SLA-based advance reservations with flexible and adaptive time QoS parameters

Marco A. S. Netto[1], Kris Bubendorfer[2], Rajkumar Buyya[1]

[1] **Gr**id **C**omputing and **D**istributed **S**ystems (GRIDS) Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
ICT Building, 111 Barry Street, Carlton, VIC 3053
{netto, raj}@csse.unimelb.edu.au

[2] School of Mathematics Statistics and Computer Science
Victoria University of Wellington
Wellington 6140, New Zealand
kris@mcs.vuw.ac.nz

Abstract. Utility computing enables the use of computational resources and services by consumers with service obligations and expectations defined in Service Level Agreements (SLAs). Parallel applications and workflows can be executed across multiple sites to benefit from access to a wide range of resources and to respond to dynamic runtime requirements. A utility computing provider has the difficult role of ensuring that all current SLAs are provisioned, while concurrently forming new SLAs and providing multiple services to numerous consumers. Scheduling to satisfy SLAs can result in a low return from a provider's resources due to trading off Quality of Service (QoS) guarantees against utilisation. One technique is to employ advanced reservations so that a SLA aware scheduler can properly manage and schedule its resources. To improve utilisation we exploit the principle that some consumers will be more flexible than others in relation to the starting or completion time, and that we can juggle the execution schedule right up until each execution starts. In this paper we present a QoS scheduler that uses SLAs to efficiently schedule advanced reservations for computation services based on their flexibility. We present our SLA and scheduling algorithms, and show experimentally that it is possible to use flexible advanced reservations to meet specified QoS while improving resource utilisation.

1 Introduction

Service Level Agreements (SLAs) are an important element of the service oriented computing paradigm and define a mutually agreed upon set of consumer expectations and provider obligations. Typically SLAs encode Quality of Service (QoS) parameters such as resource availability, response time and completion deadlines. The role of the consumer is usually limited to specifying their QoS parameters and perhaps revising those parameters if an SLA cannot be agreed.

We assume a scenario where access to a utility computing provider's computational resources is acquired through agreed SLAs. The SLAs define the time and quantity of computation along with other QoS parameters, in return for a certain price. Access to computational resources may require consideration of *external* constraints, such as the need for access to simultaneous multiple resources (co-allocation for parallel computation) [1] or to reflect timing dependencies when computing a workflow. In order to meet such external constraints, a QoS scheduler must allow consumers to reserve resources in advance. We introduce the concept of a time flexible SLA to describe advance reservations. The various SLA parameters detail the timing and any potential timing flexibility along with more conventional QoS parameters.

When a provider accepts an advance reservation, the consumer expects to be able to access the agreed resources at the specified time. However, changes may occur in the scheduling queue between the time the consumer submits the reservation to the time the consumer receives the resources. There are a number of reasons for such changes including: consumers cancelling requests, consumers modifying requests, resource failures, and errors in estimating usage time in the consumer requests. Therefore, from the resource provider's perspective, a good time-slot for the consumer at the time the SLA was agreed may be a bad time-slot in the future due to increased fragmentation. This fragmentation reduces the potential scheduling opportunities and results in lower utilisation. Indeed, even finding a free time-slot can be a challenging task since fixed advance reservations fragment the resource's availability, and limit the positions in which other jobs can be scheduled. To solve this problem we suggest that some consumers will be more flexible than others in relation to the request starting or finishing time, and we can explore this to make better, flexible, scheduling decisions. Both consumers and resource providers can declare their flexibility in the SLA and the resource scheduler can continuously adapt the scheduling to maximise the resource utilisation under the SLA constraints.

In this paper we investigate the use and scheduling of advance reservations with time flexible SLA parameters in the context of a utility computing environment. We present algorithms for scheduling flexible advance reservations, and show that it is possible to use advance reservations to meet specified the QoS without suffering from poor utilisation of the provider's resources. Section 2 describes the SLA used to drive the scheduling decisions, section 3 presents the scheduling algorithms, section 4 presents the performance results derived via simulation, section 5 provides a discussion of related work, and finally section 6 concludes our work and describes our future work.

2 SLA Specification from Execution time QoS Scenarios

This section defines the set of parameters that we need in addition to any normal SLA parameters such as incentives and penalties, security or trust requirements, etc. Figure 1 shows three different time requirement scenarios.

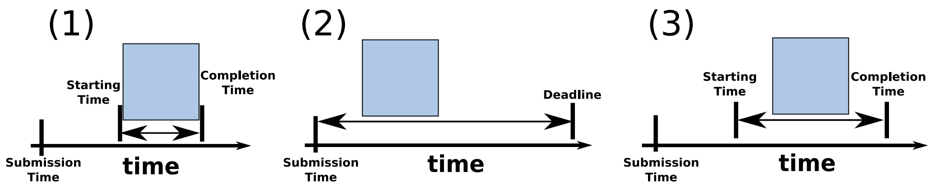

Fig. 1. Job time Qos requirements.

1. **Strict start and completion time:** Consumers require the resource at exactly this time, and for the duration specified. There is no flexibility permitted to the scheduler. This scenario maps well to the availability of a physical resource that may need to be booked for a specific period.
2. **Relaxed start time, strict completion time:** Consumers require that the execution completes prior to a deadline. This scenario typically applies when there are subsequent dependencies on the results of this computation.
3. **Flexible interval:** There is a strict start time and a defined finish time, but the time between these two points exceeds the length of the computation. This scenario fits well with forward and backward timing dependencies, such those encountered in a workflow computation.

2.1 Scheduling Issues

These cases as given above are simplistic; however scheduling them is complicated. Consider both cases 2 and 3, as the actual deadline approaches, the apparent priority of scheduling must increase to ensure that the execution completes prior to the deadline. Also early acceptance of SLA requests of type 1 will fragment the availability of the resource, which may result in wasted computation time, increased rejections, reduced utilisation and consequently reduced revenue.

The idea of having flexible intervals for advance reservations is to make possible modify or reallocate pre-existing advance reservations to fit other workload. As a side note, we would expect more flexible QoS requests to experience a lower price from the provider. The SLA itself is not flexible, but rather it encodes a set of time flexible QoS parameters. Once the SLA has been agreed upon, the scheduler may schedule the workload within those constraints, but the SLA itself cannot be changed without penalties applying.

2.2 SLA parameters

The advance reservations are defined in the SLA by a set of timing constraints, budget and computational resources. Following is the notation and parameter definitions for a job j, which can be either rigid or mouldable (parallelism versus execution time trade off):

– R_j^{min} and R_j^{max}, where $1 \leq R_j \leq m$: minimum and maximum number of resources (e.g. cluster nodes or bandwidth) required to execute the job;

- $f_j^{mol} : R_j \rightarrow T_j^e$: moldability function which specifies the relation between number of resources and execution time T_j^e;
- T_j^s: job starting time—time determined by the scheduler;
- T_j^r: job ready time—minimum starting time determined by the user;
- T_j^c: job completion time—defined as $T_j^s + T_j^e$;
- D_j: job deadline;
- B_j: job budget—maximum amount of money that the user is willing to spend to execute the job;
- C_j: job cost—the cost determined by the resource provider in order to execute the job j with the above specifications.

3 Job Scheduling

The scheduling of a job consists on finding a free time-slot that meets the job requirements. Rather than providing the user with the resource provider's scheduling queue, we assume that the user asks for a time-slot and the resource provider verifies its availability. This is sensible in competitive environments where resource providers do not want to show their workloads, as consumers and other resource providers may exploit this commercially sensitive information. We also consider the scheduling to be on-line, where users submit jobs to the resource provider's scheduler over time and the scheduler makes its decisions based only on the currently accepted jobs.

Scheduling takes place in two stages. Firstly all jobs that are currently awaiting execution on the machine (and therefore have accepted SLAs) are sorted based on some criteria. Then this list is scheduled in order, and if the new job can be scheduled, the SLA is accepted. If the job cannot be scheduled, then the scheduler can return a set of scheduleable alternative times.

3.1 Sorting

Firstly we separate the jobs currently allocated into two queues: running queue $Q^r = \{o_1, ..., o_u\} \mid u \in \mathbb{N}$ and waiting queue $Q^w = \{j_1, ..., j_n\} \mid n \in \mathbb{N}$. The first queue contains jobs already in execution and cannot be rescheduled. The second queue contains jobs that can be rescheduled. The approach we adopt here is to try to reschedule the jobs in the waiting queue by sorting them first and then attempting to create a new schedule. We use five different sorting techniques in this paper: Shuffle , First In First Out (FIFO), Biggest Job First (BJF), Least Flexible First (LFF), and Earliest Deadline First (EDF). The only sorting criteria that needs explanation is LFF, which sorts the jobs according to the flexibility terms of starting time and deadline. This approach is based on the work of Wu et al [2], but considers only the time intervals. We define the time flexibility of a job j as follows:

$$\Delta_j = \begin{cases} D_j - max(T_j^s, CT) - T_j^e : \textit{for advance reservation jobs} \\ D_j - CT - T_j^e : \textit{for jobs with deadline} \end{cases}$$

Obviously other potential criteria can be used to perform this sort, one that we will be exploring in the future is sorting based on expected revenue. In the evaluation Section 4 we present results comparing these sorting techniques.

3.2 Scheduling

Algorithm 1 gives the pseudo-code for scheduling a new job j_k at the current time CT. Before the scheduling of a new job, the state of the system is consistent, which means that the current scheduling of all jobs meets the users QoS requirements. Therefore, during the scheduling process, if a job j_i is rejected there are two options: (i) $j_i = j_k$, the new job could not be scheduled; or (ii) $j_i \neq j_k$, the new job was scheduled but generated a scheduling problem for another job $j_i \in Q^w$. In the second case we change the positions of j_k with j_i and all jobs between j_k and j_i go back to the original scheduling—function that we call $fixqueue$. In our current implementation, each job is scheduled by using first fit approach—the first available time-slot is assigned to the job. For jobs with deadline the scheduler looks for a time-slot between the interval $[CT, D_j - T_j^e]$ and for advance reservations the scheduler looks for a time-slot within the interval $[T_j^r, D_j - T_j^e]$.

Algorithm 1 Pseudo-code for scheduling a new job j_k.

```
Input: $j_k$
Output: $true$, $false$, list of options
$Q^w \leftarrow Q^w \bigcup \{j_k\}$
sort $Q^w$ according to some criteria (i.e. EDF or LFF)
$k \leftarrow$ new index of $j_k$
$jobscheduled \leftarrow true$
for $\forall j_i \in Q^w \mid i \geq k$ and $jobscheduled = true$ do
  if schedulejob $(j, Q^w, Q^r) = false$ then
    $jobscheduled \leftarrow false$
  end if
end for
if $jobscheduled = false$ then
  if $i \neq k$ then
    $fixqueue(Q^w, i, k)$ { update index of $j_k$ $(k \leftarrow i)$}
  end if
  return reschedule $\forall j_i \in Q^w \mid i \geq k$
end if
return $true$
```

When job j_k is rejected, all the jobs in Q^w after j_k, including j_k itself, must be rescheduled (Algorithm 2). However, in this rescheduling phase, other options are used to reschedule j_k. The list of options Ψ is generated based on the intersection of the new job j_k, the jobs in the running queue and the jobs in the waiting queue that are before j_k. For each job j_i that intersects j_k, job j_k is tested before T_i^s and after D_i. Once the list of options Ψ is generated, it is possible to sort it according to the percentage difference ϕ between the original T_j^r and D_j values and the alternative scheduler suggested options $OPTT_j^r$ and $OPTD_j$:

$$\phi_{opt} = \begin{cases} \frac{OPTD_j - D_j}{T^e_j} : \textit{option generated by placing } j_k \textit{ after } j_i \\ \frac{OPTT^r_j - T^r_j}{T^e_j} : \textit{option generated by placing } j_k \textit{ before } j_i \end{cases}$$

Algorithm 2 Pseudo-code for rescheduling rejected part of Q^w using the list of options Ψ for the rejected new job j_k.

```
$OT^r_k \leftarrow T^r_k$, $OD_k \leftarrow D_k$ {keep original values}
while $\forall OPT \in \Psi$ do
  $jobscheduled \leftarrow true$
  for $\forall j_i \in Q^w \mid i \geq k$ and $jobscheduled = true$ do
    if $j_i = j_k$ then
      set $T^r_k$ and $D_k$ with option $OPT$
    end if
    $jobscheduled \leftarrow$schedule($j_i$)
  end for
  if $jobscheduled = false$ then
    if $i \neq k$ then
      $fixqueue(Q^w, i, k)$
      $T^r_k \leftarrow OT^r_k$, $D_k \leftarrow OD_k$ {restore original values}
      return reschedule $\forall j_i \in Q^w \mid i \geq k$
    else
      return $false$ {already tested new options for $j_k$}
    end if
  else
    {valid option $OPT$ in $\Psi$—inform user about this possibility}
  end if
end while
if $\exists\ OPT \in \Psi \mid OPT$ generates a possible scheduling then
  return $true$
end if
return $false$
```

Once defined the possible positions of the new job j_k, all jobs in Q^w after j_k (including it) are rescheduled. If a job j_i is rejected, we have again two options: (i) $j_i = j_k$, the new job could not be scheduled; or (ii) $j_i \neq j_k$, the new job was scheduled but generated a scheduling problem for a another job $j_i \in Q^w$. In constract with Algorithm 1, in Algorithm 2, when $j_i = j_k$, it means that the scheduler has already tried all the possibilities to fit j_k in the queue, and hence, j_k will not be rescheduled again. However, if $j_k \neq j_i$, then the queue Q^w is fixed, the index of j_k is updated, T^r_k and D_k are set to the original values, and the rest of Q^w is again rescheduled. This process finishes when the scheduler has no more event times to test. For a consumer who does not require an advance reservation, the first successful option should be enough.

4 Evaluation

The basis for the design of the scheduling algorithms and the improvement in utilisation, is predicated on the idea that scheduling advance reservations with some specified flexibility will allow better scheduling decisions to be made. The experimental results in this section demonstrate that the principle is sound.

4.1 Experimental Configuration

We evaluated the use of flexible SLA parameters for advance reservation on an extended version of the PaJFit (Parallel Job Fit) [3] simulator. We used the workload trace from the IBM SP2 system, composed of 128 homogeneous processors, located at the San Diego Supercomputer Center (SDSC)[‡] as a realistic workload to drive the simulator. This workload contains requests performed over a period of two years. However, for reasons of tractability we conducted our experiments using 15 day intervals. We also removed any requests with a duration of less than one minute.

As the workload has no deadline specifications (and there are no traces with this information available), we modelled them as a function of the execution time. We observe that many workload distributions exhibit Poisson lifetimes and assume that this would also be true for deadlines. For each job j, $D_j = T^r_j + T^e_j * p$, where p a random number defined by a Poisson distribution with λ=5. As we are working with advance reservations, we defined the release time of jobs as $T^r_j = D_j - T^e_j$. To model higher loads and the subsequent performance of the scheduler, we increased the frequency of request submissions from the trace by 25% and 50%.

We also analysed four different flexible interval sizes, which we again define as a Poisson distribution: fixed interval, short interval ($\lambda \leftarrow \phi = 25\%$), medium interval ($\lambda \leftarrow \phi = 50\%$), long intervals ($\lambda \leftarrow \phi = 100\%$). For all experiments using flexible intervals, we modified only half of each workload, the other half continues to have fixed intervals. We believe a portion of users would continue to specify strict deadlines even though the resource provider would probably reduce the price for more flexible and therefore *easier* consumers.

4.2 Results and Analysis

For the first experiment we evaluated the importance of sorting the jobs in the waiting queue according to specific criteria. Figure 2 shows the results, comparing LFF, BJF (sorted by the job's size = $T^e * R$), EDF, and FIFO, against a random shuffle; all of them with backfilling strategy. The results are presented as the difference in utilisation from the random baseline. In all cases, EDF with flexible intervals produced a schedule with the highest utilisation. It is worth noting that the results are not load sensitive, shown as the load increases — from normal

[‡] 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.

(top graph) to high (bottom graph) in figure 2. As in our experiments we show comparative results, it is important to mention the system utilisation values to have an idea of the magnitude of these results. The values for the original workload and the two modifications on the frequency of request submissions, using FIFO approach, are: 46.8 ± 3.3 %, 50.9 ± 3.5 %, and 54.7 ± 3.7 %.

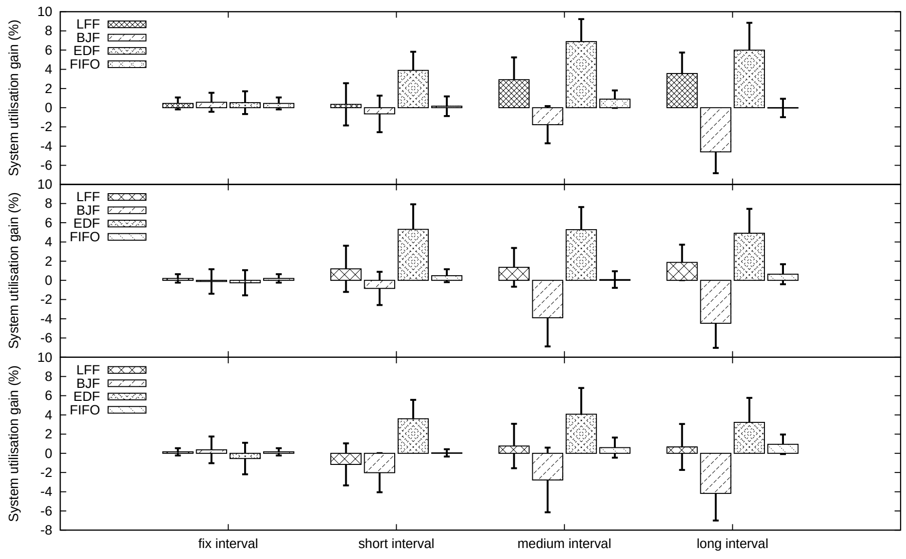

Fig. 2. Impact of sorting criteria on system utilisation

Using the EDF heuristic, we next evaluated the impact of the flexible time interval *duration* on resource utilisation (Figure 3). We observe that the longer the interval size, the higher the utilisation. This is because longer interval sizes provide the scheduler with more options for fitting (juggling) advance reservations and thereby minimising the resource fragmentation.

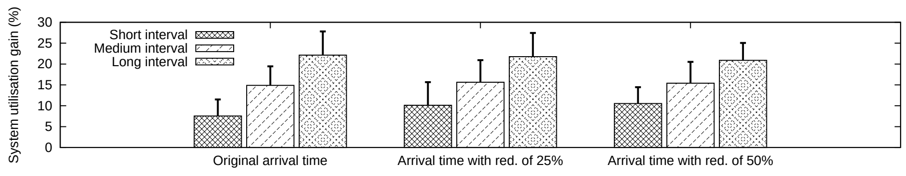

Fig. 3. Impact of time interval size on resource utilisation.

In a real scenario, users may not estimate their execution time accurately. To understand the impact of incorrect execution time estimates we performed the following experiment. We modified the actual execution time in the workload trace by a factor determined from a Poisson distribution with λ=80—we assume the users in general overestimate the execution time [4, 5].

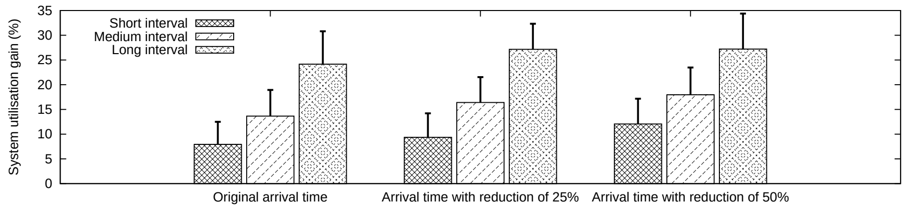

Fig. 4. Impact of time interval size on resource utilisation with inaccurate estimation time.

Figure 4 presents the results of this experiment. Compared to the results of Figure 3, we can observe that the flexible intervals have more impact when users overestimate their execution time, since otherwise the requests create small fragments that cannot be used by rigid time QoS requirements.

To ensure that a consumer knows with some assurance when the job will execute, we can fix the job when the time to receive the resources gets closer (it was initially a flexible interval job). To determine the impact of this approach, we evaluated the system utilisation by fixing the T_j^r and D_j of each job j when 25%, 50%, and 75% of the waiting time has passed. We compared these results with an approach that fixes the schedule immediately the job is accepted.

As in the first set of experiments (figure 2) we performed runs for different workloads. However in this case the results for all workloads were similar, therefore we only present the graph for the medium workload in figure 5. We observe that the longer a user waits to fix their job, the better is the system utilisation. This is a pleasing result as this is indeed what we would expect because the scheduler has more opportunities to reschedule the workload.

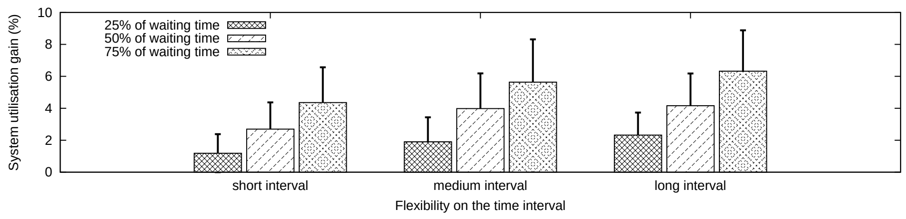

Fig. 5. The longer a job remains flexible, the better the utilisation. Premature fixing of a job's place in the schedule consistently has an adverse effect on resource utilisation.

Instead of using flexible intervals to meet time QoS requirements of users, we wanted to see what would happen when the resource provider offered an alternative slot to the consumer. When the resource provider cannot schedule a job j with the required starting time, it provides the user with another earlier option (if possible) before theinterval $[T_j^r, D_j]$. The difference in the starting times is defined as ϕ (see Section 3 for formal definition). Figure 6 shows that while this approach does increase the system utilisation, it does not perform as well as the flexible interval technique. Nevertheless, the approach of returning to the consumer with and alternative option is a useful technique for users who cannot flexible intervals.

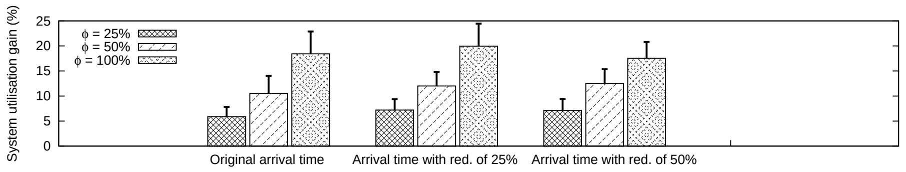

Fig. 6. System utilisation using suggested option from resource provider.

We also measured the actual and estimated time difference ϕ for the jobs accepted through the option suggested by the resource provider. From Figure 7 we observed that in average case, the value of ϕ is not significantly less than the maximum ϕ defined by the resource provider.

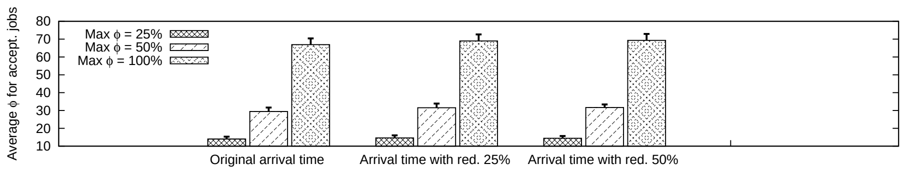

Fig. 7. Average actual ϕ of jobs accepted through suggestion by resource provider.

5 Related Work

Advance reservation is an important technique for aggregating resources from multiple places in such a way as to provide Quality-of-Service for users in a distributed computing environment. The interest in this technique has increased alongside with increasing popularity of Grid Computing.

Snell et al [6] discuss the importance of using advance reservations for executing meta jobs in multi-site environments and the problem of fragmentation generated in the computing environment due to these reservations. In their study they assume that advance reservations are strictly rigid in terms of time QoS requirements.

More recently researchers have become interested on how to improve system utilisation by including flexibilty factors in advance reservations. Naikasatam and Figueira defined *elastic reservations* in a context of network bandwidth management in LambdaGrids [7]. These elastic reservations are malleable requests (time X bandwidth) and they can be rescheduled over time. The goal of their approach is to minimise the problem of rejecting requests due to many users requiring data transfer channel at the same time-slot, and the problem of bandwidth fragmentation. In contrast to their work, we focus on the flexibility on the requests time intervals and not on the request malleability.

Chen and Lee [8] propose a flexible reservation model based on flexible intervals for starting time of advance requests. They handle the problem of optimising the scheduling by representing the advance reservations as a multistage digraph, and then finding the shortest path on the digraph. They explore the fact that there is a period between resource reservation and the real allocation, i.e. when the user starts accessing the resources, in which the scheduler rearranges the requests before they start. In contrast to their work, we consider that users may decide to fix their time schedule. That is, the flexibility is allowed until a certain period of time, since users may need to know the exact starting time to be reported some time in advance. Furthermore they do not consider requests for multiple resources.

Kaushik et al [9] study the use of flexible time intervals, which they call flexible time-windows, for advance reservations. They investigated the relation between the time-window size and the request waiting time, assuming that the request inter-arrival time follows the Pareto distribution. In our experiments we relied on inter-arrival requests from real workload from a supercomputing centre, and the Poisson distribution for defining the minimum starting time. We also consider that requests can come out of order. Furthermore they do not consider requests for multiple resources. Castillo et al [10] use concepts of computational geometry to handle resource fragmentation caused by advance reservations. In their study they consider only jobs with strict time intervals, and as in the other related work, only jobs requiring a single resource.

In addition, none of the related projects evaluate returning other scheduleable options on failure to schedule the initial request and they do not consider or provide experimental results with flexible interval scheduling when consumers incorrectly overestimate their execution time.

6 Conclusions and Further Work

In this paper we outlined consumer scenarios for advance reservations with flexible and adaptive time QoS parameters and presented the benefits for resource providers in terms of system utilisation. We evaluated these flexible advance reservations by using different scheduling algorithms, and different flexibility and adaptability QoS parameters. We investigated cases where users do not or can not specify the execution time of their jobs accurately. We also examined resource providers that do not utilise flexible time QoS parameters, but rather return alternative scheduling options to the consumer when it is not possible to meet the original QoS requirements.

In our experiments we observed that system utilisation increases with the flexibility of request time intervals and with the time the users allow this flexibility while they wait in the scheduling queue. This benefit is mainly due the ability of the scheduler to rearrange the jobs in the scheduling queue, which reduces the fragmentation generated by advance reservations. This is particularly true when users overestimate the execution time of their jobs.

For future work we can draw useful conclusions from these results. In particular the results can be used as a solid foundation for a utility computing pricing model as we have quantified the effects of varying degrees of flexibility on the utilisation of the provider's resources. Our work will include a pricing system for charging consumers for resources and give incentives or discounts for those users who are willing to provide flexibility within their QoS requirements and therefore include time flexible SLA parameters. We believe that this approach will allow resource providers to satisfy the full range of QoS timing requirements and in particular add a new option for some difficult scheduling domains such as workflow applications and resource co-allocation.

References

1. Auyoung, A., Grit, L., Wiener, J., Wilkes, J.: Service contracts and aggregate utility functions. In: Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing, Paris, France (June 19–23 2006) 119–131
2. Wu, Y.L., Huang, W., Lau, S.C., Wong, C.K., Young, G.H.: An effective quasi-human based heuristic for solving the rectangle packing problem. European Journal of Operational Research **141**(2) (2002) 341–358
3. Netto, M.A.S., Buyya, R.: Impact of adaptive resource allocation requests in utility cluster computing environments. In: Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid, Rio de Janeiro, Brazil, IEEE Computer Society (14-17 Sep 2006)
4. Chiang, S.H., Arpaci-Dusseau, A.C., Vernon, M.K.: The impact of more accurate requested runtimes on production job scheduling performance. In: Proceedings of the 8th International Workshop on Job Scheduling Strategies for Parallel Processing. Volume 2537 of Lecture Notes in Computer Science., Edinburgh, Scotland, UK, Springer (July 24 2002) 103–127
5. Lee, C.B., Snavely, A.: On the user-scheduler dialogue: Studies of user-provided runtime estimates and utility functions. International Journal of High Performance Computing Applications **20**(4) (2006) 495–506
6. Snell, Q., Clement, M.J., Jackson, D.B., Gregory, C.: The performance impact of advance reservation meta-scheduling. In: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing. Volume 1911 of Lecture Notes in Computer Science., Cancun, Mexico, Springer (May 1 2000) 137–153
7. Naiksatam, S., Figueira, S.: Elastic reservations for efficient bandwidth utilization in lambdagrids. Future Generation Computer Systems **23**(1) (2007) 1–22
8. Chen, Y.T., Lee, K.H.: A flexible service model for advance reservation. Computer Networks **37**(3/4) (2001) 251–262
9. Kaushik, N.R., Figueira, S.M., Chiappari, S.A.: Flexible time-windows for advance reservation scheduling. In: Proceedings of the 14th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), Monterey, USA (September 11–14 2006) 218–225
10. Castillo, C., Rouskas, G., Harfoush, K.: On the design of online scheduling algorithms for advance reservations and QoS in grids. In: Proceedings of the 21st IEEE International Parallel & Distributed Processing Symposium, Long Beach, USA (March 26–30 2007)