Two-Dimension Relaxed Reservation Policy for Independent Tasks in Grid Computing

Peng Xiao
School of Computer and Communication, Hunnan Institute of Engineering, Xiangtan, 411104, China
Email: xpeng4623@yahoo.com.cn

Zhigang Hu
School of Information Science and Technology, Central South University, Changsha, 410083, China
Email: zghu@mail.csu.edu.cn

***Abstract*—As an effective technique for QoS provision, reservation service has been widely applied in various grids. However, plenty of studies have shown that reservation service will bring about many negative effects on system performance, i.e. higher rejection rate and lower resource utilization. To mitigate these negative effects, a relaxed reservation policy is proposed in this paper. It allows grids accepting reservation requests that overlapping with existing ones in two dimensions (temporal and space). In addition, the proposed policy is completely compatible with conventional reservation service, which means it can be applied in all kinds of practical grids. Extensive experiments have been conducted by using real workloads, and the experimental results show that the relaxed reservation policy can achieve higher resource utilization and lower rejection rate comparing to other policies. Also, It shows better adaptiveness when grids are in presence of higher reservation rate.**

***Index Terms*—grid computing, resource reservation, relaxed policy, quality of service, reservation violation, time slot**

I. Introduction

Resource reservation is a process of requesting resources for use at a specific time in the future [1]. In grid computing [2], reservation service provides confidence that a subsequent resource allocation request will succeed, in this way, it allows users to gain concurrent access to adequate resources and guarantees the availability of resources at required times [3]. Therefore, reservation mechanism has been widely applied to provide end-to-end QoS for high-performance applications in many grids [4]. Although the effectiveness and the advantages of reservation have been proven in many practical grids, it also brings about many negative effects on system performance. Studies in [4-12] have shown that fixed-capability reservation will result in low resource utilization, and excessive reservation often leads to high rejection rate. Therefore, how to mitigate these negative effects has becomes an important issue that needs to be addressed [4, 6-8].

Performance reports on practical grids have indicated that unpredictable workload and dynamic availability of resources are the two significant characteristics in grid environments [5, 7-9]. Therefore, it is difficult for applications to precisely estimate the reservation time, if not impossible. As a result, grid applications tend to overestimate the reservation time so as to ensure their successful execution [6-9]. This behavior inevitably results in high rejection rate and low resource utilization.

Motivated by these observations, in this paper we propose a *Two-Dimension Relaxed Reservation Policy* (TDRRP), which allows grids to accept the reservation requests that overlaps with existing ones under certain conditions. By this way, TDRRP is capable of mitigating the negative effects in conventional reservation service.

The rest of this paper is organized as following: Section 2 presents the related work; Section 3 introduces the key ideology of TDRRP; Section 4 presents a relaxed co-reservation admission algorithm; In section 5, experiments are conducted to verify the performance of TDRRP. Finally, Section 6 concludes the paper with a brief discussion of the future work.

II. Related work

Since the reservation service has been incorporated in GARA architecture, its effects on system performance have been widely studied in many works [9-12]. In [9], Foster et al. investigate the impacts of reservation on the performance of scheduler in the metrics of *Mean Waiting Time*, *Mean Offset Time*, and *Request Rejection Rate*. The experimental results show that *Mean Offset Time* will be reduced when the reservation rate is low, however, in face of high reservation rate all these three metrics will increase significantly. These conclusions are repeatedly confirmed in [6-8, 10-12]. In [10], Wu and Sun study the effects of reservation on remote jobs and local jobs in non-dedicated environments. They use queuing system to model resources, and their studies also prove that excessive reservation will prolong the response time of remote jobs as well as local jobs.

Therefore, many frameworks and techniques have been proposed to overcome the demerits of reservation service [4-6, 11]. For example, Foster et al. suggest that incorporating adaptive mechanism into reservation service is a feasible approach to improve the conventional reservation mechanism, and they also develop an extended GARA architecture to provide more flexible reservation service [4]. In this extended GARA framework, reservation service is enhanced with *intelligent decision model* and *performance sensors*. However, this adaptive reservation architecture is designed for the applications with high-bandwidth and dynamic flows, which means it only adapts to bandwidth reservation for bulk-data transfer.

Recently, researchers begin to focus on more applicable and convenient techniques for mitigating the negative effects of reservation service. Backfilling [13] is one of the most famous techniques to improve resource utilization for those reservation-aware systems. Normally, backfilling techniques are categorized into two classes: (1) *Aggressive Backfilling*: small tasks are moved ahead to fill in holes in the time slot, provided they do not delay the first tasks in the waiting queue; (2) *Conservative Backfilling*: small tasks are moved ahead only if they do not delay any task in the waiting queue. So, the backfilling technique can provide improved responsiveness for short tasks combined with no starvation for long ones. In [7], backfilling-based gang scheduling is implemented and incorporated into the SCOJO scheduler for co-reservation service. The simulation results show that it can mitigate the negative effects of reservation for the applications with high ratio of computing to communication. In [14], Sulistio *et al.* apply several techniques, including *re-arranging subtask*, *interweaving task graphs* and *backfilling*, into reservation service with aiming to improve resource utilization. Although backfilling technique is effective to improve resource utilization and throughput, it still has the problem of high rejection rate when grids are in presence of high reservation rate [7, 14].

To solve the problem of high rejection rate, Kaushik et al. propose a flexible reservation window scheme [15]. By extensive simulations, Kaushik concludes that when the size of reservation window is equal to the average waiting time in on-demand queue, the 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 reservation service. In [16], Wu et al. study the bandwidth reservation for grid applications, and propose an adaptive mechanism for malleable bandwidth reservation requests to reduce rejection rate of reservation. Unfortunately, Wu's study only adapts to the systems that support malleable reservation requests, and it only can be used for bandwidth reservation.

III. Problem definition

In reservation framework described in the standard document GFD.5, resources are managed by *Reservation Manager* (RM), which performs admission control and tracks the reservations on all resources under its control. On receiving a reservation request, RM tries to allocate a free time-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 the reservation start time, or fail to keep the resource accessible during the reservation duration, it is said that a *reservation violation* occurs. So, the reservation service can be described as following.

Definition 1. A reservation request can be characterized as a 3–tuple $<ts,te,res>$, where ts is the reservation start time, te is reservation deadline, res refers to resource demands.

Definition 2. A time-slot can be characterized as a 3–tuple $<slot\_ts,slot\_te,slot\_res>$, where $slot\_ts$ is the start time of the time-slot, $slot\_te$ is the deadline of the time-slot, $slot\_res$ refers to the available resources between the duration $[slot\_ts,slot\_te]$.

Definition 3. Given a time-slot $slot_k$ and a reservation request req_i, $slot_k$ can rigidly meet the requirements of req_i if $slot\_ts_k \leq ts_i \cap slot\_te_k \geq te_i \cap slot\_res_k \geq res_i$

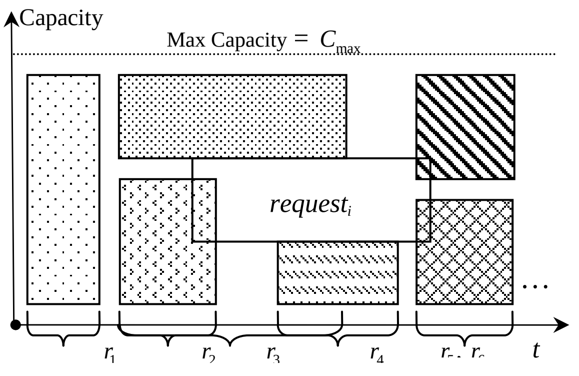


Figure 1. A example of time slot table

Given current time is t_0, a time-slot table is shown in Fig. 1. The existing reservations are illustrated by rectangles with texture. Considering a reservation request $request_i$ arrives, RM finds that no available free slot can rigidly meet the requirements of $request_i$. 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 $request_i$ are slightly overlapping with other existing reservations. If RM reserves this time slot to $request_i$, then the start time of $request_i$ can not be guaranteed because of r_2. Meanwhile, $request_i$ will overlap with the start time of r_5 and r_6.

As mentioned before, reservation requests usually tend to overestimate deadline to ensure their successful completion. If the actual deadlines of r_2 are earlier than the start time of $request_i$, and the actual deadline of $request_i$ is earlier than the start time of r_5 and r_6, then, this time slot is feasible for $request_i$ in practice. So, our strategy takes such overestimation into account, and tries to accept some requests, whose reservation requirements can not be met in conventional way.

Definition 4. Given a time-slot $slot_k$ and a reservation request req_i, let random event $\mathbf{E_i}$ represent that no reservation violation occurs on req_i. Then, $slot_k$ can relaxed meet the requirements of req_i with the probability $\Pr\{\mathbf{E_i}\}$ as following

$$\Pr\{\mathbf{E_i}\} = \Pr\{slot\_ts_k \le ts_i \cap slot\_te_k \ge te_i \cap slot\_res_k \ge res_i\} \tag{1}$$

IV. Relaxed reservation policy

A. Overlapping Sets

Considering a reservation request req_i, denoted as $<ts_i, te_i, res_i>$, arrivals to a RM. To meet the requirements of req_i, RM should find a free slot which satisfying $C_{free} \ge res_i$ during the period $[ts_i, te_i]$, where C_{free} is the amount of free capacity. If there is no such time-slot that can rigidly meet the requirements of req_i, there must be one ore more existing reservations that overlapping with the reservation time of req_i. As shown in Fig. 1, it is clear that these existing reservations can be classified as two kinds.

Definition 5. The start time overlapping set of req_i is those existing reservations that their deadlines overlap with req_i, denoted as

$$set_i^s = \{req_j \mid \forall j \in [1 \ldots i-1],\ st_j < st_i < et_j\}$$

Definition 6. The deadline overlapping set of req_i is those existing reservations that their start time overlap with req_i, denoted as

$$set_i^e = \{req_j \mid \forall j \in [1 \ldots i-1],\ st_j < et_i < et_j\}$$

Clearly, if $set_i^s = \varnothing \cap set_i^e = \varnothing \cap C_{free} \ge c_i$, then request req_i can be guaranteed. This is also the criterion of conventional reservation mechanism while accepting reservation requests. In the TDRRP, the system will follow 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 Fig. 1, if the RM decides to accept req_i, then the start time overlapping set of req_i is $set_i^s = \{r_2, r_3, r_4\}$, and the deadline overlapping set is $set_i^e = \{r_4, r_5, r_6\}$.

Given there has been $i-1$ existing reservations when req_i arrivals. To know the risk of accepting req_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 req_i, random event $\mathbf{E_i^s}$ represent that all the reservations in set_i^s do not incur the violation of req_i. Let random event $\mathbf{E_i^e}$ represent that req_i do not incur any violation on all the reservations in set_i^e. We assume that all reservation requests are independent, so the probability of $\mathbf{E_i}$ can be expressed as

$$\Pr\{\mathbf{E_i}\} = \Pr\{\mathbf{E_i^s}\} \cdot \Pr\{\mathbf{E_i^e}\} \tag{2}$$

For a job with m reservation requests, the probability of successful reservation for this job can be expressed as

$$\Pr\{\mathbf{E}\} = \prod_{i=1}^{m} \Pr\{\mathbf{E_i^s}\} \cdot \Pr\{\mathbf{E_i^e}\} \tag{3}$$

where $\mathbf{E}$ is a random variant representing that relaxed reservation policy does not result in reservation violation.

B. Caculation of $\Pr\{\mathbf{E_i^s}\}$

According to definition 5, the deadline of all reservations in set_i^s are later than the start time of req_i. If some of these existing reservations release their resource before the start time of req_i, then the system is able to reserve enough resource for req_i at the time of ts_i, which means that reservation violation will not happen on req_i. So, Let random variable $TE_j\,(j \in [1 \ldots i])$ represent the actual deadline of reservation request r_j $(r_j \in set_i^s)$, then the probability that set_i^s do not incur the violation of req_i can be calculated as

$$\Pr\{\mathbf{E_i^s}\} = \Pr\{TE_{j_1} \le t_i \cap TE_{j_2} \le t_i \cap \ldots \cap TE_{j_n} \le t_i \mid slot\_res + \sum_{j \in J} res_j > res_i\} \tag{4}$$

where set $J = \{r_{j_1}, r_{j_2}, \ldots r_{j_n}\}$ and $J \subseteq set_i^s$.

It is clear that there might be many set J that can satisfy the condition of (4). For example, $set_i^s = \{r_2, r_3, r_4\}$ as shown in Fig. 1, then $J = \{r_2\}$ or $J = \{r_3\}$ or $J = \{r_2, r_3\}$ can all be applied in (4) to calculate $\Pr\{\mathbf{E_i^s}\}$. So, the question is which one is the optimal J^*, which can be expressed as a programming problem as following

$$\begin{aligned} &\max \quad \Pr\{\mathbf{E_i^s}\} \\ &s.t. \quad slot\_res + \sum_{j \in J^*} res_j \ge res_i \\ &\qquad J^* \subseteq set_i^s \end{aligned}$$

The approach to find J^* is very similar to 0-1 knapsack algorithm [17], and the details of calculating $\Pr\{\mathbf{E_i^s}\}$ are shown as following.

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

```
  Input: $set_i^s$, $<ts_i, te_i, res_i>$, $slot\_res$
  Output: $\Pr\{\mathbf{E_i^s}\}$, $J^*$
Begin
1.  $\Pr\{\mathbf{E_i^s}\} \leftarrow 0$; $J^* \leftarrow \varnothing$
2.  For each $r \in set_i^s$
3.      $V[r] \leftarrow \Pr\{TE_r \le ts_i\}$;
4.    If $\Pr\{\mathbf{E_i^s}\} < V[r]$ Then
5.      $\Pr\{\mathbf{E_i^s}\} \leftarrow V[r]$; $J^* \leftarrow \{r\}$;
6.  End For
7.  If $slot\_res + res_{J^*} \ge res_i$ Then
8.      Return $\Pr\{\mathbf{E_i^s}\}$, $J^*$;
9.  Else
10.     For each $r \in set_i^s$
11.         If $r \in J^*$ Then Continue;
12.         Else
13.             $J^* \leftarrow J^* + \{r\}$;
14.             For each $r' \in J^*$
15.                 $V[r'] \leftarrow V[r'] \cdot \Pr\{TE_{r'} \le ts_i\}$;
16.             If $slot\_res + \sum_{j \in J^*} res_j \ge res_i$ Then
17.                 Return $\Pr\{\mathbf{E_i^s}\}$, $J^*$;
18.             Else $J^* \leftarrow J^* - \{r\}$;
19.     End For
End
```

C. *Caculation of* $\Pr\{\mathbf{E_i^e}\}$

$\Pr\{\mathbf{E_i^e}\}$ is the probability that req_i won't lead to reservation violation in set_i^e. Unlike the calculation of $\Pr\{\mathbf{E_i^s}\}$, the key point of calculating $\Pr\{\mathbf{E_i^e}\}$ is to figure out the time when req_i will release its resources.

Let random TE_i represent the deadline of req_i, and $\{r_{k_1}, r_{k_2}, \cdots, r_{k_n}\}$ represent the reservations in set_i^e. Assuming that $\{r_{k_1}, r_{k_2}, \cdots, r_{k_n}\}$ is sorted in ascending order of reservation start time, it is clear that we should find the k_l that satisfying

$$\sum\nolimits_{j=k_1}^{k_{l-1}} res_j \le res_{\max} - res_i < \sum\nolimits_{j=k_1}^{k_l} res_j \qquad (5)$$

where $res_{\max}$ is the max capacity of the resource. As long as the actual deadline of req_i is earlier than ts_{k_l}, we can ensure that req_i will not incur any violations on all the reservations in set_i^e, and the probability that req_i won't lead to reservation violation is

$$\Pr\{\mathbf{E_i^e}\} = \Pr\{TE_i \le ts_{k_l}\} \qquad (6)$$

The algorithm of calculating $\Pr\{\mathbf{E_i^e}\}$ is as following

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

```
  Input: $set_i^e$, $<ts_i, te_i, res_i>$, $res_{\max}$
  Output: $\Pr\{\mathbf{E_i^e}\}$
Begin
1.  Sort all the $r_k \in set_i^e$ in ascending order of $ts_k$;
2.  Store the sorting result as $\{r_{k_1}, r_{k_2}, \cdots, r_{k_n}\}$;
3.  For $l \leftarrow 1$ to $n$
4.      If $\sum_{j=k_1}^{k_{l-1}} res_j \le res_{\max} - res_i < \sum_{j=k_1}^{k_l} res_j$ Then
5.          Break;
6.  End For
7.  Calculating $\Pr\{\mathbf{E_i^e}\}$ by (6)
End
```

D. *Admission Algorithm of Relaxed Co-reservation*

Parameter Sweep Application (PSA) [18-19] is an important class of grid applications, which arises in scientific and engineering contexts. A significant challenge of deploying PSA is the concurrent reservation of a huge number of resources in grid environments [20-21]. So, in this paper we focus on the co-reservation for PSA by using relaxed reservation policy.

Given the grid system has N computing sites, noted as $(CE_1, \ldots, CE_N)$. A PSA task consists of m subtasks $<sub_1, sub_2 \ldots, sub_m>$, and vector $\mathbf{R} =< sr_1, sr_2, \ldots, sr_m >$ is the resource requirements of each sub-tasks. Therefore, a co-reservation scheme is the mapping of resource requirements to computing sites, noted as $S : \mathbf{R} \times \{1, \ldots, N\} \to \{0,1\}$. It is clear that a co-reservation scheme S can be noted as a $m \times N$ matrix as shown in Fig. 2. In the co-reservation matrix, $S_{i,j} = 1$ means reserving the i^th sub-tasks onto CE_j. A validated co-reservation scheme should satisfy that if $S_{i,j} = 1$ then $c_j \ge r_i$.

$$\begin{array}{c|ccccc|} & CE_1 & CE_2 & CE_3 & \cdots & CE_N \\ r_1 & 1 & 0 & 0 & \cdots & 1 \\ r_2 & 1 & 1 & 0 & \cdots & 0 \\ r_3 & 0 & 0 & 1 & \cdots & 1 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ r_m & 0 & 1 & 1 & \cdots & 0 \end{array}$$

Figure 2 Co- reservation Matrix

So, the admission algorithm of co-reservation for PSA is designed as following. In this admission algorithm, relaxed reservation is applied only when there is no feasible time-slot that can rigidly meet the requirements of reservation request.

Algorithm 3: Co-reservation Admission for PSA

```
  Input: $\mathbf{R} =< sr_1, sr_2, \ldots, sr_m >$, $\{Slot_1, Slot_2 \ldots Slot_N\}$
  Output:
     Matrix of Co-reservation Scheme: $V$
     Successful probability of $V$ : $G$
Begin
1.  $G \leftarrow 1$;
2.  For $i \leftarrow 1$ to $m$
3.      If $\exists Slot_k \in Slot\_Set$ satisfying $\Pr\{\mathbf{E_i}\} == 1$
        Then $V_{i,k} \leftarrow 1$;
4.      Else
5.          $Max\_\Pr\{\mathbf{E_i}\} \leftarrow 0$;
6.          For each $Slot_k \in Slot\_Set$
7.              Obtaining $set_i^s$ and $set_i^e$;
8.              Calculating $\Pr\{\mathbf{E_i^S}\}$ by calling Algorithm 1;
9.              Calculating $\Pr\{\mathbf{E_i^E}\}$ by calling Algorithm 2;
10.             If $Max\_\Pr\{\mathbf{E_i}\} < \Pr\{\mathbf{E_i^S}\} \cdot \Pr\{\mathbf{E_i^E}\}$ Then
11.                 $Max\_\Pr\{\mathbf{E_i}\} \leftarrow \Pr\{\mathbf{E_i^S}\} \cdot \Pr\{\mathbf{E_i^E}\}$;
12.         End For
13.         $V_{i,k} \leftarrow 1$;
14.         $G \leftarrow G \times Max\_\Pr\{\mathbf{E_i}\}$;
15. End For
16. Return $V$ and $G$;
End
```

V. Experiments Evaluation and Analysis

In this section, we conduct extensive experiments to verify the performance of relaxed reservation policy when co-reservation for multiple resources is required.

A. *Experimental Settings*

Experimental grid model is a multi-cluster system, which consists of 12 high-performance clusters. The topology and detail settings of the grid model are referenced from the famous grid test-bed DAS-2 [22].

The experimental workload is generated by Lublin-Feitelson Model [23], which is derived from real workload logs of large-scale distributed systems. The workload consists of 10 000 requests, and each of request is characterized by arrival time, number of processors, and running time. As the workload 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 in this workload, so we append each request with a reservation start time, which is set by adding the arrival time with a random value that uniformly distributed in $[100,1000]$.

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 V.D). To increase the probability of co-reservation from multiple clusters, the resource demands of each reservation request is enlarged by 20 times.

In the first experiment, the performance of TDRRP is compared with four reservation policies, including Conventional Reservation Policy (CRP), Aggressive Backfilling Reservation Policy (ABRP), Conservative Backfilling Reservation Policy (CBRP), and Malleable Reservation Policy (MRP). As these policies have been applied in different systems, we present their brief description as following:

- *CRP* [9]: It is a fixed-capability reservation policy, in which request admission follows the definition 3.
- *ABRP* [13]: It allows the system move some requests ahead to fill the holes in the time-slot table, only when the first request in the waiting queue will not be delayed by them.
- *CBRP* [13]: It allows the system move some requests ahead to fill the holes in the time-slot table, provided that they do not delay any request in the waiting queue.
- *MRP* [24]: the requirements of a reservation request are allowed to be modified during the runtime. In practice, MRP is often implemented by adding other attributes in a reservation request, so that RM can re-arrange the time-slot table when necessary.

As mentioned above, reservation rate of workload is an important factor that has a great deal of effects on the performance of reservation service. So, we conduct the experiment four times under different reservation rates, i.e. 10%, 15%, 20%, 25%. In this experiment, we set $v_* = 0.8$ for TDRRP, which means that system 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 reservation time with mean value 35%. In next experiment, we will investigate the effects of v_* and k_{over} on the performance of TDRRP in more details. In this experiment, we focus on three performance metrics, including *Resource Utilization Rate*, *Reservation Rejection Rate*, and *Mean Response Time*.

B. Resource Utilization and Rejection Rate

In simulations, we set that the requests without reservation requirements will be canceled, if its deadline expires and still has not been scheduled on required resources. It often happens on the requests with co-reservation requirement, which in turn intensifies the decline of resource utilization. The experimental results are shown in Fig 3 – Fig. 6.

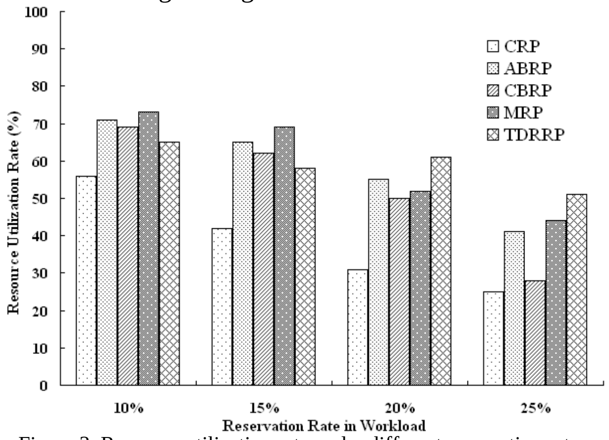


Figure 3. Resource utilization rate under different reservation rates

As we can see in Fig. 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. So, the performance of CRP is the worst in the five policies in terms of utilization rate.

When applying backfilling technique, utilization rate have been increased significantly about 14% comparing with CRP when reservation rate is about 10% and 15%. Respect to the two backfilling-based policies, ABRP outperforms over CBRP in different reservation rate. Especially, when the reservation rate is 25%, the resource utilization of ABRP is higher than ABRP about 13%. The reason is that ABRP tends to move more requests to the holes in time-slot table, which inevitably improve the resource utilization.

In this experiment, a significant result is that ABRP, CBRP, and MRP all perform better than TDRRP when reservation rate is relative low (10% and 25%). However, as the reservation rate increase to 20% and 25%, TDRRP's resource utilization becomes the highest. An interesting finding is that TDRRP's utilization rate 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 TDRRP as reservation requests increases. However, such increasing can not be sustained when the reservation rate increases to 25%.

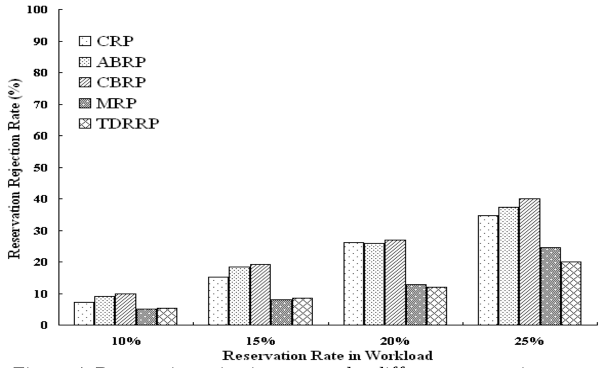


Figure 4. Reservation rejection rate under different reservation rates

Reservation rejection rate is shown in Fig. 4. Like the resources utilization, when using CRP, the rejection rate increases sharply from about 7% to 34% as the reservation rate increases from 10% to 25%. Also, we notice that backfilling technique can not lower down rejection rate. On the contrary, it leads to a slightly higher rejection rate comparing with conventional policy. That is because the experimental workload is very long (10 000 requests), the scheduler 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 TDRRP. That is why backfilling technique leads a higher rejection rate. When TDRRP is used, the rejection rate is only about 50% of conventional reservation policy in all cases.The results also show that the rejection rate of MRP is almost the same as that of TDRRP when reservation rate is 10%, 15%, and 20%. Only when it reaches 25%, MRP's rejection rate becomes higher than TDRRP's about 3%.

The conclusions of this experiment: (1) conventional reservation policy as a fixed-capability mechanism can not provide satisfying resource utilization for dynamic grid environment; (2) backfilling technique and malleable policy are effective to improve resource utilization when reservation rate is below 15%; (3) when the system faces high reservation rate (>15%), TDRRP is more effective than other policies.

C. *Task Mean Response Time*

As noted in [4, 7, 9-12], reservation service will result in the increasing of *Task Mean Response Time* (TMRT). It is because that reservation requests may hold their resources even if they are not used in reality. This situation often happens when too many reservation request arrivals in a short term. So, we log the TMRT of all the jobs in workload. As the simulative workload is very long (including 10 000 jobs), we monitor the TMRT in the manner of real-time when every 200 jobs arrivals. The real-time MTRT is shown in Fig. 5, and the total MTRT is shown in Fig. 6.

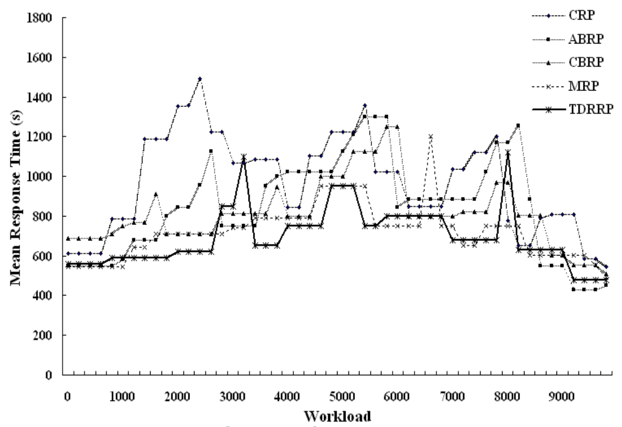


Figure 5. Real-time Task Mean Response Time

As shown in Fig. 6, the *Task Mean Response Time* of CRP, ABRP and CBRP are almost the same when the system's reservation rate is relative low (10% and 15%). On the other side, the *Task Mean Response Time* of MRP and TDRRP are very similar too. The dramatically differences occur when the reservation rate reaches 20% and 25%. For instance, the *Task Mean Response Time* of CRP increases about 230% when reservation rate increases from 10% to 25%. However, the *Task Mean Response Time* of TDRRP only increases about 50% in presence of 25% reservation rate.

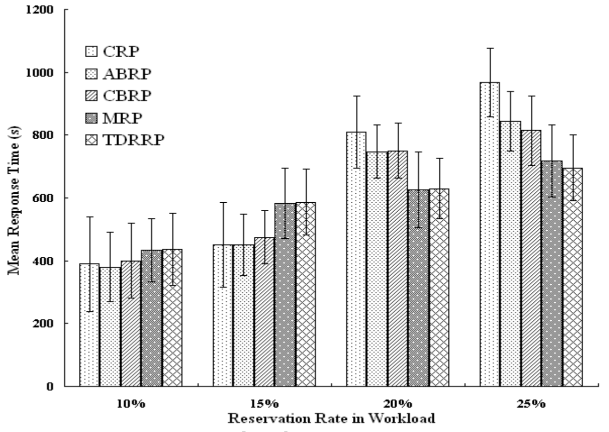


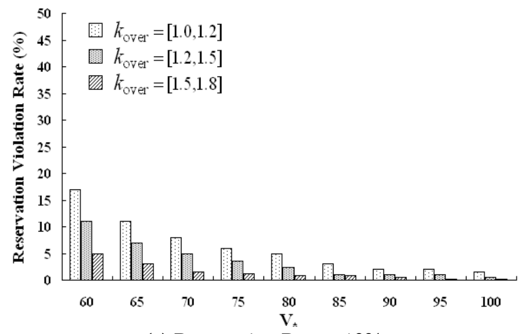
Figure 6. Total Task Mean Response Time

Refer to the results shown in Fig. 5, we notice that the *Task Mean Response Time* of CRP increases quickly even at the beginning of simulation. The reason is that CRP tends to block reservation requests when it can't meet its requirements. In our simulative workload, the distribution of reservation requests rate decreased gradually. So, CRP have to faces high reservation rate at the beginning of experiment. However, MRP and TDRRP allow the system accepting reservation requests with more flexible and relaxed criteria, so, they do not block too many reservation requests at the beginning. In the middle of experiment, as many reserved resource have been released earlier that their previous requirements, then MRP and TDRRP can take advantages of these released resources for those requests that been blocked by CRP.

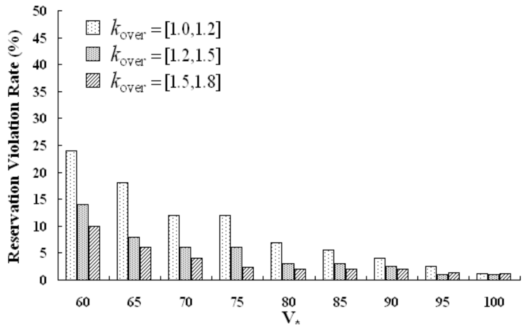
D. *Parameter Analysis for TDRRP*

In this experiment, we focus on reservation violation when using TDRRP. As noted in previous sections, there are two key parameters (v_* and k_{over}) in TDRRP. The parameter v_* is a threshold for admission algorithm in TDRRP. For instance, if the system set $v_* = 0.8$, it means that the system will accept a reservation request only when the probability of reservation violation for this request is less than 20%. As to the parameter k_{over}, it is a statistical characteristic of workload, which reflect the overestimation of reservation time. In this experiment, we modify the workload by multiple reservation time of each request with k_{over}, where k_{over} is set to be uniformly distributed in a certain interval. For instance, if k_{over} is set to be uniformly distributed in [1.0, 1.2], it means that requests in workload tend to overestimate their reservation time with mean value 10%, and [1.2, 1.5] means the mean overestimation is about 35%.

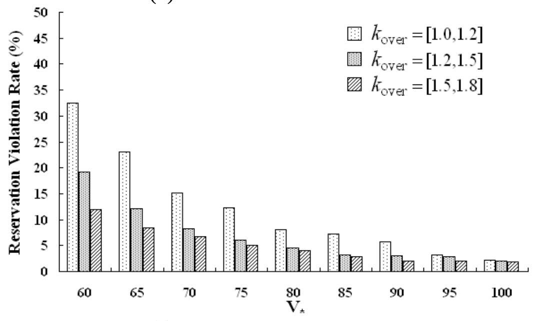
Just like the first experiment, we conduct this experiment four times with different reservation rates. In each experiment, parameter v_* is increased from 0.6 to 0.95 gradually, and parameter k_{over} is set to be uniformly distributed in three different interval, such as [1.0, 1.2], [1.2, 1.5], and [1.5, 1.8] respectively. In this way, we hope to extensively investigated the effects of v_* and k_{over} on the performance of TDRRP. The results of this experiment are shown in Fig. 7. In order to examine the effects of overestimation on resources utlization, we record all the resources utlization on different composion of k_{over} and v_* as shown in Table 1.



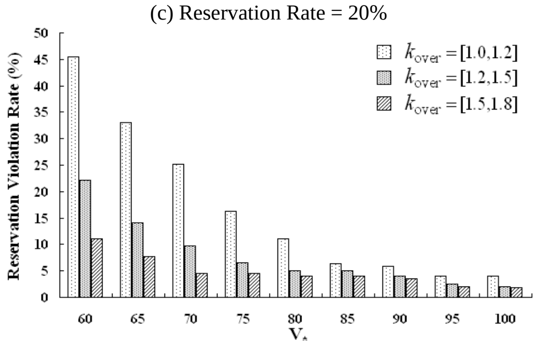
(a) Reservation Rate = 10%



(b) Reservation Rate = 15%



(c) Reservation Rate = 20%



(d) Reservation Rate = 25%

Figure 7. Reservation Violation Rate with different reservation rate

It is clearly that parameter v_* is of significant importance on the reservation violation rate. With the increasing of v_*, reservation violation rate decreases quickly, especially for those workload with very higher reservation rate (shown in Fig. 7 (c) and (d)). However, the decreasing of violation rate is not ratio to the increasing of v_*, more specifically, violation rate drops quickly when v_* is increased from 0.6 to 0.8, however, the decline of violation rate becomes stable even the parameter v_* is set to a very high value. In all cases, we note that the violation can be limited below 10% when $v_* \geq 0.8$, and If we set $v_* \geq 0.9$, the violation rate can be controlled below 5%. As to the resources utilization, we notice that higher overestimation often results lower utilization rate as shown in Table 1. However, parameter v_* is very effective to control the resources utilization, that is a small value of v_* can sigficantly increase resources utilization. For instance, when v_* is set to 60% the resources utilization is more that 70% even the overestimation has reached 35%, however, it drops to about 25%, when v_* is set to 100%.

Table 1 Resources Utilization with different k_{over} and v_*

v_* \ k_{over}	**[1.0, 1.2]**	**[1.2, 1.5]**	**[1.5, 1.8]**
60%	89.35%	84.58%	70.35%
65%	74.11%	70.21%	64.23%
70%	70.23%	64.53%	62.45%
75%	70.24%	63.32%	53.21%
80%	70.13%	63.47%	50.37%
85%	55.78%	51.02%	38.24%
90%	53.39%	50.69%	31.27%
95%	51.67%	55.34%	31.29%
100%	50.14%	31.25%	25.39%

As to parameter k_{over}, the experimental results indicate that more overestimation of reservation leads to lower reservation violation rate. hat is because many overlapped reservations do not overlap actually in run time, which makes TDRRP more effective. On the other side, the effects of k_{over} on violation rate are influenced by the reservation rate. As shown in Fig. 7 (a) and (b), the violation rate is limited in a relative lower level even the parameter v_* is set to be 0.6-0.8. However, if the reservation rate reaches 25% (as shown in Fig. 7(d)), the violation rate increases dramatically to about 45% if v_* is set to be 0.6. It is clear that a low value of v_* is not a good idea when a system is in presence of high reservation rate (>20%).

VI. Conclusion

To mitigate those effects, the paper proposes a two-demension relaxed reservation policy TDRRP, which based on the fact that applications tend to overestimate their running time to ensure their completion. Extensive simulations are conducted to verify the effectiveness of our policy. Experimental results show that TDRRP 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. For the future work, we plan to provide an adaptive mechanism for RM to dynamically set optimal v_* based on resource's runtime load. Also, we plan to incorporate TDRRP into grid economy to provide a more flexible price mechanism.

ACKNOWLEDGMENT

The authors wish to thank Dr. Buyya and his team for providing the open source grid simulator GridSim, and thank Prof. Lublin and Prof. Feitelson for their workload generator. This work was supported by a grant from the National Natural Science Foundation of China (No. 60673165 and No. 60970038).

REFERENCES

[1] A. Roy, V. Sander. “Advance Reservation API”. GFD-E.5, Scheduling Working Group, Global Grid Forum, 2002.
[2] I. Foster, I. Kesselman. “The Grid: Blueprint for a New Computing Infrastructure”. Singapore: Elsevier Inc. 2004.
[3] I. Foster, C. Kesselman, C. Lee, et al. “A Distributed Resource Management Architecture that Supports Advance Reservation and Co-Allocation”. Proc. of Int’l Workshop on Quality of Service, 1999:27-36.
[4] I. Foster, A. Roy, V. Sander. “A Quality of Service Architecture that Combines Resource Reservation and Application Adaptation”. Proceedings of International Workshop on Quality of Service, 2000:181-188.
[5] T. Rölitz, A. Reinefeld. “Co-Reservation with the Concept of Virtual Resources”. Proceeding of International Symp. on Cluster Computing and the Grid, 2005:254-261.
[6] L. O. Burchard, B. Linnert, J. Schneider. “A Distributed Load-Based Failure Recovery Mechanism for Advance Reservation Environments”. Proc of Int’l Symp. on Cluster Computing and the Grid, 2005:1071-1078.
[7] J. W. Cao, F. Zimmermann. “Queue Scheduling and Advance Reservations with COSY”. Proc. of the Int’l Parallel and Distributed Processing Symposium, 2004.
[8] A. C. Sodan, C. Doshi, L. Barsanti, D. Taylor. “Gang Scheduling and Adaptive Resource Allocation to Mitigate Advance Reservation Impact”. Proc. of Int.l Symp. on Cluster Computing and the Grid, 2006:649-653.
[9] W. Smith, I. Foster, V. Taylor. “Scheduling with Advanced Reservations”. Proceedings of International Symp. on Parallel and Distributed Processing, 2000:127-132.
[10] M. Wu, X. H. Sun, Y. Chen. “QoS Oriented Resource Reservation in Shared Environments ”. Proc. of Int.l Symp. on Cluster Computing and the Grid, 2006:601-608.
[11] C. Barz, M. Pilz, A. Wichmann. “Temporal Routing Metrics for Networks with Advance Reservations”. Proceedings of IEEE International Symp. on Cluster Computing and the Grid, 2008:710-715.
[12] A. Sulistio, K. H. Kim, R. Buyya. “Managing Cancellations and No-shows of Reservations with Overbooking to Increase Resource Revenue”. Proc. of Int.l Symp. on Cluster Computing and the Grid, 2008:267-276.
[13] A. W. Mu'alem, D. G. Feitelson. “Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling”. IEEE Trans. on Parallel and Distributed Systems, 2001, 12(5):529-543.
[14] A. Sulistio, W. Schiffmann, R. Buyya. “Advanced Reservation-based Scheduling of Task Graphs on Clusters”. Proceedings of International Conf. on High Performance Computing, 2006:60-71.
[15] N. R. Kaushik, S. M. Figueira, S. A. Chiappari. “Flexible Time-Windows for Advance Reservation Scheduling”. Proc. of Int.l Symp. on Modeling, Analysis, and Simulation of Computer and Tele. Systems, 2006:218-225.
[16] L. Wu, C. Wu, J. Cui, J. Xing. “An Adaptive Advance Reservation Mechanism for Grid Computing”. Proceedings of International Conf. on Parallel and Distributed Computing, Applications and Technologies, 2005:400-403.
[17] M. H. Alsuwaiyel. “Algorithms Design Techniques and Analysis”. U.S.A: World Scientific Publishing Co., 1999.
[18] F. Berman, R. Wolski, et al. “Adaptive Computing on the Grid Using AppLeS”. IEEE Trans. on Parallel and Distributed Systems, vol.14, no.4, 2003:369-382.
[19] H. Casanova, F. Berman. “Parameter Sweeps on the Grid with APST”. Wiley Publishers, Inc., 2002.
[20] A. Al-Saidi, N. J. Avis, I. J. Grimstead, O. F. Rana. “Distributed Collaborative Visualization Using Light Field Rendering”. Proc of Intl Symp. on Cluster Computing and the Grid, 2009:609-614.
[21] N. Jacq, J. Salzemann, F. Jacq, Y. Legré, E. Medernach, et al. “Grid-enabled Virtual Screening Against Malaria”. Journal of Grid Computing, vol.6, no.1, 2008:29-43.
[22] H. Bal, R. R. Bhoedjang, R. Hofman, et al. “The Distributed ASCI Supercomputer Project”. ACM Operating Systems Review, vol.34, no.4, 2000:76-96.
[23] U. Lublin, D. G. Feitelson. “The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs”. Journal of Parallel and Distributed Computing, vol.63, no.11, 2003:1105-1122.
[24] C. M. Hu, J. P. Huai, T. Y. Wo. “Flexible Resource Capacity Reservation Mechanism for Service Grid Using Slack Time”. Journal of Computer Research and Development, vol.44, no.1, 2007:20-28.

Peng Xiao was born in 1979. He received his master degree in Xiamen Universy in 2004. Now, he works in Hunan Institute of Engineering and is a Ph.D candidate in Central South University. His research interests include grid computing, parallel and distributed systems, network computing, distributed intelligence.

Zhigang Hu was born 1963. He is a Professor and Ph.D supervisor in Central South University. Currently, he is the Chief Secretary of Computer Science in Hunan Province. His research interests including grid computing, embedded systems, high-performance platform.