A Flexible Resource Co-Allocation Model based on Advance Reservations with Rescheduling Support

Marco A. S. Netto and 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

Several parallel and distributed applications require simultaneous access to resources located in multiple administrative domains. Current research on resource co-allocation relies on either rigid advance reservations or non-booking-in-advance mechanisms. The first approach leads to high fragmentation inside the resource provider's scheduling queue, whereas the second approach offers no starting time guarantees of user applications. In this work, we propose a new model for resource co-allocation based on flexible advance reservations and processor remapping. The model allows the metascheduler to reschedule the co-allocation requests by modifying the starting time of each subtask and remapping the number of processors used by them in each resource provider. We evaluate our model and algorithms in a scenario where users are not able to provide accurate runtime estimations of their applications—using job response time and system utilization as metrics. The results show that rescheduling co-allocation requests brings benefits for both local and multi-site applications especially when the runtime estimation quality is low and there is a reduced number of small jobs in the system.

1 Introduction

A number of distributed applications require simultaneous access to resource spread over different administrative sites; problem known as resource co-allocation [6]. There are some reasons for requiring resource co-allocation [20]: (i) applications may require certain computing power or different resources that are not available in a single site; or (ii) users may need to reduce the response time of their applications by using resources from multiple sites. In addition, users can use resource co-allocation to improve fault tolerance of their applications through redundancy of resources.

In large-scale environments, resource allocation usually requires a considerable amount of human interaction. This interaction is necessary to handle resource or software failures, define qual-

ity of service, negotiate resource usage prices, deal with security and access control policies, and schedule the requests. The interaction problem becomes more complicated when resources managed by different resource providers must be available at the same time. The reason is that the resource providers have to manage their own users and have their own goals and policies. In addition, users are responsible for interacting with each resource provider individually. Thus, we are far from executing complex large-scale applications effectively on distributed and autonomous resources in a completely automatic fashion.

A simple and straightforward solution for resource co-allocation is requesting the scheduling queue for each resource provider, verifying a free common timeslot that fits the request in all queues, and submitting the request parts to the resource providers. Although this approach works for some cases, there are a number of problems and optimizations to consider in order to co-allocate resources efficiently.

Initially, resource co-allocation was performed with no advance reservations [5]. In that case, if all the resources were available at the same time, the user could access them, otherwise the user had to wait for another opportunity (all-or-none approach). Nowadays most of the research on resource co-allocation relies on advance reservations [7, 8, 14, 17]. Advance reservations are important to guarantee that resources are available at the expected time, so users can have a certain level of quality of service. However, advance reservations reduce resource utilization due to the inflexibility they generate to schedule the other requests [20], being those advance reservations or not. Resource providers usually have timeslot fragments in their request queues generated by users supplying wrong execution time estimations or by modifications on the computing systems, e.g. resource availability. If a resource provider needs to modify an advance reservation to optimize a system metric, the user has to renegotiate the request with all the other resource providers involved in the co-allocation.

Recently, researchers working with advance reservations started to rely on flexibility of starting times and deadlines of requests in order to minimize the impact of advance reservations on system utilization [10, 13, 15, 18, 19]. Rather than specifying tight starting and completion times for advance reservations, users specify relaxed time intervals in order to allow the resource provider to optimize the scheduling. Nevertheless, the use of these flexible advance reservations for resource co-allocation has been barely explored. In particular, to the best of our knowledge, no work has investigated the rescheduling of these co-allocation requests in autonomous multi-site computing environments.

The contributions of this paper are (i) a resource co-allocation model based on flexible advance reservations and processor remapping; and (ii) the rescheduling support for multi-site parallel jobs. Due to the changes in the resource provider's queue, the rescheduling of co-allocation requests becomes important, which has not been explored in current resource co-allocation solutions. It is important to highlight that some of the ideas here can be directly applied to co-allocation of other resources such as network bandwidth links. Our work tackles the problem of co-allocation from the moment when the metascheduler has already selected the resource providers until the moment when the user applications start the execution.

The rest of this paper is organized as follows: Section 3 presents a detailed description of flexible resource co-allocation requests, including their properties and operations. Section 4 introduces algorithms for scheduling and rescheduling co-allocation requests. Section 5 presents an extensive evaluation of the flexible co-allocation model to show in which cases the use of the proposed

model reduces the response time of users applications. Section 6 provides a description of related projects on resource co-allocation. We finalize the paper in Section 7 with our concluding remarks and further work.

2 Problem Description

In the co-allocation problem an application needs to access resources spanning different autonomous sites, named *resource providers*, simultaneously. A metascheduler books resources in advance on behalf of the users. This metascheduler needs to find a *common timeslot* in the resource providers. These resource providers manage scheduling queues for both local and external requests. The scheduling queues must be updated over time due to inaccurate estimation of resource usage, cancellations or modifications of user requirements. Therefore resource providers may need to modify parts of a co-allocation request, named *sub-requests*, to increase their system utilization. However, the metascheduler must keep all the sub-requests synchronized, which may be difficult because each resource provider has its own workload, strategies, and goals. In this paper explore the inaccurate estimation of application executions as the main source for generating the fragments in the scheduling queues.

2.1 Computing environment and resource management system

The resources considered are space-shared high performance computing (HPC) machines, e.g. clusters or massively parallel processing (MPP) machines, $M = \{m_1, m_2, ...m_k\}$, where k is the total number of machines. Each machine $m_i \in M$ has a set of resources, or processors, $R = \{r_1, r_2, ...r_n\}$ where n is the total number of resources (or nodes) in a given machine m_i. For the sake of simplicity we assume that all the resources R in a given machine m_i are homogenous—which is a reasonable assumption considering that most of the clusters are composed of homogeneous resources. The machines in M can be heterogeneous among them. We consider that there is a network interconnecting these machines in order to execute parallel applications, which can be either exclusive or shared in an open environment such as the Internet.

Resource Management Systems (RMSs), named *local schedulers*, are responsible for scheduling both local and external requests in each resource provider. A metascheduler, on behalf of users requiring co-allocation requests, is responsible for negotiating the timeslots with the local schedulers. We do not assume that a metascheduler has total information of the local schedulers. In our scenario, rather than publishing the complete scheduling queue to the metascheduler, the local schedulers may want to publish only certain timeslots to optimize some local system usage. Moreover, in our computing environment schema the resource providers have no knowledge about one another. The scheduling management policy we use here is a FIFO with conservative backfilling, which provides a good level of guarantee of completion time once the users receive their scheduling timeslots.

2.2 Application model

We investigate the resource co-allocation for *parallel applications* that require simultaneous access to resources from multiple sites. This requirement may be due to: (i) insufficient resources provided by a single resource provider to execute an application with a feasible turnaround time;

and (ii) to reduce the response time by using the fragments of the scheduling queues from multiple resource providers. We consider here applications that are mainly CPU bound. Data intensive applications would require some special treatment, and therefore we do not handle them in this work.

In order to co-allocate resources we consider the worst-case scenario in terms of starting time, i.e. all application processes must start exactly at the same time. That is mainly required by parallel applications with data exchange among the processes.

The metascheduler is responsible for *decomposing* the job, named *metajob* or *external job*, request Job_j into a set of sub-requests $Job_j = \{job_j^1, job_j^2, ..., job_j^k\}$, where k is the number of sub-requests, which are executed by each machine m_i. Note that, in some cases, the user may want to incorporate some constraints to decompose the request.

2.3 Metrics

Our main optimization criteria here is the response time of user applications from both external and local users, which is defined as the time difference between the submission time of the user request and its completion time.

3 Flexible Resource Co-allocation

The flexible resource co-allocation (FlexCo) model proposed here is inspired by existing work on flexible advance reservations [10, 13, 15, 16, 18, 19]. A FlexCo request can have relaxed starting and completion times, can be moldable, i.e. number of allocated resources is not rigid, and can have flexibility on defining the number of processors used in each resource provider. As a FlexCo request is composed of sub-requests that are submitted to different machines (homogeneous or heterogeneous). Each sub-request may have a different number of resources with different capabilities.

We define a FlexCo request as a request model with the following operations (Figure 1):

- *Starting time shifting:* changes the starting time according to the relaxed time interval—the change must be the same for all sub-requests.
- *Processor remapping:* changes the number of required resources of two or more sub-requests;
- *Moldability:* changes the number of resources and execution time—the change must be the same for all sub-requests;

Combining operations is also important for the scheduler. In Figure 1 we observe that after using the moldability or processor remapping operations, it is possible to shift the requests in order to reduce the user response time.

The **starting time shifting (Shift)** operation is motivated by the fact that finding a common timeslot may be difficult for the users, hence once they commit the co-allocation based on advance reservations, they will not be willing to change it. The modification of the starting time may be useful for one resource provider in order to fill a fragment in the scheduling queue. If the other resource providers are also willing to shift the advance reservations to start earlier, the users will also have benefits. Remark that this operation is not application dependent in the sense that it is

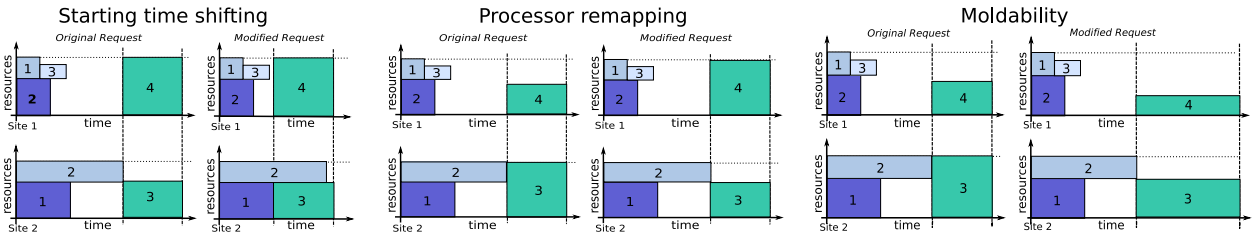

Figure 1. Operations of a FlexCo request.

only a shift on the starting time of the user application. However, as we mentioned in Section 2.2, data intensive applications would require some special method, since a shift would not be possible if the data is not ready to be processed.

The **processor remapping (Remap)** operation has the same motivation as the previous operation but works with the number of resources in each site. Since negotiating resources is a difficult process, a user requiring certain number of resources tends to decompose the request statically according to the available providers at a certain time. There are at least two problems with this approach: users may not be able to reduce the starting time, and the resources providers may not be able to perform optimizations that could, for example, reduce completion time of user applications. With this operation it is possible to remap the processors dynamically once the sub-requests are queued. This operation is application dependent since the throughput offered by each resource provider may influence the overall performance of the user application. Therefore, the users may also want to incorporate some restrictions on how the metascheduler should map and remap their requests.

The **moldability (Mold)** operation is a more complex operation since in practice it is not easy for the users to define moldability functions. This operation would require a moldability function for each different type of resource. Alternatively, the user could provide a single moldability function that represents the slowest machine when jobs require synchronization. In this work we describe scheduling strategies for the Shift and Remap operations only.

Following are the parameters and notations to represent a FlexCo request for a job j:

- $R_j^{m_k}$: number of resources, in our case CPUs, required in each site m_i, where k is the total number of sub-requests of the job j;
- T_j^s: job starting time—time determined by the scheduler;
- T_j^e: job execution time;
- T_j^x: job estimated execution time;
- T_j^r: job ready time—minimum starting time determined by the user;
- T_j^c: job completion time—defined as $T_j^s + T_j^e$;
- T_j^{xo}: job estimated network overhead when using multiple sites.

4 Scheduling of FlexCo Requests

The scheduling of a FlexCo request consists of finding a free common timeslot that meets the job requirements in a set of resource providers. We consider here the scheduling to be *on-line*, where users submit jobs to the resource provider's scheduler over time and the schedulers make decisions based only on the currently accepted jobs. The scheduling involves the manipulation of timeslots, which are data structures composed of four values:

- ts^{id}: time slot identification;
- ts^{s}: time slot starting;
- ts^{c}: time slot completion time;
- ts^{n}: number of resources available in this time slot.

4.1 Initial scheduling

The initial scheduling consists of mainly four stages, which are described by Algorithm 1. First the metascheduler asks the resource providers for the list of available timeslots, $TS = \{ts_1, ts_2, ..., ts_t\}$, where t is the number of timeslots (Lines 1-4). Second the metascheduler finds a common starting time cs that meets the request constraints, such as number of resources, starting time, and completion time (Lines 5-23). Third the metascheduler generates a list of sub-requests (Lines 24-29). Finally, in the forth stage, the metascheduler submits the sub-requests to the resource providers accordingly (Lines 30-32).

In order to find the common starting time cs (Lines 5-23), the algorithm verifies cs according to the list of available time slots TS and gets the maximum number of resources available in each machine m_i starting at time cs that fits the job. Note that if the number of resources available in a particular m_i is greater than or equal to R_j, there is no need to consider the network overhead T_j^{xo} since the job will be submitted only to that machine.

When generating the list of sub-requests (Lines 24-29), the metascheduler could follow different approaches. For example, it could try to decompose the jobs in an even way in order to maintain the same load in each resource provider. In our approach the metascheduler allocates as many processors as possible from a single resource provider per request. Every time a new external job arrives, the metascheduler uses the next-fit approach to give priority to the next resource provider. The idea behind of the second approach is to increase the chances of fitting some metajobs in a single site over time due to the rescheduling.

4.2 Rescheduling

As described in the previous subsection, the initial scheduling of a metajob involves manipulation and transfer of timeslots over the network. In order to reschedule metajobs, one must consider the cost-benefit of transfering and manipulating timelots to optimize the schedule. Therefore our approach is to reschedule a metajob only when the resource provider is not able to find a local job that fills the fragment generated due to the earlier completion time of a job (Figure 2). The local schedulers use Algorithm 2 for rescheduling jobs whenever a job completes before the estimated time. The rescheduling is based on the *compressing* method described by Weil and Feilson [22], which consists of bringing the jobs to the current time according to their starting times, not arrival

Algorithm 1 Pseudo-code for scheduling a new FlexCo request j_k.

1: **for** $\forall m_i \in M$ **do**
2: $TS_i \leftarrow$ Get time slots from m_i
3: Sort $TS_i \mid ts_1^s \leq ts_2^s \leq ... \leq ts_t^s$
4: **end for**
5: $foundLocalSite \leftarrow false$
6: $bestT_j^s \leftarrow null$
7: $cs \leftarrow null$
8: **while** $foundLocalSite \leftarrow false$ and cs is not the last in TS **do**
9: $cs \leftarrow$ Get next starting time from TS
10: $totalAvailableResources \leftarrow 0$
11: $listAvailableResources \leftarrow null$
12: **for** $\forall m_i \in M$ **do**
13: $numResources \leftarrow$ max R from $m_i \mid \Big(cs \geq T_j^r$ and $(cs+T_j^x+T_j^{xo} \leq ts_i^c$ or $cs+T_j^x \leq ts_i^c$ when $R \geq R_j)\Big)$
14: $totalAvailableResources \leftarrow totalAvailableResources + numResources$
15: **if** $numResources \geq R_j$ **then**
16: $foundLocalSite \leftarrow true$
17: **end if**
18: $listAvailableResources_i \leftarrow numResources$
19: **end for**
20: **if** $foundLocalSite = false$ and $totalAvailableResources \geq R_j$ and $bestT_j^s \neq null$ **then**
21: $bestT_j^c \leftarrow cs + T_j^x + T_j^{xo}, \quad bestT_j^s \leftarrow cs, \quad bestM \leftarrow m_i$
22: **end if**
23: **end while**
24: **if** $foundLocalSite = true$ and ($bestT_j^s \neq null$ or $cs + T_j^x \leq bestT_j^c$) **then**
25: $subreqs \leftarrow$ Get sub requests ($bestM$)
26: **else if** $bestT_j^c \neq null$ **then**
27: $subreqs \leftarrow$ Get sub requests ($listAvailableResources$)
28: $cs \leftarrow bestT_j^s$
29: **end if**
30: **if** found cs **then**
31: Submit $subreqs$ to required $m_i \in M$
32: **end if**

times (Lines 2-7, 17-21). This avoids the violation of the completion time of jobs given by the original schedule. When implementing the algorithm, one could keep a list of sorted jobs according to starting time instead of sorting them when executing the rescheduling (Line 2).

Once the metascheduler receives a notification for rescheduling a job j_i from the resource provider, it performs the rescheduling in a similar way as described in the initial scheduling procedures (Algorithm 1). The main difference is that the metascheduler may need to remove the subrequest from a resource provider since a job may not need all the original sites anymore.

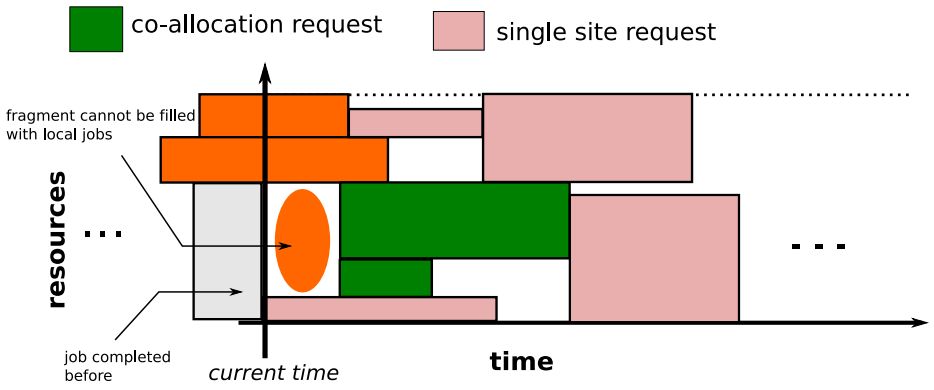

Figure 2. Reschedule of multi-site jobs in the head of the waiting queue.

Algorithm 2 Pseudo-code for rescheduling jobs in the resource provider.

```
1: coallocHeadRescheduled ← false
2: Sort Q^w | {T_1^s ≤ T_2^s ... ≤ T_n^s}, where n is number of jobs in the waiting queue
3: for ∀ j_i ∈ Q^w do
4:    if j_i is local job then
5:       Schedule job with backfilling
6:    end if
7: end for
8: if there are idle resources then
9:    for ∀ multisite jobs j_i in the head of the queue do
10:      previousT_{j_i}^c ← T_{j_i}^c
11:      Contact metascheduler to reschedule j_i
12:      if T_{j_i}^c ≤ previousT_{j_i}^c then
13:         coallocHeadRescheduled ← true
14:      end if
15:   end for
16:   if coallocHeadRescheduled = true then
17:      for ∀ j_i ∈ Q^w do
18:         if j_i is local job then
19:            Schedule job with backfilling
20:         end if
21:      end for
22:   end if
23: end if
```

5 Evaluation

We evaluated our strategies using an event-driven simulator, named PaJFit (Parallel Job Fit) [16], which has been extended to support multi-site environments and FlexCo requests. We used real traces from supercomputers available at the Parallel Workloads Archive[1] to model the user

[1]Parallel Workloads Archive: http://www.cs.huji.ac.il/labs/parallel/workload

applications. We compared the use of Shift and Shift with Remap operations against the traditional co-allocation model based on rigid advance reservations, which provides response time guarantees but suffers from high fragmentation inside resource provider's scheduling queues. This section presents a detailed description of the environment setup and metrics followed by the results and our analysis.

5.1 Experimental configuration

We modeled an environment composed of 4 clusters with their own load and one metascheduler which receives metajobs from external users. For the local jobs we used the traces from the 416-node Intel Paragon located at the San Diego Supercompter Center (SDSC). The Parallel Archive has two traces from this machine. We used the version 2.1, year 1995 and version 2.1 year 1996. We split each of these files in halves and synchronized the submission times in order to have 4 files which represent the load of each one of the simulated clusters for 6 months. For the metajobs we used a larger machine, the San Diego Supercomputer Center (SDSC) Blue Horizon with 1,152 processors: 144-node IBM SP, with 8 processors per node, trace version 3.1 year 2000. In order to have a considerable number of metajobs for 4 clusters, we used the first year of this trace, split in 2 parts of 6 months each and mixed them by bringing the load of the second semester to the first semester, hence maintaining load approximately 80% in the local clusters.

The traces we chose for the local jobs provide no user estimation times, only the actual job execution times. In order to include user estimations we relied on a model proposed by Tsafrir et al. [21][2]. As the mandatory parameter of this model, the maximum estimation time, we used 13 hours. We also generated different runtime estimations using alternative random number generator seeds.

For the network overhead of multi-site jobs, we assigned to each job a randomly value defined by a Poisson distribution with λ=20. A study by Ernemann et al. [9] shows that it pays off to use co-allocation when the penalty for the network overhead is up to approximately 25%. Therefore, we limited the network overhead under this value.

We evaluated the system utilization, response time, which is the difference between the job completion time and submission time, as well as the bounded-slowdown, which is the response time normalized by the running time [11]:
bounded-slowdown = $max\left\{\frac{Tw+Tr}{max\{Tr,\tau\}}, 1\right\}$, where Tw is the waiting time, Tr is the task running time, and τ is a threshold factor which we setup with the value of 10 minutes.

We investigate the behavior of these metrics according to the precision of the job runtime estimations, which we varied from the original value defined in the workload to 25% plus the original value. We also evaluated these metrics by executing the experiments without considering small jobs, i.e. $T^x \leq 1$ hour and $R \leq 64$.

5.2 Results and analysis

We calculated the values of the response time and slowdown for (a) all jobs, for (b) only local jobs, and for (c) only meta jobs in order to have a better understanding of the behavior of the metrics for each type of user, i.e. local and external users.

[2]Estimation runtimes generator model: http://www.cs.huji.ac.il/labs/parallel/workload/m_tsafrir05

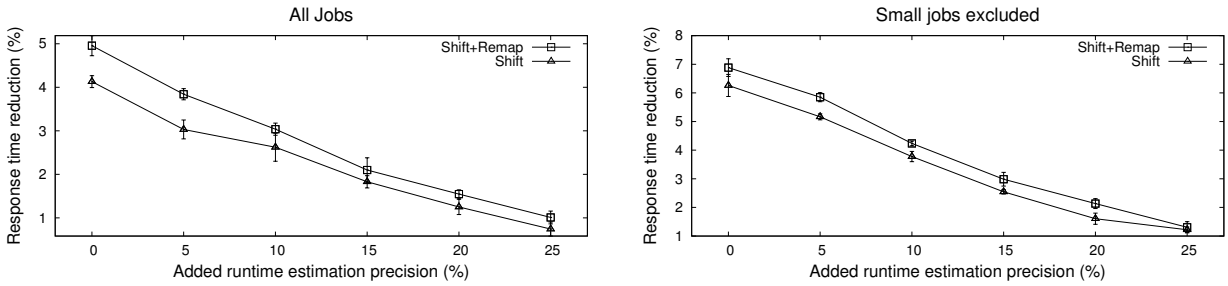

Figure 3. Response time reduction considering both local and meta jobs.

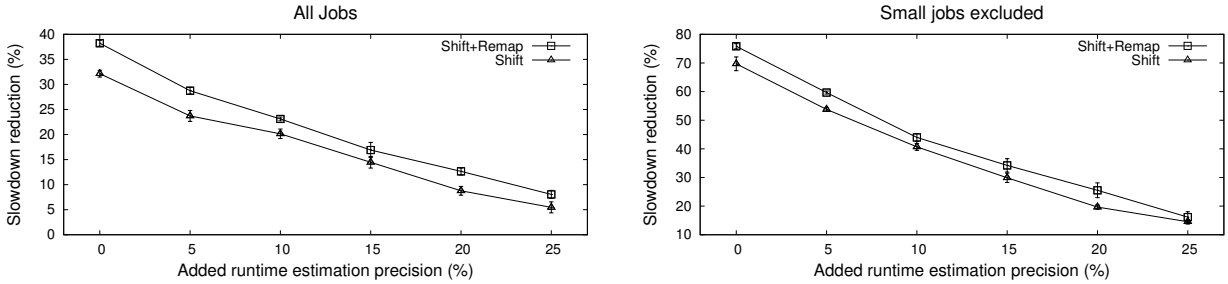

Figure 4. Slowdown reduction for both local and meta jobs.

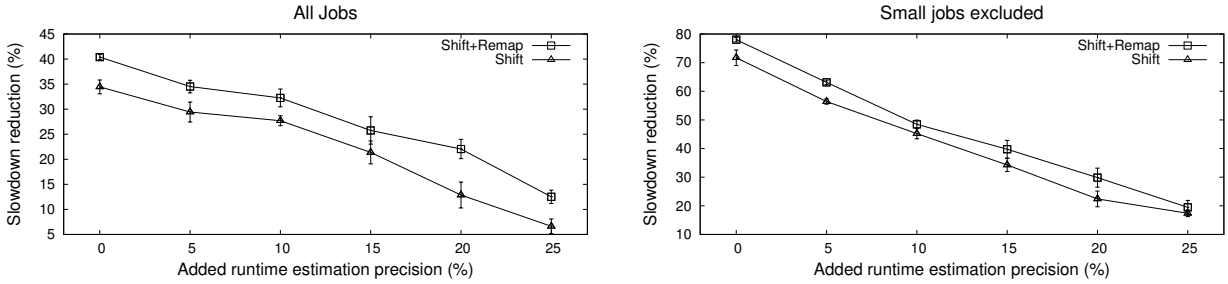

Figure 5. Slowdown reduction for local jobs.

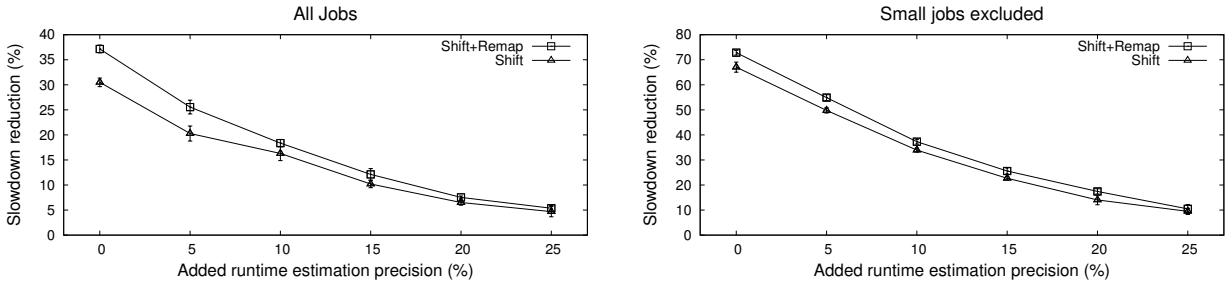

Figure 6. Slowdown reduction for meta jobs.

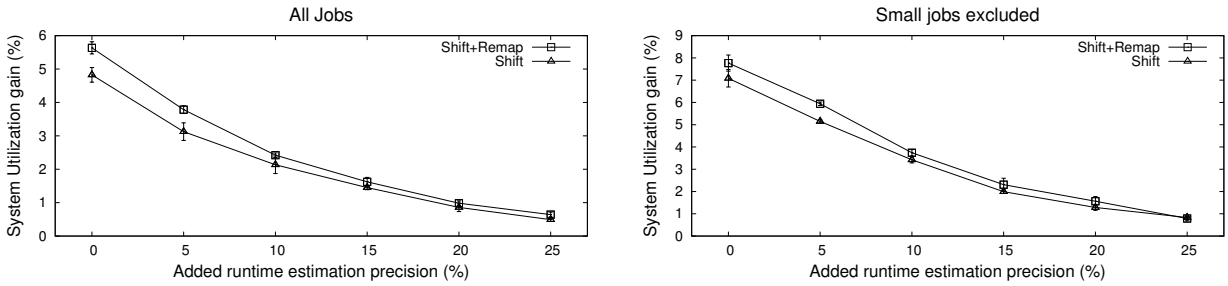

Figure 7. Global system utilization gain.

Figure 3 presents the response time reduction when using FlexCo requests with Shift and Shift with Remap operations, both over the co-allocation using the rigid advance reservation. We observe that the FlexCo requests with Shift and processor Remapping together generate better results than only with Shift operation. This happens because jobs have more flexibility to find better scheduling options that generate earlier completion times. Moreover, having these two operations together increases the possibility of moving multi-site jobs into a single cluster, hence reducing significantly the response time due to the network overhead that is eliminated. In addition, when small jobs are removed, the response time reduction is higher since there are no jobs to fit the gaps generated by the jobs that completed before the expected time. We can also conclude that the FlexCo requests have a higher impact on environments where users do not provide precise runtime estimations. The results for only local and only external jobs are the same for the response time, thus we have not included them here.

Figures 4, 5, 6 present the slowdown reduction when using FlexCo requests with Shift and Remap operations and only Shift, both over the co-allocation using rigid advance reservations. We observe that the reduction is more significant than the response time since the slowdown normalizes the response time with the execution time of the jobs. Therefore small jobs that are delayed have a higher impact; therefore, the reductions of the head co-allocation jobs minimize the delay of the short jobs. In addition, similar to the response time, we observe the benefit of the FlexCo requests over the traditional co-allocation based only on rigid advance reservations. Comparing the benefit of external and local users, we see that there is a slightly advantage for the local users in relation to the meta users. This is because the metric slowdown gives more advantage for small jobs, which constitutes a large number of local jobs and not external users.

Figure 7 shows the global system utilization gain by using the model with rescheduling support. The metric has the same behavior as the other metrics but with different values.

6 Related Work

Czajkowski et al. [5] deal with the co-allocation problem with the main focus on failures during allocation. Different from our work, they do not use advance reservations due to the lack of support of the local resource managers at that time. They rely on a solution based on the current availability of the resources and queue-time estimations of the resource providers. Later Czajkowski et al. [6] propose an approach in which users could modify the co-allocation specification their requests initialize via *add*, *delete*, and *substitute* operations. Moreover in their work resources could be

classified in categories, *required*, *interactive* and *optional*, in order to simplify the management of resource failures.

Foster et al. [12] propose and describe the prototype of the Globus Architecture for Reservation and Allocation (GARA). This prototype aims to provide a platform with support for quality of service guarantees through advanced reservations. Their work focuses more on middleware aspects rather than on scheduling optimizations.

Alhusaini et al. [1, 2] investigate the mapping of a set of independent tasks on compute and non-compute distributed resources. The tasks have co-allocation requirements and their goal is to minimize schedule length. Their solution is based on a two-phase approach. The first phase is an off-line planning where the scheduler assigns tasks to resources assuming that all the applications hold all the required resources for their entire execution. The second phase is the run-time adaptation phase where the scheduler takes decisions according to the actual computation and communication costs, which may be different from the estimated costs used in the first phase. Some applications may release some resources before the conclusion of the execution. Different from our work they assume that each task to be mapped is known a priori and that all the resources are exclusive for the co-allocation tasks, i.e. there are no local jobs competing for resources. Similar to our work they consider wrong estimation of job requirements and the need of a rescheduling phase to overcome the problem.

MacLaren et al. [14] discuss the problem of resource co-allocation, in particular focusing on fault tolerance, and propose a co-allocation system called HARC (Highly-Available Robust Co-allocator). Their system uses a two-phase approach based on advance reservations to handle the distributed transaction problem. Similar to our approach, the scheduler does not have access the scheduling queue of the resource managers but asks for the free time slots. The system supports the creation of a reservation, cancellation, modification of number of requested CPUs and time of the reservation. They do not address issues such as finding an optimal scheduling or managing the reservations once they have been made. Therefore we see HARC as a middleware which provides services that can be used to deploy the policies described in this paper.

Azzedin et al. [3] propose a co-allocation mechanism that does not rely on advanced reservations. Their main argument for this approach is the strict timing constraints on the client side due to the advance reservations, i.e. once a user requests an allocation, the initial and final time are fixed. Consequently, advanced reservations generate fragments that the schedulers cannot utilize. Furthermore, the authors argue that a resource provider can reject a co-allocation request at any time in favor of internal requests, and hence the co-allocation would fail. Their schema, called synchronous queuing (SQ), synchronizes the subtasks at the scheduling cycles (or more often), by speeding them up or slowing them down. One of the main problems of this approach is the *co-allocation skew*, i.e. time difference between the fastest running and the slower running subtask, may be long. Therefore, the resources would not be effectively utilized. Another problem is that, depending on the application and on the computing environment, it is not possible to modify the subtasks execution speed. The strategies we have presented in this paper are a step to overcome the limitations the authors mentioned about using advance reservations for co-allocation.

Bucur and Epema [4] investigate scheduling policies and different queuing structures for resource co-allocation in multicluster systems. They evaluate the differences of having single global schedulers, only local schedulers and both structures together, as well as different priorities for local and metajobs. Moreover, they do not use advance reservations and therefore it is not possible

to give guarantees of completion time to the users requiring co-allocation and local resources may be idle until all the co-allocation requirements are satisfied.

7 Conclusions and Further Work

In this paper we have proposed and evaluated a resource co-allocation model for multi-site parallel jobs based on flexible advance reservations and processor remapping. The model overcomes the limitations of existing solutions for resource co-allocation in relation to the starting time guarantees of user applications and the reduction of fragments in the scheduling queues of the resource providers. From our experiments we concluded that the model is particularly important in environments where users are not able to provide accurate runtime estimation time of their tasks and where there is a lack of small jobs to fill the fragments.

Rescheduling co-allocation requests involves the cooperation of different autonomous parties. In this paper we investigated a cooperative environment, which is a typical research environment such as TeraGrid and Grid'5000. As future work we will explore a more competitive environment where resource providers need to increase their own system utilization. Moreover, in this paper we have used the FlexCo requests to improve the user response time. As next step we will also explore the modification of the already scheduled FlexCo requests in order to meet the requirements of incoming requests.

References

[1] A. H. Alhusaini, V. K. Prasanna, and C. S. Raghavendra. A framework for mapping with resource co-allocation in heterogeneous computing systems. In *Proceedings of the Heterogeneous Computing Workshop (HCW), in conjunction with the 14th International Parallel & Distributed Processing Symposium (IPDPS)*, pages 273–286, 2000.

[2] A. H. Alhusaini, C. S. Raghavendra, and V. K. Prasanna. Run-time adaptation for grid environments. In *Proceedings of the Heterogeneous Computing Workshop (HCW), in conjunction with the 15th International Parallel & Distributed Processing Symposium (IPDPS)*, pages 864–874, San Francisco, USA, 23-27 April 2001.

[3] F. Azzedin, M. Maheswaran, and N. Arnason. A synchronous co-allocation mechanism for grid computing systems. *Cluster Computing*, 7(1):39–49, 2004.

[4] A. I. D. Bucur and D. H. J. Epema. Scheduling policies for processor coallocation in multicluster systems. *IEEE Transactions on Parallel and Distributed Systems*, 18(7):958–972, 2007.

[5] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In *Proceedings of the 4th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, volume 1459 of *Lecture Notes in Computer Science*, pages 62–82, Orlando, USA, 1998. Springer.

[6] K. Czajkowski, I. Foster, and C. Kesselman. Resource co-allocation in computational grids. In *Proceedings of the 8th International Symposium on High Performance Distributed Computing (HPDC)*, pages 219–228, Redondo Beach, USA, 3-6 Aug. 1999. IEEE Computer Society.

[7] J. Decker and J. Schneider. Heuristic scheduling of grid workflows supporting co-allocation and advance reservation. In *Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, pages 335–342, Rio de Janeiro, Brazil, May 14-17 2007. IEEE Computer Society.

[8] E. Elmroth and J. Tordsson. A standards-based grid resource brokering service supporting advance reservations, coallocation and cross-grid interoperability. Submitted: `http://www.cs.umu.se/~elmroth/papers/elmroth_tordsson_2006_draft.pdf`. Dec 2006.

[9] C. Ernemann, V. Hamscher, U. Schwiegelshohn, R. Yahyapour, and A. Streit. On advantages of grid computing for parallel job scheduling. In *Proceedings of the 2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid*, pages 39–, Berlin, Germany, 22-24 May 2002. IEEE Computer Society.

[10] U. Farooq, S. Majumdar, and E. W. Parsons. A framework to achieve guaranteed QoS for applications and high system performance in multi-institutional grid computing. In *Proceedings of the 35th International Conference on Parallel Processing (ICPP)*, pages 373–380, Columbus, USA, August 14–18 2006. IEEE Computer Society.

[11] D. G. Feitelson. Metrics for parallel job scheduling and their convergence. In *Proceedings of the 7th International Workshop Job Scheduling Strategies for Parallel Processing (JSSPP)*, volume 2221 of *Lecture Notes in Computer Science*, pages 188–206, Cambridge, USA, 2001. Springer.

[12] I. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt, and A. Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In *Proceedings of the 7th International Workshop on Quality of Service (WQoS)*, pages 27–36, 31 May-4 June 1999.

[13] N. R. Kaushik, S. M. Figueira, and S. A. Chiappari. Flexible time-windows for advance reservation scheduling. In *Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 218–225, Monterey, USA, September 11–14 2006. IEEE Computer Society.

[14] J. Maclaren, M. M. Keown, and S. Pickles. Co-allocation, fault tolerance and grid computing. In *Proceedings of the 5th UK e-Science All Hands Meeting (AHM)*, pages 155–162, Nottingham, UK, September 18–21 2006.

[15] S. Naiksatam and S. Figueira. Elastic reservations for efficient bandwidth utilization in lambdagrids. *Future Generation Computer Systems*, 23(1):1–22, 2007.

[16] M. A. S. Netto, K. Bubendorfer, and R. Buyya. SLA-based advance reservations with flexible and adaptive time QoS parameters. In *Proceedings of the 5th International Conference on Service-Oriented Computing (ICSOC)*, volume 4749 of *Lecture Notes in Computer Science*, pages 119–131, Vienna, Austria, 2007. Springer.

[17] T. Röblitz and A. Reinefeld. Co-reservation with the concept of virtual resources. In *Proceedings of the 5th IEEE International Symposium on Cluster Computing and the Grid (CCGrid)*, pages 398–406, Cardiff, UK, May 9–12 2005. IEEE Computer Society.

[18] T. Röblitz, F. Schintke, and A. Reinefeld. Resource reservations with fuzzy requests. *Concurrency and Computation: Practice and Experience*, 18(13):1681–1703, 2006.

[19] T. Röblitz, F. Schintke, and J. Wendler. Elastic grid reservations with user-defined optimization policies. In *Proceedings of the Workshop on Adaptive Grid Middleware (AGridM)*, September 2004.

[20] Q. Snell, M. J. Clement, D. B. Jackson, and C. Gregory. The performance impact of advance reservation meta-scheduling. In *Proceedings of the 6th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, volume 1911 of *Lecture Notes in Computer Science*, pages 137–153, Cancun, Mexico, May 1 2000. Springer.

[21] D. Tsafrir, Y. Etsion, and D. G. Feitelson. Modeling user runtime estimates. In *Proceedings of the 11th International Workshop Job Scheduling Strategies for Parallel Processing (JSSPP)*, volume 3834 of *Lecture Notes in Computer Science*, pages 1–35. Springer, 2005.

[22] A. M. Weil and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):529–543, 2001.