Rescheduling Co-Allocation Requests based on Flexible Advance Reservations and Processor Remapping

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

Large-scale computing environments, such as TeraGrid, Distributed ASCI Supercomputer (DAS), and Grid'5000, have been using resource co-allocation to execute applications on multiple sites. Their schedulers work with requests that contain imprecise estimations provided by users. This lack of accuracy generates fragments inside the scheduling queues that can be filled by rescheduling both local and multi-site requests. Current resource co-allocation solutions rely on advance reservations to ensure that users can access all the resources at the same time. These co-allocation requests cannot be rescheduled if they are based on rigid advance reservations. In this work, we investigate the impact of rescheduling co-allocation requests based on flexible advance reservations and processor remapping. The metascheduler can modify the start time of each job component and remap the number of processors they use in each site. The experimental results show that local jobs may not fill all the fragments in the scheduling queues and hence rescheduling co-allocation requests reduces response time of both local and multi-site jobs. Moreover, we have observed in some scenarios that processor remapping increases the chances of placing the tasks of multi-site jobs into a single cluster, thus eliminating the inter-cluster network overhead.

1 Introduction

One of the promises of Grid Computing is to enable the execution of applications across multiple sites. These applications can be bag-of-tasks, workflows or parallel applications based on message passing paradigm. While bag-of-tasks do not require all the resources to be available at the same time, some phases of workflows and parallel applications do require simultaneous access to resources spread over multiple sites—problem known as resource co-allocation [7].

Large-scale computing environments, such as TeraGrid, Distributed ASCI Supercomputer (DAS), and Grid'5000, are using resource co-allocation to execute applications on multiple sites. TeraGrid has deployed Generic Universal Remote (GUR) [32] and Highly-Available Resource Co-allocator (HARC) [18], the DAS project has developed KOALA [19], and Grid'5000[1] relies on the OAR(Grid) scheduler [5] to allow the execution of applications requiring co-allocation. Semiconductor processing [29] and computational fluid dynamics [9] are some examples of applications that have been used on multi-site environments.

Some of the reasons for requiring resource co-allocation are [28]: (i) applications may require certain computing power or different resources that are not available in a single site; or (ii) users may want to reduce the response time of their applications by using resources from multiple sites. Researchers have also investigated how co-allocation can reduce the job response time by merging fragments of multiple scheduling queues considering the network overhead associated with the execution [4, 11, 14].

Most of the current resource co-allocation solutions rely on advance reservations [8,10,13,18,24]. Although advance reservations are important to guarantee that resources are available at the expected time, they reduce resource utilization due to the inflexibility introduced in scheduling other jobs around the reserved slots [28]. To overcome this problem, many researchers are working with flexible (or elastic) advance reservations, i.e. requests that have relaxed time intervals [12, 16, 21, 25, 26]. Nevertheless, the use of these flexible advance reservations for resource co-allocation has been barely explored [15].

By introducing flexibility to the advance reservations of co-allocation requests, schedulers can hence reschedule them to increase system utilization and improve response time of both local and multi-site jobs. This is particu-

[1] Grid'5000 had approximately 6000 co-allocation requests in 2.5 years, i.e. an average of 200 requests per month. Data collected from Grid Workloads Archive: http://gwa.ewi.tudelft.nl/pmwiki

larly necessary due to the wrong estimations provided by users [17,20,30].

Little research has been devoted to resource co-allocation with rescheduling support [1, 2]. Therefore the primary contribution of this paper is a *resource co-allocation model based on flexible advance reservations and processor remapping, which allows the rescheduling of multi-site parallel jobs* (Sections 3 and 4). The flexible advance reservations are used to shift the start time of the job components, whereas the processor remapping allows multi-site jobs to change the number of processors and clusters they use. These changes make it possible to remap multi-site jobs to a single cluster, thus eliminating unnecessary network overhead. The secondary contribution is the *evaluation of scheduling co-allocation requests considering both user-estimated and actual runtimes, as well as response time guarantees for local and multi-site requests* (Section 5). Current research on co-allocation assumes accurate estimation of application runtimes and does not provide users with response time guarantees once they receive their scheduling time slots.

We have evaluated our model and scheduling strategies with extensive simulations and analyzed several metrics to have a better understanding of the improvements achieved here. We also discuss issues on deploying the model on real environments. It is important to highlight that some of the ideas proposed in this work can be directly applied to the co-allocation of other resources such as network links, as well as scheduling of parallel phases in workflows.

2 Problem Description

A metascheduler books resources across multiple autonomous sites to execute parallel jobs. Each site has its own scheduling queue and policies to manage both local and external requests. As resource providers rely on inaccurate runtime estimations, they must update their queues to produce better schedules. Therefore, they may also need to modify parts of a co-allocation request, named *sub-requests* or *sub-jobs*. However, all these sub-requests must be synchronized, i.e. starting at the same time, otherwise the parallel applications cannot be executed.

Computing environment. 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 processors, $R = \{r_1, r_2, ...r_n\}$ where n is the total number of processors in a given machine m_i. For simplicity, we assume that all the processors R in a given machine m_i are homogeneous—which is a reasonable assumption considering that most of the parallel machines are composed of homogeneous processors. The machines in M can be heterogeneous. We consider that there is a network interconnecting these machines, which can be either exclusive or shared in an open environment such as the Internet.

Resource Management Systems. These systems, also named *local schedulers*, schedule both local and external requests in a machine m_i. We do not assume that a metascheduler has complete information about the local schedulers. In our scenario, rather than publishing the complete scheduling queue to the metascheduler, the local schedulers may want to only publish certain time slots to optimize 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 FIFO with conservative backfilling, which provides completion time guarantees once users receive their scheduling time slots [20].

Application model. We investigate resource co-allocation for *parallel applications* requiring simultaneous access to resources from multiple sites. We consider applications that are mainly compute-intensive. Data-intensive applications have different requirements, and therefore we do not consider them in this work. 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. This is mainly required by parallel applications with data exchange among the processes. These applications have a delay when using inter-cluster communication. The metascheduler decomposes a request to execute a parallel application into k sub-requests, where each sub-request is sent to a machine m_i. Note that, in some cases, the user may want to incorporate some constraints to decompose the request.

Metrics. Our main aim is to optimize job response time, i.e. the difference between the submission time of the user request and its completion time. We also evaluate system utilization, number of machines used by each job, number of jobs that received resources before expected, among other metrics.

3 Flexible Resource Co-Allocation Model

The flexible resource co-allocation (FlexCo) model proposed here is inspired by existing work on flexible advance reservations [12, 16, 21, 22, 25, 26]. A request, or a *multi-site job*, following this model can have relaxed start and completion times, and the flexibility to define the number of processors used in each machine. A FlexCo request is composed of sub-requests that are submitted to different machines. Each sub-request may have a different number of resources with different capabilities. The following parameters and notations represent a multi-site job j based on the FlexCo model:

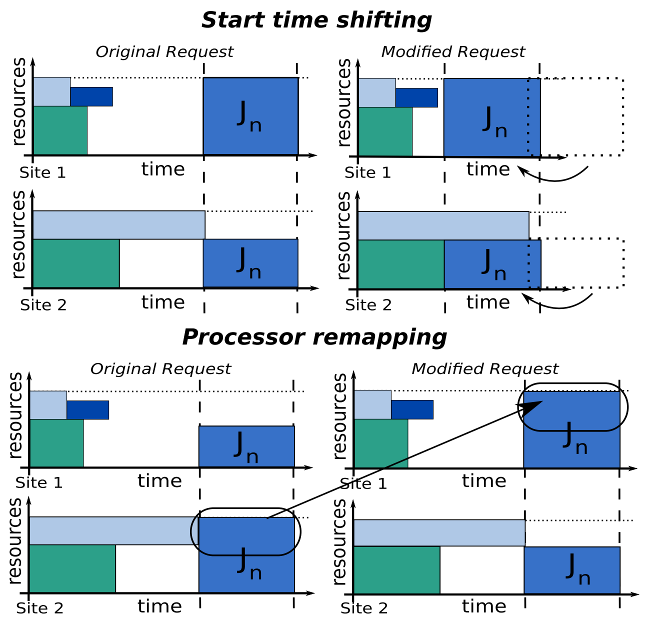

Figure 1. Operations of a FlexCo request.

- $R_j^{m_k}$: number of processors required in each machine m_i, where k is the total number of sub-requests of the job j;
- T_j^s: job start 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 start 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.

A FlexCo request has two operations (Figure 1): (i) *Start time shifting:* changes the start time according to the relaxed time interval—the change must be the same for all sub-requests; and (ii) *Processor remapping:* changes the number of required resources of two or more sub-requests. Combining both operations is also important for the scheduler. In Figure 1, we observe that after using the processor remapping operation, it is possible reduce job response time by shifting the sub-requests. Note that the schedulers perform these operations while jobs are waiting for resources, and not during runtime. The idea here is to redefine the request specifications, not to migrate jobs.

Start Time Shifting (Shift): finding a common time slot may be difficult for users, hence once they commit the co-allocation based on advance reservations, they will not be willing to change it. The modification of the start 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. Note that this operation is not application dependent in the sense that it is only a shift on the start time of the user application.

Processor Remapping (Remap): A user requiring a certain number of resources tends to decompose the request statically according to the available providers at a certain time. Therefore users may not be able to reduce the start time of their applications when resources become available. To overcome this problem, *Remap* allows automatic remapping of the processors once the sub-requests are queued. This operation is application dependent since the throughput offered by each resource provider may influence the overall application performance. Thus, users may also want to incorporate restrictions on how the metascheduler should map and remap their requests. Branch-and-bound-based solvers for optimization problems are an example of application that is flexible to deploy and hence can have benefits from this operation. For network demanding applications, this operation allows the reduction of the number of resource providers required by a co-allocation request, which has a direct impact on the network utilization.

4 Scheduling of Multi-Site Requests

The scheduling of a multi-site request consists in finding a free common time slot that meets the job requirements in a set of machines. We consider the scheduling to be *on-line*, where users submit jobs to resource providers over time and their schedulers make decisions based on only currently accepted jobs. The scheduling involves the manipulation of time slots, which are data structures composed of four values: (i) ts^{id}: identification; (ii) ts^s: start time; (iii) ts^c: completion time; and (iv) ts^n: number of resources available in this time slot.

4.1 Initial scheduling

The metascheduler performs the initial scheduling of an external request by following these four steps:

1. Ask the resource providers for the list of available time slots, $TS = \{ts_1, ts_2, ..., ts_n\}$, where n is the number of time slots.

2. Find the earliest common start time T_j^s that meets the request constraints, such as number of resources, start time, and completion time.

3. Generate a list of sub-requests.

4. Submit the sub-requests to the resource providers accordingly.

In order to find the common start time T_j^s, the metascheduler verifies the values of T_j^s 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 T_j^s 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 to a single machine m_i.

When generating the list of sub-requests, the metascheduler could follow different approaches. For example, it could try to decompose the multi-site jobs evenly 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 the second approach is to increase the chances of fitting some multi-site jobs in a single site over time due to the rescheduling.

4.2 Rescheduling

As described in the previous subsection, the initial scheduling of a multi-site job involves manipulation and transfer of time slots over the network. In order to reschedule multi-site jobs, one must consider the cost-benefit of transferring and manipulating time slots to optimize the schedule. Therefore, our approach is to reschedule a multi-site job only when the resource provider is not able to find a local job that fills the fragment generated due to the early completion of a job. The local schedulers use Algorithm 1 to reschedule jobs whenever a job completes before its estimated time. The rescheduling is based on the *compressing* method described by Weil and Feitelson [20], which consists in bringing the jobs to the current time according to their estimated start times, not their arrival times (Lines 3-5, 11-14). 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 jobs sorted according to start time instead of sorting them when rescheduling (Line 2).

Algorithm 1: Pseudo-code for rescheduling jobs, which is executed on the local schedulers when a job completes before the expected time.

```
1  $coallocRescheduled \leftarrow false$
2  Sort $Q^w \mid \{T_1^s \leq T_2^s ... \leq T_n^s\}$, where $n$ is number of jobs in the waiting queue
3  for $\forall j_i \in Q^w$ do
4      if $j_i$ is local job then
5          Schedule job with backfilling
6  while there are idle resources do
7      for $\forall$ multisite jobs $j_i$ in $Q^w$ do
8          Contact metascheduler to reschedule $j_i$
9          if $T_{j_i}^c \leq previousT_{j_i}^c$ then
10             $coallocRescheduled \leftarrow true$
11 if $coallocRescheduled = true$ then
12     for $\forall j_i \in Q^w$ do
13         if $j_i$ is local job then
14             Schedule job with backfilling
```

Once the metascheduler receives a notification for rescheduling a multi-site job j_i from the resource provider (Line 8), it performs the rescheduling in a similar way as described in the initial scheduling procedures (Section 4.1). The main differences are that (i) for the **Shift** operation, the metascheduler asks for time slots only from those resource providers which hold the sub-requests of the multi-site job j_i; and (ii) for the **Remap** operation the metascheduler contacts other resource providers rather than only the original ones. In addition, for this latter operation, the metascheduler may remove sub-requests from resource providers.

When deploying the model, a service, which we call here the metascheduler, has to be available to access the information of multi-site jobs such as total number of required processors and location of resource providers holding the sub-requests. An alternative is to associate to each sub-request a list of the resource providers holding the other sub-requests. The first approach brings the simplicity to the middleware of the local schedulers since they need to negotiate and keep track of only a single entity, i.e. the metascheduler. However, such a centralized entity becomes a bottleneck. The second approach has opposite advantages and drawbacks.

5 Evaluation

In order to see the effects of a long-term usage of our model, we decided to evaluate it by means of simulations. We have used our event-driven simulator, named PaJFit (Parallel Job Fit) [22], which we have extended to support multi-site environments and FlexCo requests. We used real traces from supercomputers available at the Parallel Workloads Archive[2] to model the user applications. We compared the use of Shift and Shift with Remap operations against the co-allocation model based on rigid advance reservations, which provides response time guarantees but suffers from high fragmentation inside the resource provider's scheduling queues. This section presents a detailed description of the environment setup and metrics followed by the results and our analysis.

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

Table 1. Summary of workloads.

Location	Procs	Jobs	Actual — Estimated Load
Cluster 1	430	10,757	60% — 225%
Cluster 2	240	4,722	65% — 138%
Cluster 3	128	4,686	88% — 376%
External	798	170	10% — 35%
External	798	339	30% — 77%

5.1 Experimental configuration

We modeled an environment composed of 3 clusters with their own scheduler and load, and one metascheduler which receives external jobs that can be executed in either a single or multiple clusters. For the local jobs we used the traces: 430-node IBM SP2 from The Cornell Theory Center (CTC SP2v2.1), 240-procs AMD Athlon MP2000+ from High-Performance Computing Center North (HPC2N v1.1) in Sweden, 128-node IBM SP2 from The San Diego Supercomputer Center (SDSC SP2 v3.1). For the external jobs we used the trace of a larger machine, the San Diego Supercomputer Center Blue Horizon with 1,152 processors: 144-node IBM SP, with 8 processors per node, considering jobs requiring at least 128 processors (SDSC BLUE v3.1). We simulated 45 days of these traces. In order to evaluate our model under different conditions, we varied the external load on the system by changing the arrival times of the external jobs.[3] Table 1 summarizes the workload characteristics. We can observe that the estimated load is much higher than the actual load due to the wrong user estimations. More details on the workloads can be found at the Parallel Workloads Archive.

For the network overhead of multi-site jobs, as there is no trace available with such information, we have assigned to each job a random value defined by a Poisson distribution with λ=20. A study by Ernemann et al. [11] shows that co-allocation is advantageous when the penalty for network overhead is up to approximately 25%. Therefore, we limited the network overhead under this value.

We evaluated the system utilization and response time, i.e. the difference between the job completion time and submission time. In addition, we analyzed the behavior of multi-site jobs due to the rescheduling. We investigated these metrics according to the runtime estimation precision of all jobs in the system, which we varied from the original value defined in the workload to 20% plus the original value: e.g. if a job requests 10 minutes and it takes 6 minutes, we evaluate the execution with 6, 7, and 8 minutes. The metrics are also a function of the external load.

[3]To vary the load we used a strategy similar to that described by Shmueli and Feitelson to evaluate their backfilling strategy [27], but we fixed the time interval and included more jobs from the trace.

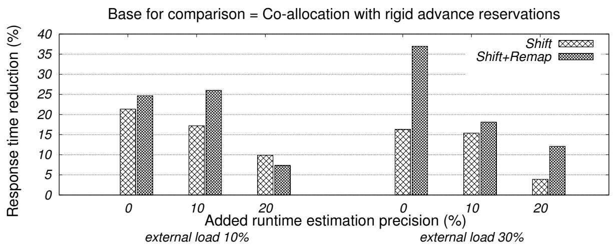

Figure 2. Response time reduction of local jobs as a function of runtime estimation precision and external load.

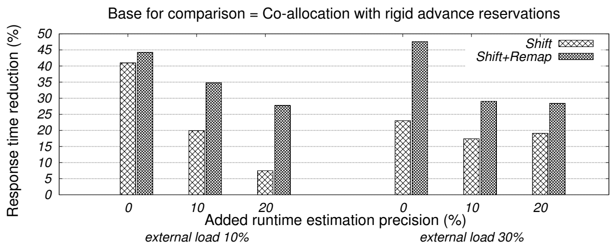

Figure 3. Response time reduction of external jobs as a function of runtime estimation precision and external load.

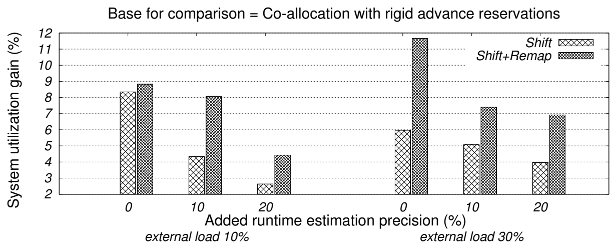

Figure 4. Global utilization gain as a function of runtime estimation precision and external load.

5.2 Results and analysis

In order to evaluate the response time, we have separated the results for local and external jobs, Figures 2 and 3 respectively. We observe that rescheduling multi-site jobs brings benefits for both local and external jobs. Local jobs have more benefit because they can better fill the gaps in the head of the scheduling queue due to their characteristics, i.e. many jobs required less time and fewer processors. By using co-allocation based on rigid advance reservations, some local jobs may not be placed at the head of the scheduling queue since they would overlap with the advance reservations. Therefore, the benefit for local jobs is mainly due to those jobs that had this problem, and therefore, by shifting the multi-site jobs, some local jobs could be shifted as well. We can also observe from these results that in most scenarios, Shift+Remap provides better schedules than only the Shift operation. That is because of the higher flexibility

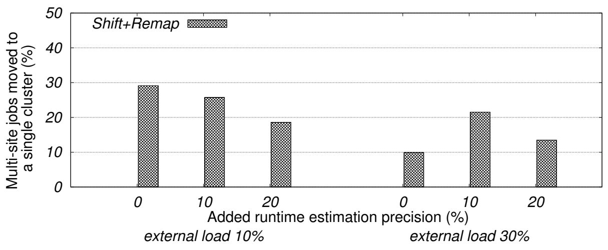

Figure 5. Percentage of multi-site jobs moved to a single cluster as a function of runtime estimation precision and external load.

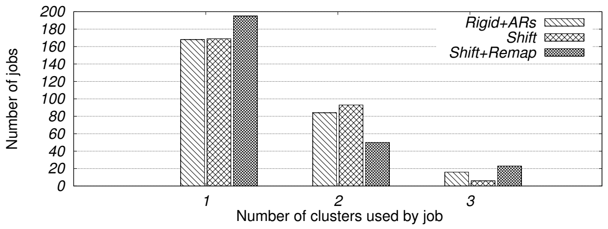

Figure 6. Number of clusters used by job with system external load=10%.

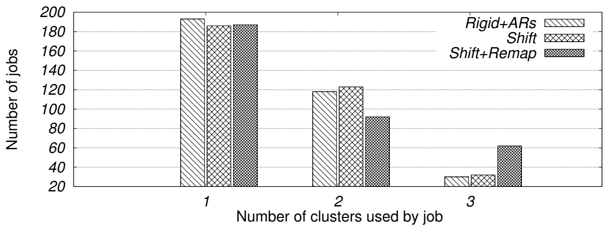

Figure 7. Number of clusters used by job with system external load=30%.

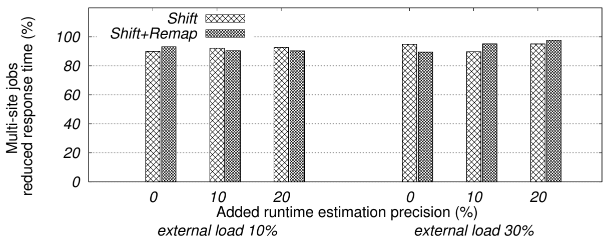

Figure 8. Percentage of multi-site jobs that reduce their response time due to rescheduling as a function of runtime estimation precision and external load.

the Shift+Remap gives to the scheduler and by the fact that some multi-site jobs can be remapped to a single cluster.

Filling the gaps using FlexCo requests has a direct impact on the system utilization, as can be observed in Figure 4. For system utilization we can see that Shift+Remap consistently provides better results than only shifting the requests, reaching its peek in an improvement of over 10% in relation to co-allocation based on rigid advance reservations. For both response time and system utilization we observe that the higher the imprecision on runtime estimations, the better the benefit of rescheduling multi-site jobs.

To better understand what happens with the multi-site jobs, we measured the number of jobs remapped to a single cluster due to the rescheduling. From Figure 5 we observe that approximately 20% of multi-site jobs, that otherwise would use inter-cluster communication, were migrated to a single cluster. Different from the utilization and response time, this metric does not present a smooth behavior. That is because moving jobs to a single site is highly dependent on the characteristics and packing of the jobs.

Figure 6 and 7 illustrate the total number of clusters used by the external jobs. We observe that when the external load is only 10% more jobs can be moved to a single cluster. However, when the system has a higher load, it is more difficult to find big spaces in a single cluster. Moreover, in this second case, multi-site jobs may end up accessing fragments of more sites to reduce their response time.

Figure 8 illustrates the percentage of multi-site jobs that were initially submitted to more than one site and that were able to access the resources before expected due to rescheduling. We observe that this improvement occurs for almost all multi-site jobs for all scenarios. Both operations helped to improve the schedule of multi-site jobs, however, as we have already showed, Shift+Remap provides a higher impact on the improvement.

The experimental results presented in this section demonstrate that local jobs are not able to fill all the fragments in the scheduling queues and therefore co-allocation jobs need to be rescheduled. The more the users are imprecise with their estimations the more important is the rescheduling. That is because the need for the rescheduling increases with the number and size of the fragments generated by the wrong estimations in the head of the scheduling queues.

6 Related Work

Czajkowski et al. [6] focus on failures of co-allocation requests. 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. [7] propose an approach in which users could modify their co-allocation requests via *add*, *delete*, and *substitute* operations. Resources could be classified in categories, *required*, *interactive* and *optional*, in order to simplify the management of resource failures. Their work does not consider the rescheduling of multi-site jobs without user interaction.

Foster et al. [13] propose the Globus Architecture for Reservation and Allocation (GARA), which aims to provide a platform with support for quality of service guarantees through advance reservations. Their work focuses more on middleware aspects rather than on scheduling optimizations.

Alhusaini et al. [1,2] investigate the mapping of independent tasks requiring co-allocation on distributed resources in order to minimize schedule length. Their solution is based on a two-phase approach. The first phase is off-line planning where the scheduler assigns tasks to resources assuming that all applications hold all the required resources for their entire execution. The second phase is run-time adaptation where the scheduler makes decisions according to the actual computation and communication costs, which may be different from the estimated costs used in the first phase. Applications may release some resources before the completion of the execution. Similar to our work, they consider the wrong estimation of job requirements and the need of a rescheduling phase to overcome this problem. However, 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.

MacLaren et al. [18] discuss the problem of resource co-allocation, in particular focusing on fault tolerance, and propose a 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 to the scheduling queue of the resource managers but asks for free time slots. The system supports the creation of a reservation, cancellation, modification of number of requested CPUs and time of the reservation. Nevertheless, they do not address issues such as finding an optimal schedule or managing the reservations once they have been made. Therefore, we see HARC as a middleware that 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 advance reservations. 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 slowest running subtask, may be long. Therefore, resources would not be effectively utilized. In addition, depending on the application and computing environment, it is not possible to modify the execution speed of sub-tasks. The strategies presented in this paper can overcome the limitations Azzedin et al. mentioned about using advance reservations for co-allocation.

Bucur and Epema [4] investigate scheduling policies and different queuing structures for resource co-allocation in multi-cluster systems. They evaluate the differences of having a single global scheduler, only local schedulers and both structures together, as well as different priorities for local and external. Their work does not use advance reservations. Therefore, it is not possible to give completion time guarantees 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 shown the impact of rescheduling co-allocation requests in environments where resource providers deal with inaccurate runtime estimations. As local jobs are not able to fill all the fragments in the scheduling queues, the co-allocation requests should not be based on rigid advance reservations. Our flexible co-allocation (FlexCo) model relies on shifting of advance reservations and processor remapping. These operations allow the rescheduling of co-allocation requests, therefore overcoming the limitations of existing solutions in terms of response time guarantees and fragmentation reduction.

Regarding the rescheduling operations, Shift provides good results against the rigid-advance-reservation-based co-allocation and is not application dependent since it only changes the start time of the applications. Shift with Remap provides even better results but is application dependent since it also modifies the amount of work submitted to each site. Parallel applications that have flexible deployment requirements, such as branch-and-bound-based solvers for optimization problems, can have benefits from the Remap operation. In our experiments we showed that depending on the system load, Remap can reduce the number of clusters used by multi-site requests. In the best case, a job initially mapped to multiple sites can be remapped to a single site, thus eliminating unnecessary network overhead, which is important for network demanding parallel applications.

As future work we will explore the use of system-generated predictions rather than user runtime estimates [23, 31]. As user estimations are usually very imprecise, system-generated predictions may reduce the need for rescheduling. However, this hypothesis needs to be verified. We will also consider requests with underestimated usage times.

Acknowledgments

We would like to thank Marcos Dias de Assunção, SungJin Choi, Chee Shin Yeo, and Srikumar Venugopal for their comments on the paper. This work is partially supported by research grants from the Australian Research Council (ARC) and Australian Department of Innovation, Industry, Science and Research (DIISR).

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 *Proc. of the HCW, in conj. with the 14th IPDPS*, pages 273–286, 2000.

[2] A. H. Alhusaini, C. S. Raghavendra, and V. K. Prasanna. Run-time adaptation for grid environments. In *Proc. of the HCW, in conj. with the 15th IPDPS*, pages 864–874, San Francisco, USA, 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 Trans. on Paral. and Dist. Sys.*, 18(7):958–972, 2007.

[5] N. Capit, G. D. Costa, Y. Georgiou, G. Huard, C. Martin, G. Mounié, P. Neyron, and O. Richard. A batch scheduler with high level components. In *Proc. of the 5th International Symposium on Cluster Computing and the Grid (CCGrid)*, pages 776–783, Cardiff, UK, 2005.

[6] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In *Proc. of the 4th JSSPP*, pages 62–82, Orlando, USA, 1998.

[7] K. Czajkowski, I. Foster, and C. Kesselman. Resource co-allocation in computational grids. In *Proc. of the 8th HPDC*, pages 219–228, Redondo Beach, USA, 1999.

[8] J. Decker and J. Schneider. Heuristic scheduling of grid workflows supporting co-allocation and advance reservation. In *Proc. of the 7th CCGrid*, pages 335–342, Rio de Janeiro, Brazil, 2007.

[9] S. Dong, G. E. Karniadakis, and N. T. Karonis. Cross-site computations on the TeraGrid. *Computing in Science and Engineering*, 7(5):14–23, 2005.

[10] 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/et_jss.pdf`. 2007.

[11] C. Ernemann, V. Hamscher, U. Schwiegelshohn, R. Yahyapour, and A. Streit. On advantages of grid computing for parallel job scheduling. In *Proc. of the 2nd CCGrid*, pages 39–, Berlin, Germany, 2002.

[12] 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 *Proc. of the 35th ICPP*, pages 373–380, Columbus, USA, 2006.

[13] 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 *Proc. of the 7th IWQoS*, pages 27–36, 1999.

[14] W. M. Jones, W. B. L. III, L. W. Pang, and D. C. S. Jr. Characterization of bandwidth-aware meta-schedulers for co-allocating jobs across multiple clusters. *The Journal of Supercomputing*, 34(2):135–163, 2005.

[15] N. Kaushik, S. Figueira, and S. A. Chiappari. Resource co-allocation using advance reservations with flexible time-windows. *SIGMETRICS Perform. Eval. Rev.*, 35(3):46–48, 2007.

[16] N. R. Kaushik, S. M. Figueira, and S. A. Chiappari. Flexible time-windows for advance reservation scheduling. In *Proc. of the 14th MASCOTS*, Monterey, USA, 2006.

[17] C. B. Lee and A. Snavely. On the user-scheduler dialogue: Studies of user-provided runtime estimates and utility functions. *International Journal of High Performance Computing Applications*, 20(4):495–506, 2006.

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

[19] H. H. Mohamed and D. H. J. Epema. Experiences with the KOALA co-allocating scheduler in multiclusters. In *Proc. of the 5th CCGrid*, Cardiff, UK, 2005.

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

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

[22] M. A. S. Netto, K. Bubendorfer, and R. Buyya. SLA-based advance reservations with flexible and adaptive time QoS parameters. In *Proc. of the 5th ICSOC*, pages 119–131, Vienna, Austria, 2007.

[23] D. Nurmi, J. Brevik, and R. Wolski. Qbets: queue bounds estimation from time series. In L. Golubchik, M. H. Ammar, and M. Harchol-Balter, editors, *SIGMETRICS*, pages 379–380. ACM, 2007.

[24] T. Röblitz and A. Reinefeld. Co-reservation with the concept of virtual resources. In *Proc. of the 5th CCGrid*, pages 398–406, Cardiff, UK, 2005.

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

[26] T. Röblitz, F. Schintke, and J. Wendler. Elastic grid reservations with user-defined optimization policies, 2004.

[27] E. Shmueli and D. G. Feitelson. Backfilling with lookahead to optimize the packing of parallel jobs. *J. Parallel Distrib. Comput.*, 65(9):1090–1107, 2005.

[28] Q. Snell, M. J. Clement, D. B. Jackson, and C. Gregory. The performance impact of advance reservation meta-scheduling. In *Proc. of the 6th JSSPP*, pages 137–153, Cancun, Mexico, 2000.

[29] H. Takemiya, Y. Tanaka, S. Sekiguchi, S. Ogata, R. K. Kalia, A. Nakano, and P. Vashishta. Sustainable adaptive grid supercomputing: multiscale simulation of semiconductor processing across the pacific. In *Proc. of the ACM/IEEE SC'06*, page 106, Tampa, USA, 2006.

[30] D. Tsafrir, Y. Etsion, and D. G. Feitelson. Modeling user runtime estimates. In *Proc. of the 11th JSSPP*, pages 1–35, 2005.

[31] D. Tsafrir, Y. Etsion, and D. G. Feitelson. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Trans. Parallel Distrib. Syst.*, 18(6):789–803, 2007.

[32] K. Yoshimoto, P. A. Kovatch, and P. Andrews. Co-scheduling with user-settable reservations. In *Proc. of the 11th JSSPP*, pages 146–156, Cambridge,USA, 2005.