Impact of Adaptive Resource Allocation Requests in Utility Cluster Computing Environments

Marco A. S. Netto Rajkumar Buyya

Grid Computing and Distributed Systems Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
{netto, raj}@csse.unimelb.edu.au

Abstract

Maximizing resource provider profit and satisfying user requirements at the same time is a challenging problem in utility computing environments. In this paper, we introduce adaptive resource allocation requests and investigate the impact of using them in utility cluster computing environments. The Service Level Agreements established between users and resource providers rely not only on fixed values, but also on functions that associate allocation parameters. In addition, the resource provider scheduler can automatically modify the number of resources and usage time of allocation requests, as well as split them into subrequests. Users may receive incentives for supplying flexible requests which produce more scheduling options. By using rescheduling, resource providers are able to prioritize the most profitable requests dynamically and still satisfy the requirements of the already accepted user requests. From our experimental results we observed an increase of 14% in the resource provider profit and a reduction of 20% in the average response time of user requests when compared to traditional approaches.

1. Introduction

Organizations interested in providing computing resources and infrastructure management to customers on demand, charging them according to their usage, rely on the utility computing model [8]. In this model, computing resources are provided as services in the same way as other utility services, such as electrical power, water, and Internet access. For users, the most important benefit of this model is the reduction of costs and complexities regarding computing resources and infrastructure management. Companies can bring more agility and flexibility to their business with the ability to decide when, from which provider, and how much resource they wish to use without having to make large investments. Moreover, utility computing enables small companies to take on projects requiring IT-related services far beyond what their budgets would normally allow. Additionally, for resource providers, utility computing is a valuable model for increasing their profits. As the resources and infrastructure are not available and configured to a single user solution, it is easier to provide services for a larger variety of customers. Consequently, resource providers can reduce operational costs and underutilized resources.

Scheduling the incoming user requests in such a way as to satisfy user Quality of Service and maximize resource provider profit is a challenging problem. One of the main reasons is that usually the objectives of the resource providers and users are in conflict. When the resource provider receives an allocation request, its scheduler must find a free timeslot in its queue of requests according to some optimization criteria. The user requests are usually composed of parameters such as the number of resources, usage time, deadline, and the cost for satisfying all these parameters. The use of rigid allocation parameters reduces the number of mapping possibilities the scheduler can consider.

In this paper, we introduce adaptive resource allocation requests for utility computing environments, in particular for cluster machines, and show their impact for resource provider profit and user response time. In these environments, guarantees are important since users are paying to access the resources. These guarantees are established through Service Level Agreements (SLAs), which are contracts between users and resource providers to specify the required Quality of Service parameters. By including flexibility in these contracts, through adaptive requests and rescheduling techniques, resource providers are able to improve their profit and reduce the average response time of user requests. This is possible because resource providers

can have more scheduling options; the scheduler can automatically modify the amount of time and number of resources allocated to each request, as well as split them into smaller subrequests. With these operations, instead of only finding free timeslots in the request queue, the scheduler is able to free new slots by modifying the existing requests. Moreover, as the scheduling queue is modified over time, it is easier for the scheduler to adapt the request without user interaction.

As a way to attract more users, the resource provider may offer incentives to users supplying requests that support more scheduling options. The resource provider can prioritize the most profitable requests dynamically, by using rescheduling, and still satisfy the requirements of the already accepted user requests. We develop the scheduler to work with these adaptive requests and analyze the impact on the resource provider profit, the reduction of average request completion time and the rejection of requests. In this work we have considered *on-line scheduling* of requests with *hard deadlines*.

In the next section we present some of the existing works on job scheduling in utility computing environments. In Section 3 we define adaptive resource allocation requests, present two scheduling algorithms used for an initial evaluation of these requests, and discuss about cost functions. In Section 4 we show the impact of using the adaptive requests through simulations with different workloads and input parameters. Finally, in Section 5 we present some concluding remarks and plans for extending our work.

2 Related Work

Resource allocation and job scheduling are very well researched topics in Computer Science [7]. A number of resource management and scheduler systems, such as PBS [1], Condor [11], and MAUI [4], have been developed with their own specific purposes. These systems rely on static request parameters for allocating resources. Moreover, although these systems can be used in utility computing environments, they do not provide incentive mechanisms for increasing resource provider profit.

The use of economic models and market-based policies for resource allocation is becoming more popular. Several projects make use of these models and policies to increase resource provider profit while satisfying user required quality of service (QoS) [12]. Libra [10] is market-based scheduler that can be integrated into existing cluster resource managers to support resource allocation based on users' QoS. Libra makes use of pricing functions to assist the resource providers in determining the value of their resources according to parameters such as the current system workload. Incentive is given to encourage users to submit jobs with longer deadlines. Chun and Culler [2] discuss market-based cluster batch scheduling algorithms. They present a performance analysis of market-based batch schedulers using user-centric performance metrics. The cost charged to the users depends on the completion time of their jobs and decreases linearly over time until it reaches zero. Their work makes use of only one type of function and they do not explore the modification of request parameters; only deadline and budget.

Kalé et al [6] present an adaptive job scheduler for resource allocation of malleable jobs. The goal is to maximize the utilization of the system and reduce job response time. In their scheduler, when a high profit request arrives and has a tight deadline, low priority requests can be shrunk, releasing processors to be allocated to high priority request [5]. Their work focuses on the use of minimum and maximum values of number of resources. In addition, they do not consider splitting a request into smaller subrequests to assist in the scheduling process.

The use of adaptive resource allocation requests in utility computing environments has not been well explored. In this paper we will address this providing a detailed definition of adaptive requests and discussing the impact of these requests in utility computing environments.

3 Adaptive Resource Allocation Requests

We investigate the use of adaptive resource allocation requests in utility cluster computing environments. Therefore, *resources* here are defined as cluster nodes managed by Resource Management Systems (RMSs). These systems contain schedulers responsible for receiving the user requests and placing them into a waiting queue according to the available timeslots. The scheduler must decide which request can receive resources and at what time.

In this work we consider an *adaptive request* as the one that can be modified in terms of not only the number of resources and usage time, but also the starting and completion time that are not fixed when the request has been scheduled. Instead of using only rigid allocation parameters, the RMSs can rely on flexible parameters defined through functions that associate user requirements. This flexibility brings benefits for both users and resource providers. Following are some of the benefits of using adaptive requests:

- *Use of free fragments:* Due to the dynamic nature of the computing environment, the scheduling queue varies over time. While a user request is waiting for resources, other users may wish to modify or even cancel a request. In addition, it is common requests finishing before allocated time due to wrong estimation of execution times. These modifications produce timeslot fragments in the scheduling queue. Adapting user requests to fulfil these fragments demands user request

renegotiation. In this case, user-oriented renegotiation is avoided since the scheduler automatically adapts the requests without user interaction.

- *Creating new free timeslots:* Instead of only finding free timeslots, the scheduler attempts to create new ones by modifying the already scheduled requests. This flexibility is a mechanism for automatically reducing the completion time of user requests and increasing the resource provider profit.
- *Rescheduling the requests:* As the Service Level Agreements are flexible, the resource provider can reschedule the user requests without reducing the quality of service that users are expecting to receive.

3.1 Request definition

Usually a request in a utility computing environment is defined through the use of four parameters: (i) R, the number of resources; (ii) T, the resources usage time; (iii) D, the deadline (i.e. maximum time to finish the request); and (iv) B, the budget associated to this request—the maximum amount of money the user is willing to spend to use the resources.

In this work we extend this definition by considering two request modifications: (i) changing the number of resources and usage time (*moldable requests*), and (ii) splitting a request into smaller subrequests (*preemptive requests*). Each one of these modifications has an associated cost defined by the resource provider. Furthermore, these modifications can be executed together. For example, after splitting a request, one or more of these subrequests can be modified to access fewer resources for more time.

These modifications, together with the rescheduling, enable automatic resource negotiation among user requests. We illustrate an example through Figure 1: let a user, with a moldable and preemptive request Req_C, who needs the results of his/her application as soon as possible and is willing to pay for this. The scheduler can modify the other requests to reduce the finish time of the request Req_C. The figure shows an initial state where the scheduler has two candidates to analyze, the requests Req_A and Req_B, and five possible scheduling results.

Moldable requests. The relation between the number of resources and request duration is defined by the user application behavior. Therefore, the users should be responsible for deciding the number of resources required, and for how long they are needed. This is because in a real scenario the resource providers may not know the performance model of the user application. We define this $moldability$ parameter as a function f_{mol} that has as input the amount of resources R and the time T required to use these resources, $f_{mol} : R \rightarrow T$. The parameter f_{mol} defines the

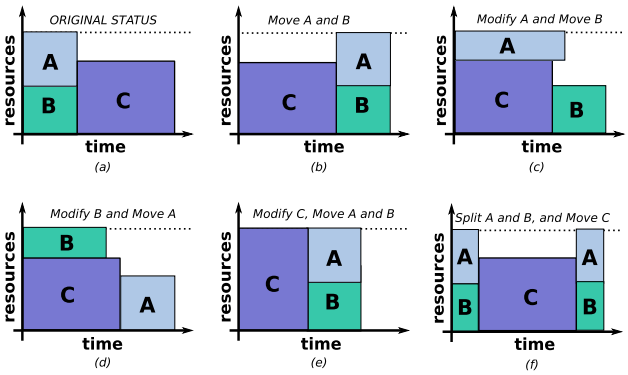

Figure 1. Five possibilities for reducing the completion time of the request C.

possible *shapes* of a request based on speed up functions. The user can define these functions through benchmarks and application profiling as discussed by Sevcik [9]. There is also existing work on characterizing moldable jobs [3]. Besides the function f_{mol}, the user needs to define R^{min} and R^{max}, which are the minimum and maximum number of resources, respectively. Thus, the scheduler can use an $R \in \mathbb{N}$, $R^{min} \leq R \leq R^{max}$. The user may restrict the value R to R^{max} since the application does not scale properly after such a value or because the price to access a certain number of resources is very high. The value R^{min} can be restricted since the application takes long time to execute if $R \leq R^{min}$ or due to memory requirements. Moreover, the user can restrict the values that R can assume, such as $R = 2^x$, where $x \in \mathbb{N}$, due to factors such as algorithmic constraints [3].

Preemptive requests. Splitting a request into smaller subrequests is a profitable way for filling fragments in the request queue or making possible the reduction of the request completion time. Here, splitting means the possibility of interrupting and restart later an execution. We define the parameter Sp to deal with preemptive requests as follows:

$$Sp = (N, I, M)$$

The argument N defines the maximum number of subrequests that a request can be split: $N \in \mathbb{N}$ or $N = \infty$; I defines the time intervals a request can be split, i.e. the points where the request can be split in the time axis: $I \in \mathbb{N}$; and M specifies whether the moldability can be applied independently for each subrequest: $M = true|false$.

Depending on the user application, the execution can be interrupted at any time or only at same specific times (parameter I). These interruptions are possible due to several factors. For example, the application has a checkpointing support, or the application is composed of a set of parallel tasks that are executed in sequence. In this second case, every time a task finishes, the application could be preempted.

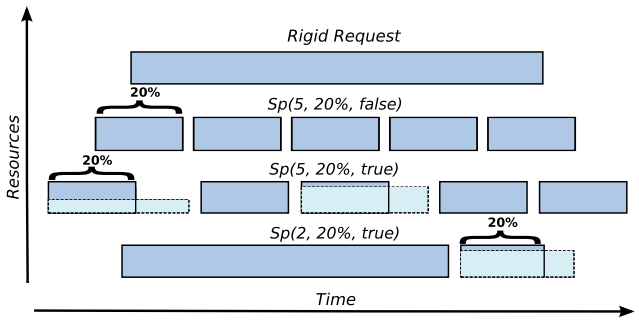

Figure 2. Usage examples of parameter Sp for preemptive requests.

In the current implementation of our scheduler, we allow these interruptions to be only at specific times according to a certain frequency. For example, the request can be split every 20% of the execution. The user may also restrict the number of times the scheduler can preempt a request due to performance reasons, e.g. high cost to execute a checkpointing (parameter N). Remark that there must be no overlaps between two subrequests of a request. This means that if a subrequest finishes at time t, the following subrequest must start after t.

In Figure 2 we present three usage examples of the parameter Sp for a given request. The first request can be split up to 5 subrequests that should be in the same size. The second request can be split up to 5 subrequests with the same initial size that can be modified independently. In the third case, the request can be split up to 2 subrequests with different initial size that can be modified independently. In this last example, although a subrequest can be created in each 20% of the request duration, only 2 subrequests can be created.

The splitting operation can bring benefits for both users and resource provider. However, it may not be easy to use this operation since it is highly dependent on the user application. Moreover, the user may need to setup additional configuration to work properly. For this reason, the resource provider could supply incentive mechanisms to the users in the form of resource usage discounts for example.

With these new two parameters, f_{mol} and Sp, an adaptive request Req is defined as:

$$Req(R^{min}, R^{max}, f_{mol}, D, B, Sp)$$

Note that the time T is not explicit present here, since it is defined in the function f_{mol}. If the request is not moldable the function f_{mol} should return a constant value.

3.2 Pricing functions

One common problem in utility computing environments is the resource usage pricing and the provision of incentive mechanisms. The resource provider can define the resource usage price P as a function that receives as input the request Req, the starting time T_s, resource usage duration, and number of resources R specified by the scheduler:

$$P = \omega(Req, T_s, Duration, R)$$

Note that the resource provider can classify users through different profiles and therefore there can be a different function for each user profile. In addition, depending on the current workload, the resource provider could adopt different profiles.

All the user parameters defined in Req are important for defining the price P. In particular, the parameters f_{mol} and Sp are an incentive mechanism for the users to supply flexible requests in order to assist the resource provider scheduler. Thus, the more *shapes* these parameters provide, the greater the discount for using the resources. We define this number of *shapes* as *flexibility factor*.

Internally, the resource provider deals with the ω function using a set of relations that associate the request parameters. These relations are: (i) Time $\Leftrightarrow$ Price; (ii) Deadline $\Leftrightarrow$ Price; (iii) Resources $\Leftrightarrow$ Price; and (iv) Flexibility Factor $\Leftrightarrow$ Discount. In Section 4 we provide an example of how to implement the ω function.

3.3 Scheduling

When scheduling user requests, ideally we want to maximize the resource providers' profit[1], reduce the *completion time* T_c of the users requests and serve as many requests as possible.

In this initial work, we implemented two simple scheduling strategies to compare the effectiveness of adaptive requests: (i) First in First Out algorithm; and (ii) Missing Deadline First; both with conservative backfilling. The main difference between these strategies is that the second one has a rescheduling phase. Here we consider *on-line scheduling*, i.e. the resource provider receives the requests and schedules them without the knowledge of the future requests. Furthermore, we consider requests with *hard deadlines*. If the resource provider is not able to schedule a request, the user can modify some allocation parameter and try a new request. The resource provider does not take the risk of accepting a request if it cannot be completed before the deadline.

[1] In this work we use the term *profit* as the total incoming from users. We are not considering that the resource providers have internal expenses to maintain infrastructure, such as power consumption, resource replacements, and IT staff.

Algorithm 1 presents how the scheduling of adaptive requests is currently performed. For preemptive requests, the scheduler split them into subrequests and deal with them independently; but as said in Section 3.1, no subrequest overlap is allowed. The *SchedulingEventTimes* variable in the algorithm is a list of the expected starting and completion times of the scheduled requests, and variable $Placement$ contains information of the request placement in the queue, such as, starting time, completion time and list of resources.

Algorithm 1 Pseudo-code for scheduling an adaptive request Req_i.

```
Best T_c ← null
BestPlacement ← null
for all et in the SchedulingEventTimes list do
  R ← R^max
  T ← f_mol(R)
  repeat
    Placement ← Allocate R resources for duration T
    if (Placement ≠ null and
        (Best T_c = null or Best T_c > T_c)) then
      BestPlacement ← Placement
      Best T_c ← T_c
    end if
    Decrement R, T ← f_mol(R)
  until R < R^min or Placement ≠ null
end for
if BestPlacement ≠ null then
  Allocate resources for Req_i using BestPlacement
  Update SchedulingEventTimes
else
  Not possible to schedule request Req_i
end if
```

The Missing Deadline First algorithm considers the rescheduling of the requests in the waiting queue. For every received request, all requests in the queue waiting for resources are sorted and rescheduled using Algorithm 1. In order to sort the requests, the scheduler considers the current time, the resource usage time T when for $R = R^{max}$, and the requests' deadline. It first places the requests close to missing their deadlines. Once a request receives the resources and starts execution, it moves to the queue of running requests (Figure 3). In this case, it is not rescheduled together with the requests waiting for resources.

The scheduler could attempt to optimize the profit each time a scheduling event occurs. Examples of events are: (i) the resource provider receives a new request; (ii) a request finishes before the expected deadline; and (iii) a user cancels or modifies a request. Moreover, it is possible that new resources become available or are removed from the environment. In these cases, the optimization procedure could also be executed. We plan to consider these events in future.

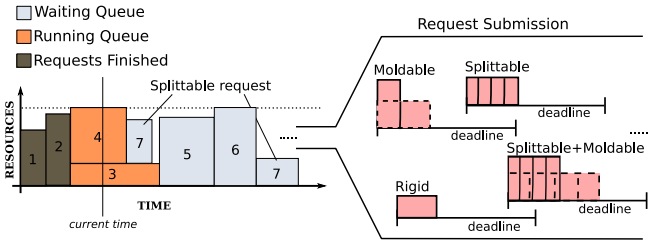

Figure 3. Scheduling queues.

4 Evaluation

We evaluated the use of adaptive resource allocation requests by simulations. We developed a software that implements the scheduling heuristics and utilizes the moldable and preemptive requests. The following sections present how we setup the environment and the results we obtained.

4.1 Environment setup

The setup of the experiments involves several variables, such as workload, pricing functions, user profiles, deadlines, budgets, and computing environment. Workloads that consider deadlines, budget, moldable and preemptive requests cannot be easily found. For this reason, we used real workload data and modified it according to our needs. We chose the workload of the IBM SP2 system, composed of 128 homogeneous processors, located at the San Diego Supercomputer Center (SDSC)[2]. This workload contains requests performed in an interval of two years. We conducted the experiments by splitting the workload in intervals of 15 days. We removed the requests whose duration was less than a minute. Hence we got a workload set composed of 48 files, each file with 244± 68 requests. The total number of requests was 11244. For each experiment, after collecting the data, we discarded the four best and four worst results to reduce the deviation.

As we mentioned in Section 3.1, a request is defined as $Req(R^{min}, R^{max}, f_{mol}, D, B, Sp)$. Considering these parameters, we only got from the workload the number of requested resources, defined as R^{max}, and the expected execution time, defined as T^{min}. We used the *submission time* from the workload to setup the frequency at which the requests were to be submitted. Table 1 shows the configuration of the other parameters we used to modify the workload files according to our needs. The 60% and 20% regarding the Sp and price profile parameters, respectively, were used randomly.

We considered that the cost of checkpointing/preempting a request is negligible in relation to the time of executing an

[2]We used the version 3.1 of the IBM SP2 - SDSC workload, available at: http://www.cs.huji.ac.il/labs/parallel/workload/logs.html.

Table 1. Request variable definitions.

Variable	Definition	Comment
R^{min}	80% of R^{max}	Requests with up to 10 resources have $R^{min} \leftarrow R^{max}$
T^{max}	$T^{min} * R^{max}/R^{min}$	Maximum time using minimum resources
f_{mol}	$k - c * R$	Moldability function—amount of work remains constant
$c\ (R^{max} \neq R^{min})$	$(T^{max} - T^{min})/(R^{max} - R^{min})$	Constant for the moldability function
$c\ (R^{max} = R^{min})$	0	Constant for the moldability function
$k\ (R^{max} \neq R^{min})$	$T^{min} + R^{max} * c$	Constant for the moldability function
$k\ (R^{max} = R^{min})$	T^{min}	Constant for the moldability function
$Sp(N, I, M)$	(10, 10%, true)	Requests less than 1 hour were not split
		Only 20% of the requests were considered as preemptive
B	$R^{max} * T^{min} * PriceProfile$	-
Price profile	1 (80%), 3 (20%)	We defined two categories of requests
		20% of the requests are highly profitable
$D\ (PriceProfile = 1)$	$T^{min} * DeadlineFactor + three\ days$	-
$D\ (PriceProfile = 3)$	$T^{min} * DeadlineFactor$	-
$DeadlineFactor$	5	Gives the additional time to delay the deadline
Incentive for preemp.	0%	
Resource usage price	1$ per machine hour	-

Algorithm 2 Pseudo-code for calculating the price of an allocation.

$T_c = T_s + Duration$
$PercentageBeforeDeadline = \frac{(T_c - T^{min})}{(D - T^{min})}$
$P = Duration * R$
$P = P - \lfloor (PercentageBeforeDeadline * P * 0.6) \rfloor$
$P = P * PriceProfile$
if Req is preemptive **then**
  return $P - P * incentive$
else
  return P
end if

application. Thus, this cost should not interfere with the results and hence we did not this take into account.

Algorithm 2 defines the ω function (Section 3.1) used by the scheduler to calculate the price of an allocation request. Note that the price associated with a request decreases according to its finish time. We allow this decrement to be up to 60% of the original user request budget.

After modifying the workload according to our needs, the simulator reads the workload file considering each request as a rectangle that must be fit in a Gantt chart according to the scheduling algorithm. These rectangles represent the number of resources and request duration. The budget, deadline, and the preemptive and moldability functions define the properties of these rectangles. Therefore, the speed of the simulated machine, i.e. the IBM SP2, and other environment details were not required to perform the experiments.

4.2 Experiments and results

Given the described environment set up, we first measured the impact of the adaptive requests by analyzing three metrics: (i) total resource provider profit; (ii) average request finish time; and (iii) number of rejected requests. We executed the experiments using four types of requests: (a) rigid; (b) preemptive; (c) moldable; and (d) preemptive plus moldable. We performed the experiments using the *Missing Deadline First* algorithm, which has rescheduling, and FIFO, which has no rescheduling.

As the amount of load that a resource provider must deal with affects the results, we performed the experiments by varying the frequency of request submissions. We increased the frequency of the original workload until 50%. For these experiments we consider no additional incentive for preemptive requests.

Figure 4 shows the percentage of profit gain compared to the conventional approach according to the workload. It is possible to observe that the maximum benefit on the original workload is only approximately 4.5% for moldable and preemptive plus moldable requests—both using the rescheduling approach. Without the rescheduling, the maximum benefit is only 2%. However, there is a considerable benefit when we increase the load of the resource provider and use the rescheduling approach. With the increased frequency of submissions of 50%, the use of moldable requests makes the profit increase approximately 3% compared to use of rigid requests, using rescheduling, and 14% compared to the traditional approach (FIFO with rigid requests).

In Figure 5 we can see a clear advantage of using

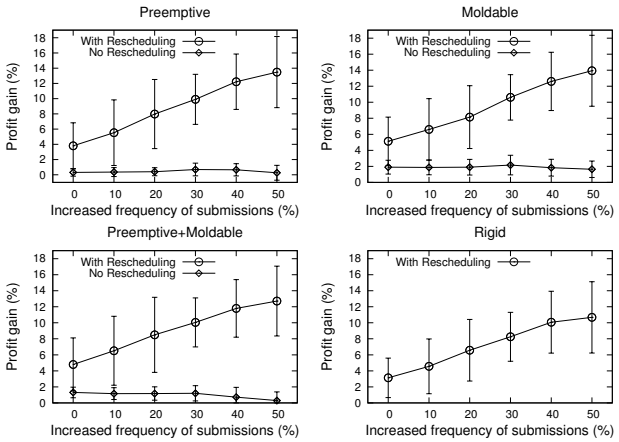

Figure 4. Profit gain according to cluster load.

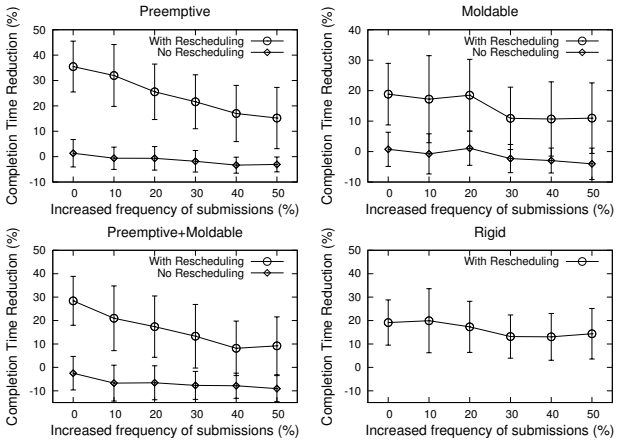

Figure 5. Average reduction of the request completion time according to cluster load.

rescheduling for reducing the completion time of user requests. In this case, the preemptive requests provide the best results. Note also that the there is a reduction of the benefit when we increase the cluster load. In addition, we can observe a considerable variation on the results for each different load. Therefore, the completion time reduction is highly dependent on the workload.

The number of rejected requests has a direct impact on the amount of profit the resource provider loses due to an inappropriate scheduling strategy. Furthermore, in a real environment, it is not of interest to a resource provider to reject a large number of *customers*, since reputation comes into play. From Figure 6 we observe that rescheduling plays an important role in preventing the rejection of requests.

In these experiments we considered no additional incentive for preemptive requests. We performed another experiment that consists of measuring the maximum incentive a

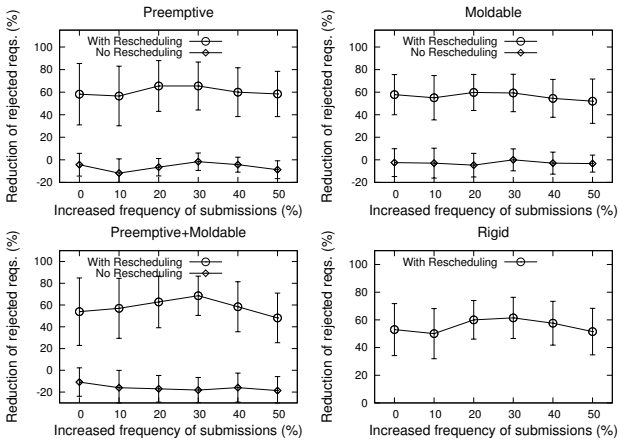

Figure 6. Reduction of rejected requests according to cluster load.

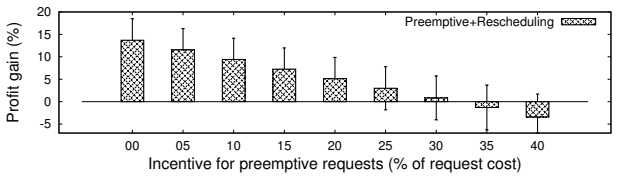

Figure 7. Resource provider profit according to the incentive for preemptive requests.

resource provider should give to those users who provide preemptive requests. We selected the best obtained result in the previous experiment, which is the rescheduling approach using preemptive requests on a workload with the increased submission rate of 50%. Then we varied the incentive from 0 to 40%. We can observe in Figure 7 that after 30% of the resource usage price the resource provider has a benefit of less than 1%, after that, in our experiments, the resource provider starts to lose profit. This value "30%" could be less if we added more preemptive requests in our workload. However, the important issue to consider here is that there a risk of choosing a wrong incentive value for preemptive requests. We will investigate in future how to choose appropriately a correct incentive value for preemptive requests. However, remark that there is already an implicit benefit for the users to supply preemptive requests since they reduce average finish time.

We also analyzed the characteristics of the accepted moldable requests. First we measured the percentage of moldable requests that used fewer resources than R^{max}, i.e. the number of moldable requests that the scheduler modified to improve the scheduling (Figure 8). Second we measured how much such requests were modified, e.g. if a request used R^{min} the scheduler modified 100%, if the re-

quest used R^{max} the scheduler has not modified the request (Figure 9). From these experiments, we observe that the scheduler tends to use the moldable requests more in highly utilized clusters. Moreover, when moldable requests can be split, the scheduler modifies more of the requests in order to bring them as close as possible to the current time.

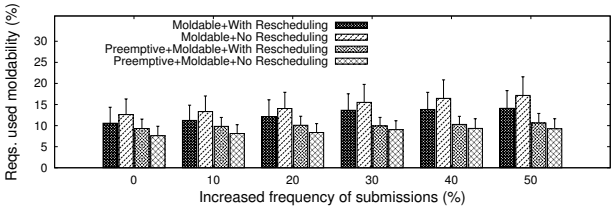

Figure 8. Moldable requests that used fewer resources than $\mathbf{R^{max}}$.

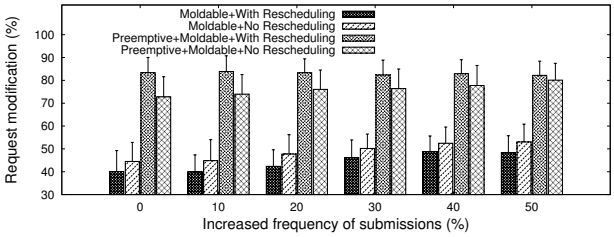

Figure 9. Moldable request modification.

5 Conclusion and Further work

In this paper we have presented a definition of adaptive resource allocation requests and their impact in utility cluster computing environments. The Service Level Agreements between users and resource providers should be flexible, in order to increase the resource provider profit and, at the same time, reduce the average response time of user requests.

We observed in our experiments that by making use of preemptive requests and the rescheduling technique, the resource provider was able to increase the profit by 14% and the average user response time was reduced by 20% in a highly utilized cluster environment. It is important to mention that these results are based on a subset of workloads. We will evaluate in different workloads to have more precisely conclusions.

As next steps we will mainly focus on developing better scheduling algorithms for these requests. We will explore cost functions that consider the current load of the resources and imprecision of user requirements. For example, defining precisely when the execution preemptions occur or when an application finishes in advance is difficult, and in some cases, not feasible. Therefore, handling imprecision is an important issue to deploy the scheduling algorithms in utility computing environments.

Acknowledgments

We would like to thank Kyong Hoon Kim, Hussein Gibbins, and the anonymous reviewers for their valuable comments.

References

[1] A. Bayucan. Portable Batch System Administration Guide, August 2000.

[2] B. Chun and D. Culler. User-centric performance analysis of market-based cluster batch schedulers. In *Proceedings of the 2nd IEEE/ACM Symposium on International Cluster Computing and the Grid*, pages 30–38, Berlin, Germany, 21-24 May 2002.

[3] W. Cirne and F. Berman. A model for moldable supercomputer jobs. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium*, page 59, 2001.

[4] D. B. Jackson, Q. Snell, and M. J. Clement. Core algorithms of the maui scheduler. In *Proceedings of the 7th International Workshop on Job Scheduling Strategies for Parallel Processing*, pages 87–102, Cambridge, USA, 16 Nov. 2001.

[5] L. Kalé, S. Kumar, M. Potnuru, J. DeSouza, and S. Bandhakavi. Faucets: efficient resource allocation on the computational grid. In *Proceedings of the 33rd International Conference on Parallel Processing*, volume 1, pages 396–405, Montreal, Canada, 2004.

[6] L. V. Kalé, S. Kumar, and J. DeSouza. A malleable-job system for timeshared parallel machines. In *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 230–237, 21-24 May 2002.

[7] M. Pinedo. *Scheduling: theory, algorithms, and systems*. Prentice Hall, Englewood Cliffs, N.J., 1995.

[8] M. A. Rappa. The utility business model and the future of computing services. *IBM Systems Journal*, 43(1):32–42, 2004.

[9] K. C. Sevcik. Characterizations of parallelism in applications and their use in scheduling. In *SIGMETRICS*, pages 171–180, 1989.

[10] J. Sherwani, N. Ali, N. Lotia, Z. Hayat, and R. Buyya. Libra: a computational economy-based job scheduling system for clusters. *Software: Practice and Experience*, 34(6):573–590, 2004.

[11] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[12] C. S. Yeo and R. Buyya. A taxonomy of market-based resource management systems for utility-driven cluster computing. *Software: Practice and Experience*, 36(13):1381–1419, 2006.