. So, a RM may use a more suitable $v_*(0<v_*<1)$ as its criterion to decide whether accept requests r_i or not. A high value of v_* indicates that RM is conservative when using SRS; a low v_* means that it is willing to take more risks to improve utilization of its resources; when $v_*=1$, the SRS is the same as conventional reservation policy. So, the SRS is more general than conventional policy. The RM can adjust its parameter v_* at runtime according to its resource management policy, even retrieve to conventional reservation mechanism.

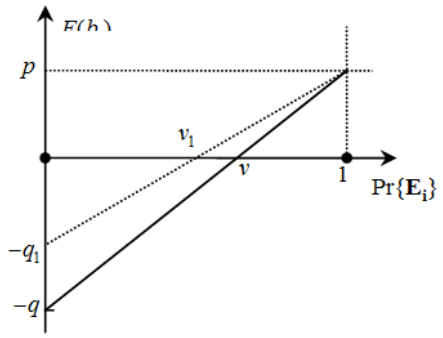


Figure 2. Relationship Between $\Pr\{\mathbf{E_i}\}$ and $\mathrm{E}(b_i)$

In another scenario, if RM can flexibly adjust its penalty price q, the relationship between $\Pr\{\mathbf{E_i}\}$ and $\mathrm{E}(P_i)$ will be changed correspondingly. In Figure 2, we also illustrate the case that RM chooses a lower penalty price q_1. Correspondingly, v changes into v_1, and $v_1 < v$. It means that to get the same $\mathrm{E}(b_i)$, a lower penalty price leads to lower risks. That is consistent with our common sense. The optimal price scheme about p and q is out of the scope of this paper, so we do not discuss too much about it here.

5. Experiment and Performance Evaluation

In this section, we conduct extensive simulations to verify the performance of SRS when co-reservation for multiple resources is required. We compare the effectiveness of SRS with conventional reservation policy and backfilling technique. In addition, we also evaluate the risk we should take when using SRS.

5.1. Experimental Setting

We use GridSim [21], a distributed resource management and scheduling simulator, to conduct simulations. A multi-cluster computational Grid model is constructed, which consists of six clusters. The detail setting of each cluster is listed in Table 1.

Table 1. Grid System Model in Experiments

Cluster ID	Processor numbers	MIPS / processor
Cluster 1	32	377
Cluster 2	100	410
Cluster 3	64	380
Cluster 4	64	285
Cluster 5	120	285
Cluster 6	100	515

In simulations, we choose Lublin-Feitelson Model [22], which is derived from real workload logs, to generate experimental workload (reservation requests). The workload consists of 10000 requests, which is characterized by arrival time, number of processors, and running time. As the model is based on long-term jobs on supercomputer, we divided the arrival times and running times by 60 to reduce the overall time of simulations. There is no start time of reservation in this workload, so we

append each request with a reservation start time. The start time is set by adding the arrival time with a random value that uniformly distributed in interval $[100s,1000s]$. To reflect the overestimation of reservation deadline, we multiply the running time of each request with a random factor k_{over} (more details about k_{over} is discussed in Section 5.3). To increase the probability that the requests require co-reservation from multiple clusters, the resource demands of each reservation request is enlarged 20 times.

5.2. Resource Utilization and Rejection Rate

Firstly, we compare the performance of SRS with conventional advance reservation policy and backfilling technique. The basic workload used consists of 10000 requests, and it is modified into four different workloads, each with advance reservation rate of 10%, 15%, 20%, 25% respectively. In this experiment, we set $v_* = 0.8$, which means that RM will accept a reservation request only when the probability of reservation violation for this request is less than 20%; and the factor k_{over} is set to be uniformly distributed in the interval [1.2, 1.5], which means reservation requests tend to overestimate their relative deadline with mean value 35%. In next section, we will investigate the effects of v_* and k_{over} on the performance of SRS. The experimental results are shown in Figure 3 and Figure 4.

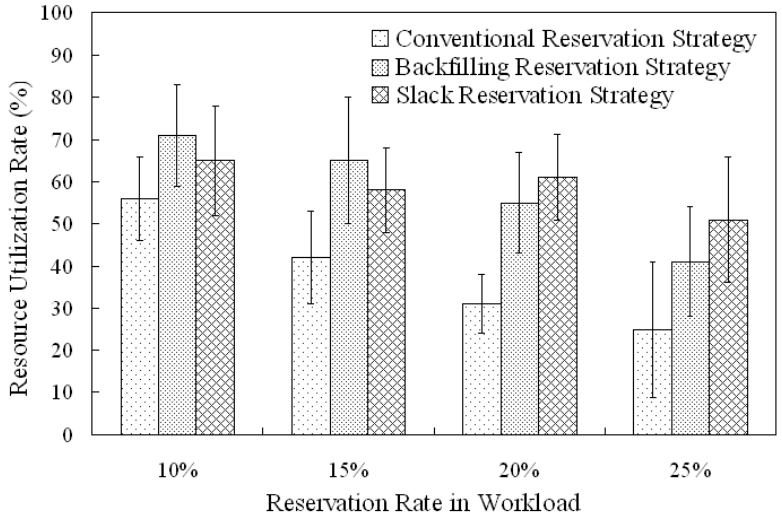


Figure 3. Resource Utilization Rate with different Reservation Rate

In simulations, we set that the requests without advance reservation requirements will be canceled, if its deadline expires and still has not been scheduled on required resources. Such cases often happen on the requests with co-reservation requirement, which in turn intensifies the decline of resource utilization. As we can see from Figure 3, for conventional reservation policy, with reservation rate increasing from 10% to 25%, resource utilization drops dramatically from 56% to about 25%. It is because that many requests have been blocked by existing reservations. When applying backfilling technique, utilization rate have been increased significantly about 14% comparing with conventional reservation policy when reservation rate is about 10% and 15%. At such low level of reservation rate, the performance of SRS is not as good as backfilling, but still better than conventional reservation policy. As the reservation rate increase to 20%

and 25%, SRS's resource utilization becomes higher than both conventional mechanism and backfilling. An interesting finding is that SRS's utilization increases about 3% when reservation rate increases from 15% to 20%. The reason is that there are more free slots can be allocated by using SRS as reservation requests increases. However, such increasing can not be sustained when the reservation rate increases to 25%.

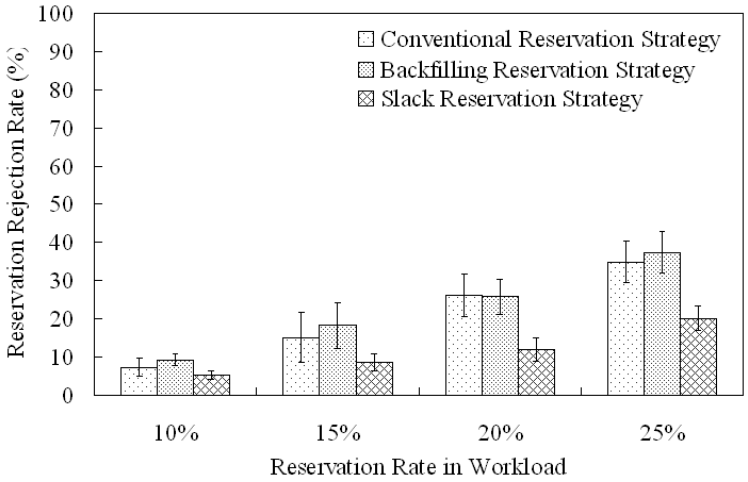


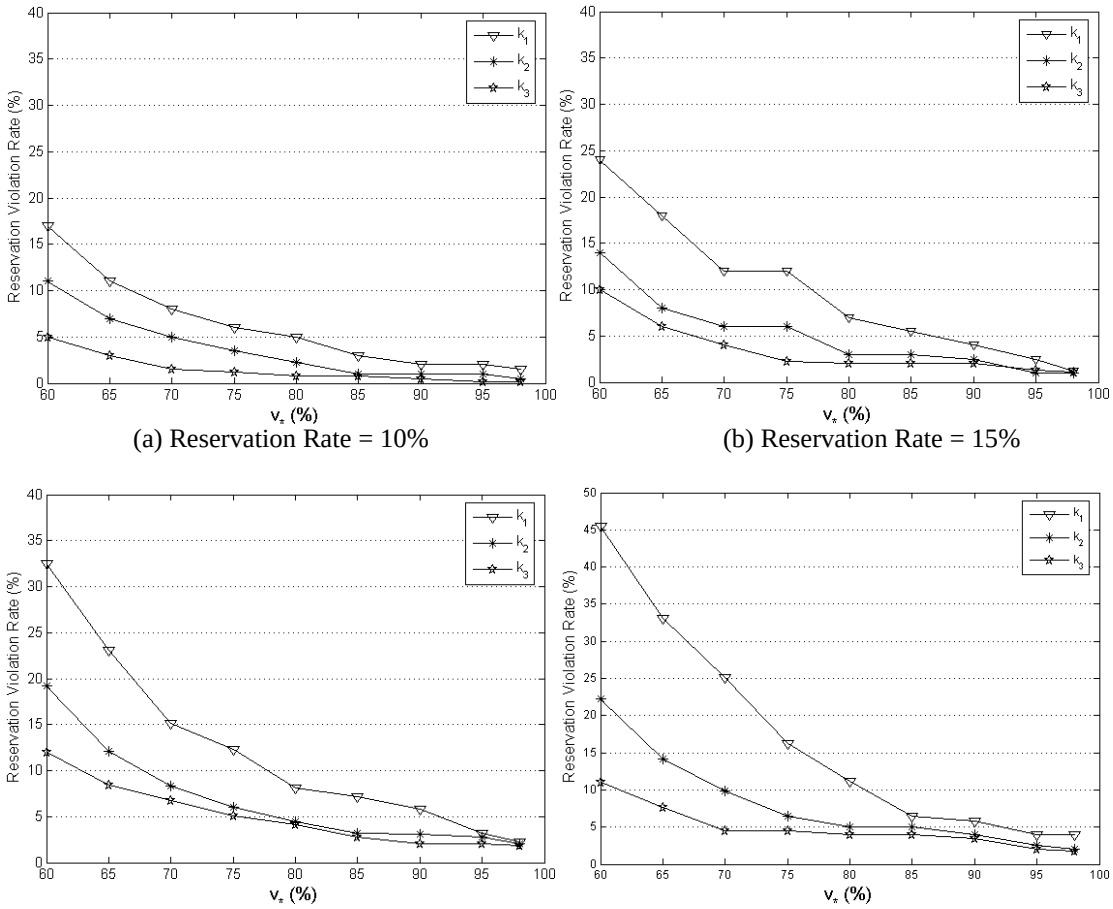
Figure 4. Reservation Rejection Rate with different Reservation Rate

Reservation rejection rate is shown in Figure 4. Like the resources utilization, when using conventional reservation policy, the rejection rate increases sharply from about 7% to 34% as the reservation rate increases from 10% to 25%. Also, we notice that backfilling can not lower down rejection rate, on the contrary, it leads to a slightly higher rejection rate comparing with conventional policy. That is because our workload is very long (10000 requests), RM only deal with a small part of reservation requests each time. So, some non-reservation requests are backfilled into a lot of free slots, which probably may be allocated to other reservation requests later if using SRS. That is why backfilling technique leads a higher rejection rate. When SRS is used, the rejection rate is only about 50% of conventional reservation policy in all cases.

Based on above experimental results, we draw the following conclusions: (1) backfilling is effective to improve resource utilization when reservation rate is below 15%; (2) when the system faces high reservation rate (>15%), SRS is more effective than backfilling; (3) SRS can reduce rejection rate as long as applications overestimate their deadlines.

5.3. Reservation Violation

In this section, we focus on reservation violation, which is caused by using SRS. As our experiments are conducted on simulator, violations cased by network disconnection, system crash and etc. are all ignored. So, we assume that the violation rate is zero when using conventional reservation policy or backfilling technique, and only investigate the violation when SRS is used. In this experiment, the effects of v_* and k_{over} on the performance of SRS is extensively investigated. In Figure 5, reservation violations for four reservation rates are shown respectively.



(a) Reservation Rate = 10%

(b) Reservation Rate = 15%

(c) Reservation Rate = 20%

(d) Reservation Rate = 25%

Figure 5. Reservation Violation Rate with Different Reservation Rate

As mentioned before, we multiply the deadline of each reservation with a factor $k_{over}(k_{over} \geq 1)$ to reflect the overestimation of deadline. In this experiment, we have conducted simulations with different level of k_{over}. More specifically, we set k_{over} uniformly distributed in the different intervals, such as [1.0, 1.2], [1.2, 1.5], and [1.5, 1.8] respectively. We denote them as k_1, k_2, k_3 for the sake of simplifying representation. When k_{over} is set in level k_1, it means requests in workload tend to overestimate their deadlines with mean value 10%. So, k_2 means 35% overestimation, and k_3 means 65% overestimation. As shown in Figure 5, it is clearly that more overestimation of deadline leads to lower reservation violation rate. That is because many overlapped reservations do not overlap actually in run time, which makes SRS more effective.

Parameter v_* is the threshold, by which RM decides whether a free time slot can be allocated to an overlapped reservation or not. So, v_* is a strategic parameter of RM, a higher value of v_* indicates that RM is conservative. In this experiment, we increase v_* from 0.6 to 0.95 gradually. The results show that for k_1 and k_2 the violation rate drops quickly when v_* is increased from 0.6 to 0.8, then the decline becomes stable. In all cases tested, we found that, the reservation violation rate can be limited below 10%

when $v_* \geq 0.8$. If we set $v_* \geq 0.9$, the violation can be controlled below 5%. In Figure 5(c) and Figure 5(d), the experimental results indicate that low value of v_* is not a good idea when system is in presence of high reservation rate (>20%).

The increasing of reservation violation is the price we have to pay while using SRS, experimental results indicate that we can limit the violation rate in a relative low level by adjusting parameter v_*. Currently, many systems have support reservation re-negotiation mechanism, such as Maui [24], EASY [15], COSY [6]. Combining SRS and reservation re-negotiation, the RM can take the advantages of SRS as well as avoiding the risk of reservation violation.

It is noteworthy that different resource providers may prefer different v_* in practical systems, and RM may dynamically set v_* for different type of resources based on its resource reservation policies. In our simulations, we set an identical v_* for all resources only for simplicity.

6. Conclusion

In this paper, we study the negative effects brought about by advance reservation in computational grid. To mitigate those effects, we proposed a novel slack reservation strategy (SRS) based on the fact that applications tend to overestimate their running time to ensure their completion. Extensive simulations based on real workload are conducted to verify the effectiveness of our policy. Experimental results show that the relaxed reservation policy can bring about higher resource utilization and lower rejection rate at the price of a slightly increasing of reservation violations. Furthermore, the policy also shows adaptive in presence of higher reservation rate. Although the increasing of reservation violation rate is the price we have to pay, we can still limit it by setting a higher value of v_*. For those systems that support reservation re-negotiation, combining SRS and re-negotiation can avoid most reservation violation caused by SRS.

For the future work, as parameter v_* have significant influences on the performance of SRS, we plan to provide an adaptive mechanism for RM to dynamically set optimal v_* based on resource's runtime load. Also, we also plan to incorporate SRS into grid economy to provide a more flexible price mechanism. For example, we are now taking efforts to device a SRS-based auction mechanism, in which each resource is labeled by a list of pairs $< price, risk >$. In this way, consumers can flexibly select the resources based on their budgets and how much risk they are willing to take.

Acknowledgements

This work is supported by grants from the National Natural Science Foundation of China (No.60970038 and No. 61272148). It is also supported by the Provincial Science & Technology plan project of Hunan (No.2012GK3075).

References

[1] I. Foster and C. Kesselman, "The Grid: Blueprint for a New Computing Infrastructure", Second Edition. Elsevier Inc., San Francisco, **(2004)**.

[2] I. Díaz, G. Fernandez, P. Gonzalez, M. J. Martín and J. Touriño, "Extending the Globus Information Service with the Common Information Model", Proceedings of the 9th International Symposium on Parallel and Distributed Processing with Applications, Busan, Korea, **(2011)** May 26-28, pp. 113-119.

[3] A. S. Grimshaw and A. Natrajan, "Legion: Lessons Learned Building a Grid Operating System. Proceedings of the IEEE, vol. 3, no. 93, **(2005)**, pp. 589-603.
[4] I. Foster, A. Roy and V. Sander, "A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation", Proceedings of International Workshop on QoS, Karlsruhe, USA, **(2000)** June 5-8, pp. 181-188.
[5] W. Smith, I. Foster and V. Taylor, "Scheduling with Advanced Reservations", Proceedings of International Symposium on Parallel and Distributed Processing, Cancun, Mexico, **(2000)** May 1-5, pp. 127-132.
[6] J. W. Cao and F. Zimmermann, "Queue Scheduling and Advance Reservations with COSY", Proceedings of the 18th International Parallel and Distributed Processing Symposium, Santa Fe, New Mexico, USA, **(2004)** April 26-30, pp. 1-8.
[7] R. Santos, G. Lipari, E. Bini and T. Cucinotta, "On-line Schedulability Tests for Adaptive Reservations in Fixed Priority Scheduling", Real-Time Systems, vol. 5, no. 48, **(2012)**, pp. 601-634.
[8] C. Castillo, G. N. Rouskas and K. Harfoush, "Online Algorithms for Advance Resource Reservations", Journal of Parallel and Distributed Computing, vol. 7, no. 71, **(2011)**, pp. 963-973.
[9] M. Wu, X. H. Sun and Y. Chen, "QoS Oriented Resource Reservation in Shared Environments", Proceedings of International Symposium on Cluster Computing and the Grid, Singapore, **(2006)** May 16-19, pp. 601-608.
[10] Y. Li, S. Ranka and S. Sahni, "In-advance Path Reservation for File Transfers in e-Science Applications", Journal of Supercomputing, vol. 3, no. 59, **(2012)**, pp. 1167-1187.
[11] C. Li and L. Li, "Design and Implementation of Economics-based Resource Management System in Ad Hoc Grid", Advances in Engineering Software, vol. 1, no. 45, **(2012)**, pp. 281-291.
[12] D. J. Veit and W. Gentzsch, "Grid Economics and Business Models", Journal of Grid Computing, vol. 3, no. 6, **(2008)**, pp. 215-217.
[13] A. Roy and V. Sander, "Advance Reservation API", GFD-E.5, Scheduling Working Group, Global Grid Forum, **(2002)**.
[14] S. Zikos and H. D. Karatza, "The Impact of Service Demand Variability on Resource Allocation Strategies in a Grid System", ACM Transactions on Modeling and Computer Simulation, vol. 4, no. 20, **(2010)**.
[15] A. W. Mu'alem and D. G. Feitelson, "Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling", IEEE Transactions on Parallel and Distributed Systems, vol. 6, no. 12, **(2001)**, pp. 529-543.
[16] A. Sodan and X. Huang, "Adaptive Time/Space Scheduling with SCOJO", Proceedings of International Symposium on High Performance Computing Systems, Winnipeg, Canda, **(2004)** May 16-19, pp. 165-176.
[17] A. Sulistio, W. Schiffmann and R. Buyya, "Advanced Reservation-based Scheduling of Task Graphs on Clusters", Proceedings of International Conference on High Performance Computing, Bangalore, India, **(2006)** December 18-21, pp. 18-21.
[18] N. R. Kaushik, S. M. Figueira and S. A. Chiappari, "Flexible Time-Windows for Advance Reservation Scheduling", Proceedings of International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, Monterey, Canada, **(2006)** September 11-14, pp. 218-225.
[19] L. Wu, C. Wu, J. Cui and J. Xing, "An Adaptive Advance Reservation Mechanism for Grid Computing", Proceedings of International Conference on Parallel and Distributed Computing, Applications and Technologies, Dalian, China, **(2005)** December 5-8, pp. 400-403.
[20] D. El Baz and M. Elkihel, "Load Balancing Methods and Parallel Dynamic Programming Algorithm using Dominance Technique Applied to the 0-1 Knapsack Problem", Journal of Parallel and Distributed Computing, vol. 1, no. 65, **(2005)**, pp. 74-84.
[21] A. Sulistio, U. Cibej, S. Venugopal, B. Robic and R. Buyya, "A Toolkit for Modelling and Simulating Data Grids: an Extension to GridSim", Concurrency and Computation-Practice & Experience, vol. 13, no. 20, **(2008)**, pp. 1591-1609.
[22] U. Lublin and D. G. Feitelson, "The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs", Journal of Parallel and Distributed Computing, vol. 11, no. 63, **(2003)**, pp. 1105-1122.
[23] L. O. Burchard, "On the Performance of Computer Networks with Advance Reservation Mechanisms", Proceedings of the International Conference on Networks, Sydney, Australia, **(2003)** September 28-31, pp. 449-454.
[24] Z. Liu, A. Liang and L. Xiao, "A Parallel Workload Model and its Implications for Maui Scheduling Policies", Proceedings of International Conference on Computer Modeling and Simulation, Hainan, China, **(2010)** January 22-24, pp. 384-389.

Authors

Peng Xiao received the Ph.D degree in computer science from the Central South University in 2009. Currently, he is an associate professor in the Hunan Institute of Engineering. Also, he is the advanced network engineer in HP High-performance Network Centre in Hunan. His research interests include grid computing, distributed resource management. He is a member of ACM, IEEE, and IEEE Computer Society.

Ning Han received his bachelor degree in Beijing University of Science and Technology, and now persuading master degree in Xiangtan University. Currently, he works in the HP High Performance Lab of Hunan Institute of Engineering as a senior networking engineer. His research interests include complex networking deployment, distributed computing, information security technology, fault-tolerance in distributed systems.