Adaptive Co-Allocation of Distributed Resources for Parallel Applications

by

Marco Aurélio Stelmar Netto

Submitted in total fulfilment of
the requirements for the degree of

Doctor of Philosophy

Department of Computer Science and Software Engineering
The University of Melbourne, Australia

November 2009

Adaptive Co-Allocation of Distributed Resources for Parallel Applications

Marco Aurélio Stelmar Netto

Supervisor: Prof. Rajkumar Buyya

Abstract

Parallel applications can speed up their execution by accessing resources hosted by multiple autonomous providers. Applications following the message passing model demand processors to be available at the same time; a problem known as resource co-allocation. Other application models, such as workflows and bag-of-tasks (BoT), can also benefit from the coordinated allocation of resources from autonomous providers. Applications waiting for resources require constant rescheduling. However, different from single-provider settings, rescheduling across multiple providers is challenging due to the autonomy and information availability participants are willing to disclose.

This thesis contributes to the area of distributed systems by proposing adaptive resource co-allocation policies for message passing and BoT applications, which aim at reducing user response time and increasing system utilisation. For message passing applications, the co-allocation policies rely on start time shifting and process remapping operations, whereas for BoT applications, the policies consider limited information access from providers and coordinated rescheduling. This thesis also shows practical deployment of the co-allocation policies in a real distributed computing environment. The four major findings of this thesis are:

1. Adaptive co-allocation for message passing applications is necessary since single-cluster applications may not fill all scheduling queue fragments generated by inaccurate run time estimates. It also allows applications to be rescheduled to a single cluster, thus eliminating inter-cluster network overhead;

2. Metaschedulers using system-generated run time estimates can reschedule applications to faster or slower resources without forcing users to overestimate execution times. Overestimations have a negative effect when scheduling parallel applications in multiple providers;

3. It is possible to keep information from providers private, such as local load and total computing power, when co-allocating resources for deadline-constrained BoT applications. Resource providers can use execution offers to advertise their interest in executing an entire BoT or only part of it without revealing private information, which is important for companies to protect their business strategies;

4. Tasks of the same BoT can be spread over time due to inaccurate run time estimates and environment heterogeneity. Coordinated rescheduling of these tasks can reduce response time for users accessing single and multiple providers. Moreover, accurate run time estimates assist metaschedulers to better distribute tasks of BoT applications on multiple providers.

This is to certify that

(i) the thesis comprises only my original work,

(ii) due acknowledgement has been made in the text to all other material used,

(iii) the thesis is less than 100,000 words in length, exclusive of table, maps, bibliographies, appendices and footnotes.

Signature_____

Date_____

ACKNOWLEDGMENTS

I thank my supervisor, Professor Rajkumar Buyya, who gave me the opportunity to conduct this research in the area of distributed systems, and for the constant encouragement and suggestions throughout my PhD candidature. In particular, I appreciated the freedom and independence I experienced during the work in his research group.

I would also like to thank all past and current members of the CLOUDS Laboratory, at the University of Melbourne. In particular, I thank Marcos dias de Assunção, Mukaddim Pathan, Christian Vecchiola, Alexandre di Costanzo, Chee Shin Yeo, Srikumar Venugopal, Rajiv Ranjan, Krishna Nadiminti, Hussein Gibbins, Jia Yu, Mustafizur Rahman, James Broberg, Xingchen Chu, Sungjin Choi, Suraj Pandey, Kyong Hoon Kim, Saurabh Garg, Anton Beloglazov, William Voorsluys, Mohsen Amini, Chao Jin, Rodrigo Calheiros, Nithiapidary Muthuvelu, Dileban Karunamoorthy, Amir Vahid, and Adam Barker for their friendship and help during my PhD. I also thank Rao Kotagiri, Adrian Pearce, Carlos Varela, Kris Bubendorfer, Michael Kirley, and Carlos Queiroz for their discussions during the development of my thesis. Several experiments I present in this thesis were conducted in Grid'5000, a large-scale computing facility in France. I thank the excellent support from the Grid'5000 team. I am also grateful to Dr. Dror Feitelson for maintaining the Parallel Workload Archive, and all organisations and researchers who made their workload logs available. I extend my gratitude to the staff members and friends from the CSSE Department for their help and support. I also thank the anonymous reviewers who evaluated the content of this thesis.

I would like to express my deep gratitude to my previous supervisors Cesar A. F. De Rose, Osmar Norberto de Souza, Avelino Zorzo, and Alfredo Goldman for helping me during my first steps as a researcher. I also thank Angela Morgan, Carly Fatnowna, Stacy Barelos, Andreas Schutt, Jason Lee, Ziyuan Wang, Ben Horsfall, Trevor Hansen, Khalid Aljasser, and all members of the Melbourne Portuguese Speakers Meetup Group for their constant encouragement and friendship.

I acknowledge the University of Melbourne and the Australian Government for providing me with scholarships to pursue my doctoral studies. I also thank the support received from the Australian Research Council (ARC) and Australian Department of Innovation, Industry, Science and Research (DIISR) at various times during my candidature. Without their support, I would not have been able to carry out this research.

Last but never the least, I thank my family for their love and support at all times. My mother, father, and sister have always been a source of inspiration during my whole life.

Marco A. S. Netto
Melbourne, Australia
November 2009

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

The Ghan[1] is a passenger train that operates between Darwin to Adelaide in Australia. A person who departs from Melbourne needs to book the following: a flight ticket from Melbourne to Darwin, and another ticket from Adelaide to Melbourne, accommodation in Darwin and Adelaide; and naturally, the train ticket (Figure 1.1). When booking these services, one can either contact an agency or book each service directly. The process of allocating these services in a coordinated manner is called *co-allocation* or *co-scheduling*. Apart from co-allocating these services, one must take into account possible change of plans, i.e. rescheduling the bookings. The challenge is that changing a single booking may affect all the other bookings due to their interdependency. Co-allocation is also necessary in many other activities of our daily lives. For example, when scheduling a meeting with multiple participants, it is necessary to make sure all participants can attend the meeting at a specified time, and that a room and additional resources, such as a projector, are available. In the digital world, several software systems also require co-allocation of multiple components to execute properly, and most importantly, co-allocation decisions may change over time to meet user requirements.

Distributed computing [117] is a field of Computer Science that investigates systems consisting of multiple components over a computer network. These components, which can be software or hardware, interact with one another via a communication protocol, i.e. all components communicate at the same time or in a sequence. The complexity of component interactions depends on the scale of the system. Large-scale systems, with hundreds or thousands of resources, tend to have more complex communication protocols in order to handle heterogeneous components and failures in hardware and software layers.

Large-scale distributed computing systems gained attention in the 1990s due to the

[1]The Ghan Website: http://www.gsr.com.au

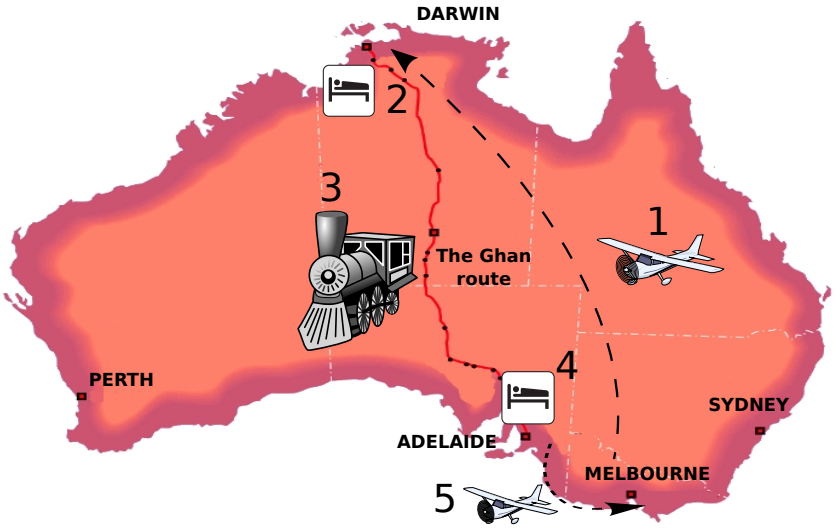


Figure 1.1: Booking services for a trip.

growth of the Internet, the popularity of clusters composed of thousands of computers, and efforts in Grid Computing [50, 52]. Clusters [23, 126], which are groups of computers that work to act as a single machine, became available in several universities and institutes in the 90s. One of the goals of Grid Computing has been to interconnect these clusters from autonomous sites in order to speed up executions of large-scale applications. More recently, Cloud Computing [6, 24] has become an important platform for large-scale computations. Cloud Computing is based on the utility computing paradigm [95], in which resources are delivered as services in a pay-as-you-go manner, whereas Grid Computing is a more collaborative distributed platform.

The two main reasons for executing applications on multiple sites are: the lack of special resources in a single administrative domain, such as scientific instruments, visualisation tools, and supercomputers; and the possibility of reducing response time of parallel applications by increasing the number of resources [34, 112] (Figure 1.2). There are also other applications that require resources from multiple sites. Conference and multimedia users engaged in activities, such as scientific research, education, commerce, and entertainment, require multi-party real-time communication channels [49, 128]. Data-intensive applications can collect data from multiple sources in parallel [121, 129]. In addition, increasing the number of resources is a requirement of applications demanding considerable amounts of memory, storage, and processing power. Examples of these applications are semiconductor processing [116] and computational fluid dynamics [43].

Efforts in developing resource co-allocation protocols increased in the late 90s [34, 35], mainly to execute applications with inter-process communication developed with Message Passing Interface (MPI) [54]. Most MPI applications require all processors to be

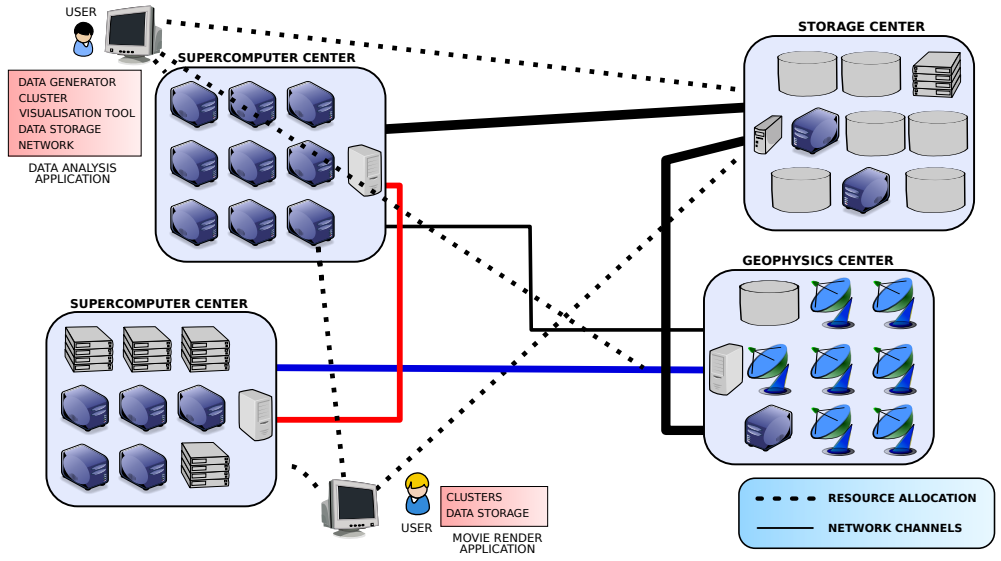


Figure 1.2: Example of two applications requiring resources from multiple sites.

available at the same time and the unavailability of a single processor can compromise the entire application. Therefore, in order to execute large-scale MPI applications, researchers developed policies to allocate resources from multiple sites in a coordinated manner. To co-allocate resources, users rely on metaschedulers, which are responsible for generating co-allocation requests that are submitted to management systems of each site. These requests, also known as *jobs*, represent the amount of time and number of processors required by the user application. In order to guarantee that all jobs start at the same time, the metascheduler allocates resources using *advance reservations* [51].

Recently, several researchers investigated resource co-allocation for workflow applications [134]. Workflows are applications composed of tasks that have time dependencies and control flow specifications; a process cannot start if its input is not available or a set of conditions are not satisfied. Therefore, resources that execute a workflow have to be available in a coordinated manner, usually in a sequence. However, in the literature, resource co-allocation for workflows is usually referred as *workflow scheduling* [41, 134].

Bag-of-Tasks (BoT) applications [27, 33] are another application model that requires co-allocation, but with different constraints. These applications have been used in several fields including computational biology [92], image processing [110], and massive searches [5]. In comparison to the message passing model, BoT applications can be easily executed on multiple resource providers to meet a user deadline or reduce the user response time. Although BoT applications comprise independent tasks, the results produced by all tasks constitute the solution of a single problem. In most cases, users need the whole set of tasks executed in order to post-process or analyse the results. The optimisation of the aggregate set of results is important, and not the optimisation of a particular

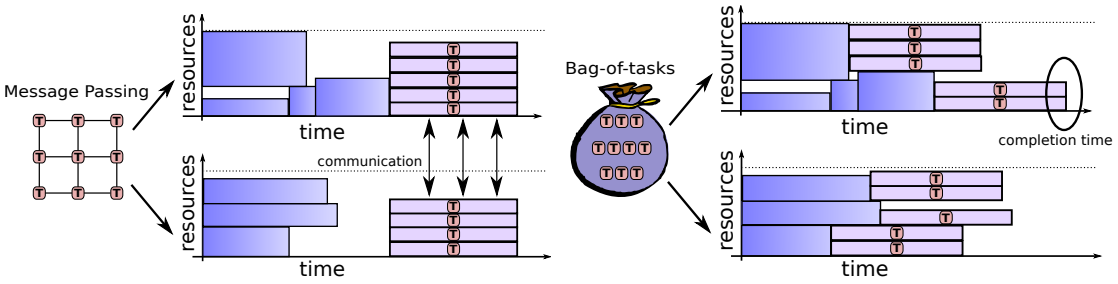
task or group of tasks [8]. Therefore, message passing, workflow, and bag-of-tasks applications have different resource co-allocation requirements.

Various projects have developed software systems with resource co-allocation support for large-scale computing environments, such as TeraGrid, Distributed ASCI Supercomputer (DAS), and Grid'5000. TeraGrid has deployed Generic Universal Remote (GUR) [133] and Highly-Available Resource Co-allocator (HARC) [78], the DAS project has developed KOALA [83, 84], and Grid'5000 [18] has relied on the OAR(Grid) scheduler [25] to allow the execution of applications requiring co-allocation. For these systems, co-allocation is mostly performed for applications that require simultaneous access to resources from several sites. For workflows, researchers rely on workflow engines, which are middleware systems to schedule workflow applications. In Cloud Computing space, initiatives such RESERVOIR [100] are emerging to co-allocate resources from multiple commercial data centers. These centers keep important information required by metaschedulers to co-allocate resources, such as scheduling policies, local load, and total computing capabilities.

1.1 Resource Co-Allocation and Rescheduling

This thesis focuses on resource co-allocation for message passing and bag-of-tasks application models (Figure 1.3). The former model requires all resources to be available at the same time, and hence advance reservations are important building blocks for co-allocation policies under this application model. However, the use of advance reservations increases fragmentation in the scheduling queues, thus reducing system utilisation. On the other hand, bag-of-tasks do not require all tasks to start at the same time, and therefore, tasks of the same application tend to complete at different times.

Most of the current research on resource co-allocation focuses on the initial scheduling. Once jobs are distributed among providers, they remain there until completion. However, run time estimations of user applications are usually inaccurate [72], and hence jobs completing before the estimated time create fragments in the scheduling queue. Initial schedules then need to be adapted to fill these fragments in order to reduce user response time and increase system utilisation. For message passing model, the advance reservations have to be modified such that they start at the same time. For bag-of-tasks model, tasks can be rescheduled as long as the completion time of the last task is minimised. Understanding the impact of rescheduling these applications on multiple autonomous providers is an open question that this thesis addresses. Note that, different from single-provider settings, rescheduling across multiple providers is challenging due to the autonomy and information availability participants are willing to disclose.



(a) Message Passing Application. (b) Bag-of-Tasks Application.

Figure 1.3: Co-allocation for message passing and bag-of-tasks applications.

1.2 Research Question and Objectives

The research question addressed in this thesis is:

"What are the benefits for users and resource providers when rescheduling message passing and bag-of-tasks applications on multiple autonomous providers?"

The following objectives are requirements to answer the thesis research question:

- Understand the impact of inaccurate run time estimates in computing environments with applications co-allocating resources from multiple providers;
- Design, implement, and evaluate co-allocation policies with rescheduling support;
- Investigate technical difficulties to deploy the co-allocation policies in real environments.

To meet the above objectives, this thesis proposes resource co-allocation policies that consider the following aspects:

- **Inaccurate run time estimations of user applications:** it is well-known that users cannot estimate their application run time precisely. Therefore, it is important to handle these inaccurate estimations when scheduling applications;
- **Completion time guarantees:** users can better plan other activities, such as result analysis, when they have precise estimation of their application completion time, in particular when an application is part of a workflow. Thus, resource providers and metaschedulers have an important role in generating completion time estimations;
- **Coordinated rescheduling:** rescheduling is necessary mainly due to inaccurate run time estimates of user applications. However, a rescheduling decision in a site may also affect other sites;

- **Limited information access:** resource providers may want to keep their load and total computing power private, especially in utility computing facilities. Therefore, co-allocation in these environments is more difficult due to the limited information access.

1.3 Contributions and Main Findings

Considering the objectives described in the previous section, the major contributions of this thesis are:

1. **A detailed study on flexible advance reservations.** Advance reservations are building blocks of resource co-allocation for message passing applications. We show the importance of rescheduling advance reservations for system utilisation using four scheduling heuristics under several workloads, reservation time intervals and inaccurate run time estimates. Moreover, we investigate cases when users accept an alternative offer from the resource provider on failure to schedule the initial request.

   The main finding is that system utilisation increases with the flexibility of request time intervals and with the time users allow this flexibility while they wait for resources. This benefit is mainly due to the ability of the scheduler to rearrange the jobs waiting for resources, which in turn reduces the fragmentation generated by advance reservations. This is particularly true when users overestimate application run time.

2. **A co-allocation model that supports two rescheduling operations for message passing applications.** Most of existing work on resource co-allocation assumes that once the requests are placed in the scheduling queues, their schedule is not updated. We therefore:

   - Introduce a co-allocation model for message passing applications with rescheduling support based on two operations: start time shifting and process remapping;
   - Show the benefits of rescheduling co-allocation requests using user response time and system utilisation as main metrics;
   - Present technical challenges to deploy the co-allocation policies and how to address them.

   The main findings of this model are 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. We have also observed in some scenarios that process remapping increases the chance of placing the tasks of multi-site jobs into a single cluster, thus eliminating any inter-cluster network overhead. Moreover, a simple and practical approach can be used to generate run time predictions depending on the application. Predictions are important since applications may be aborted when rescheduled to slower resources; unless users provide high run time overestimations. When applications are rescheduled to faster resources, backfilling may not be explored if estimated run times are not reduced.

3. **A resource co-allocation model based on execution offers for BoT applications.** Execution offers are a mechanism in which resource providers advertise their interest in executing an entire BoT or only part of it without revealing their load and total computing power. Both generation and composition of offers have an impact on co-allocating resources for BoT applications. We therefore:

   - Propose two offer generators for deadline-constrained BoT applications;
   - Propose three offer composition policies defined according to the amount of information resource providers disclose to metaschedulers;
   - Compare the offer composition policies against a well-known policy based on load information, i.e. based on free time slots.

   The main findings are that offer-based scheduling delays less jobs that cannot meet deadlines in comparison to scheduling based on load availability (i.e. free time slots); thus it is possible to keep providers' load information private when scheduling multi-site BoTs; and if providers publish their total computing power configuration, more local jobs can meet deadlines.

4. **A coordinated rescheduling algorithm and evaluation of run time estimates for bag-of-tasks running across multiple providers.** Existing research on scheduling of bag-of-tasks across multiple providers considers that each provider reschedules tasks independently. We therefore:

   - Propose a coordinated rescheduling algorithm for bag-of-tasks running across multiple providers;
   - Show the importance of accurate run time estimates when co-allocating resources for bag-of-tasks applications on multiple providers;
   - Propose the use of on-line system generated predictions for bag-of-tasks.

The main findings are that tasks of the same BoT can be spread over time due to inaccurate run time estimates and environment heterogeneity. Coordinated rescheduling of these tasks can reduce user response time. Moreover, accurate run time estimates assist metaschedulers to better distribute the tasks of BoT applications on multiple sites. Although system generated predictions may consume time, the schedules produced by more accurate run time estimates pay off the profiling time since users have better response times than simply overestimating resource usages.

1.4 Methodology

Most of the results presented in this thesis are based on simulations using workloads from real production systems. In order to analyse technical challenges to deploy the co-allocation policies, we performed experiments using Grid'5000, which consists of a set of clusters in France dedicated to large-scale experiments[2]. Here we present an overview of the workloads used in our experiments and the scheduling system in which we developed the co-allocation policies.

1.4.1 Workload Files from Parallel Machines

We used workload logs from real production systems available at the Parallel Workloads Archive[3]. These logs are ASCII text files that follow the Standard Workload Format[4]. The top of a log file contains comments on the machine where the log was obtained, such as name, number processors and their configuration, and scheduling queue attributes. The body of the log file contains a sequence of lines, each representing a job. There are eighteen fields containing job attributes, being four of them relevant for this thesis: arrival time, estimated run time, actual run time, number of required processors.

We adapted the workload files to meet the requirements of our experiments. In order to evaluate the co-allocation policies under different loads, we modified job arrival times, by either reducing or increasing them, such that to achieve the required load. This strategy was also used by other researchers [104]. We also incorporated parameters, such as network overhead, deadlines, and time to obtain system-generated run time predictions. We discuss their inclusion when describing the experiment each of these parameters were required.

[2]Grid'5000 website: https://www.grid5000.fr
[3]Parallel Workloads Archive: http://www.cs.huji.ac.il/labs/parallel/workload
[4]Standard Workload Format: http://www.cs.huji.ac.il/labs/parallel/workload/swf.html

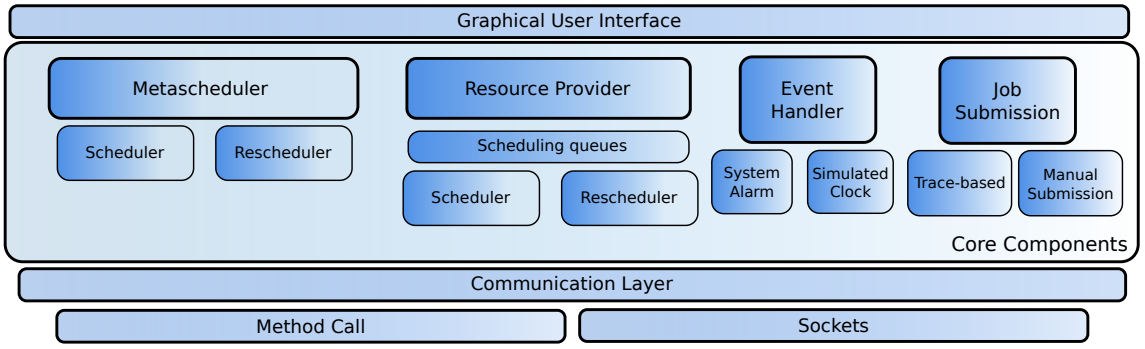


Figure 1.4: Parallel Job Fit (PaJFit) architecture and its main components.

1.4.2 Scheduling System

We developed a system called Parallel Job Fit (PaJFit) to perform experiments in both simulated and actual execution modes. The discrete-event simulator provides the means to perform experiments in various conditions and long runs that could not be possible using a real environment. The communication layer based on sockets allowed us to perform experiments in a real testbed. We used Java to implement the system, which currently consists of 70 classes and approximately 20 thousand lines of source code.

PaJFit architecture is composed of a metascheduler, a resource provider, event handler, and a job submission handler. For both metascheduler and resource provider components, we implemented plug-ins to schedule message passing and bag-of-tasks applications. Figure 1.4 illustrates the main components, which we detail throughout the thesis. The event handler and communication layer components are responsible for differentiating the execution between simulation and actual modes. Event handler in the simulated mode contains a simulated clock and a list of events that are executed at each simulated time unit. In the actual execution mode, each component, such as metascheduler and resource providers, contains its own clock and a list of events to process. The communication layer is a Java Interface that has two implementations, one based on method calls, in which the simulator contacts any component by calling Java methods, and another based on sockets, in which components running in different machines communicate through sockets.

The implementation of the co-allocation algorithms is distributed between the scheduler and rescheduler of both resource provider and metascheduler components, as illustrated in Figure 1.5. Note that as we used the metascheduler as a mediator between providers during rescheduling phase, the metascheduler also requires methods for rescheduling. An alternative implementation could be to distribute the rescheduling responsibilities among the resource providers. This second approach would increase the middleware complexity, being only required for settings with large number of providers, which is not the case in this thesis.

PaJFit also contains a simple but effective graphical user interface to visualise job

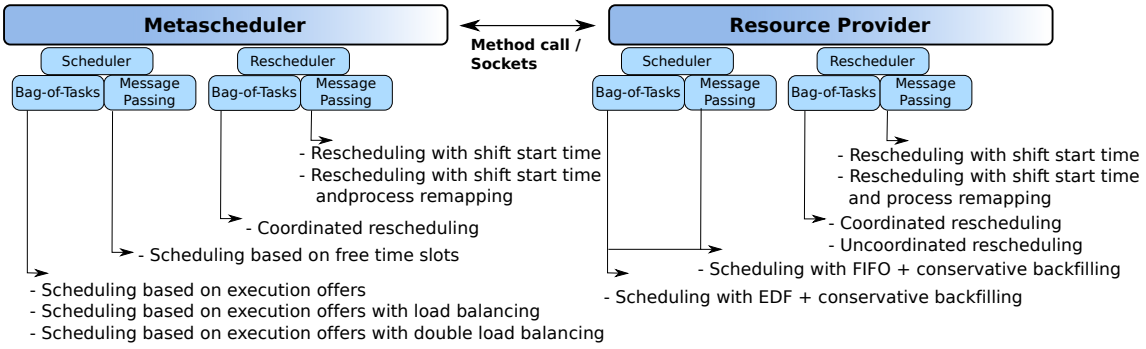


Figure 1.5: Algorithms implemented in the metascheduler and resource provider classes.

schedules (Figure 1.6), which proved to be an important debugging tool during the development of the co-allocation policies. Each resource provider window has a display with its job schedule, which can be zoomed in or out, and a list of jobs scheduled and their attributes such as arrival time, run time estimate, and number of required processors. For the simulated mode, it is possible execute events one by one ("STEP" button), by amount of simulated time ("STEP SIZE"), until the last event ("RUN ALL") or until a specified simulated time ("RUN UNTIL").

Figure 1.6: PaJFit graphical user interface.

1.5 Thesis Roadmap

Apart from this chapter and the Conclusion chapter, the thesis consists of other six chapters. Figure 1.7 represents the relation among the chapters. There are three chapters on the area of advance reservation based co-allocation for message passing applications and two chapters on co-allocation for bag-of-tasks applications.

The core chapters of this thesis derive from research papers published/submitted dur-

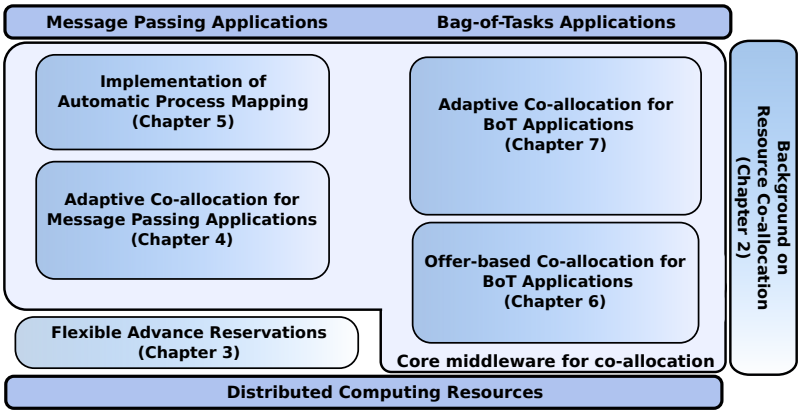


Figure 1.7: Organisation of the thesis chapters.

ing the course of the PhD candidature. The thesis chapters and their respective papers are the following:

- Chapter 2 presents background information, challenges, and existing solutions for resource co-allocation. It also positions the thesis in relation to existing work:
  - **Marco A. S. Netto** and Rajkumar Buyya. Resource Co-allocation in Grid Computing Environments. Chapter in *Handbook of Research on P2P and Grid Systems for Service-Oriented Computing: Models, Methodologies and Applications*. Edited by Nick Antonopoulos, Georgios Exarchakos, Maozhen Li and Antonio Liotta. IGI Global publisher, 2009 (ISBN 1-61520-686-8).
- Chapter 3 describes a study on flexible advance reservations, which is the foundation for the adaptive co-allocation of resources for message passing applications:
  - **Marco A. S. Netto**, Kris Bubendorfer and Rajkumar Buyya. SLA-based advance reservations with flexible and adaptive time QoS parameters. *Proceedings of the International Conference on Service Oriented Computing (ICSOC'07)*, pages 119–131, Vienna, Austria - September 17-20, 2007. Springer Verlag Lecture Notes in Computer Science.
  - **Marco A. S. Netto**, Rajkumar Buyya. Impact of Adaptive Resource Allocation Requests in Utility Cluster Computing Environments. *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid (IEEE CCGrid'07)*, pages 214–221, Rio de Janeiro, Brazil, May 2007.
- Chapter 4 presents the resource co-allocation model with rescheduling support for message passing applications. The model consists of two operations that reduce user response time and increase system utilisation:

  - **Marco A. S. Netto** and Rajkumar Buyya. Rescheduling Co-Allocation Requests based on Flexible Advance Reservations and Processor Remapping. *Proceedings of the 9th IEEE/ACM International Conference on Grid Computing (IEEE/ACM GRID'08)*, Tsukuba, Japan, Sept. 29-Oct. 1, 2008.

- Chapter 5 proposes the use of performance predictions for co-allocating iterative parallel applications with rescheduling support. The chapter presents a detailed case study using Grid'5000 and a multi-objective optimisation application with synchronous and asynchronous communication models. This chapter describes how rescheduling can be deployed in practice from the application's perspective:

  - **Marco A. S. Netto**, Christian Vecchiola, Michael Kirley, Carlos A. Varela, and Rajkumar Buyya, Resource Co-Allocation based on Application Profiling: A Case Study in Multi-Objective Evolutionary Computations, Technical Report, CLOUDS-TR-2009-5, Cloud Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, Aug. 3, 2009.

- Chapter 6 introduces the co-allocation problem for bag-of-tasks applications and evaluates the impact of information resource providers disclose to metaschedulers. This chapter also describes the concept of execution offers and offer composition:

  - **Marco A. S. Netto** and Rajkumar Buyya. Offer-based Scheduling of Deadline-Constrained Bag-of-Tasks Applications for Utility Computing Systems. *Proceedings of the 18th International Heterogeneity in Computing Workshop (HCW09), in conjunction with the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS'09)*, Roma, Italy, May 2009.

- Chapter 7 presents the coordinated rescheduling for BoT applications and the impact of run time estimates when executing these applications across multiple providers. Results are based on both simulations and real executions using Grid'5000. This chapter also describes an example of application-profiling using a ray-tracing tool:

  - **Marco A. S. Netto** and Rajkumar Buyya, Coordinated Rescheduling of Bag-of-Tasks for Executions on Multiple Resource Providers, Technical Report, CLOUDS-TR-2010-1, Cloud Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, Jan 22, 2010.

Chapter 8 concludes the thesis with a discussion of our main findings and future research directions in the area of resource co-allocation and co-related areas.

Chapter 2

Background, Challenges, and Existing Solutions

One of the promises of distributed systems is the execution of applications across multiple resources. Several applications require coordinated allocation of resources hosted on autonomous domains—problem known as resource co-allocation. This chapter describes and categorises existing solutions for the main challenges in resource co-allocation: distributed transactions, fault tolerance, network overhead, and schedule optimisation. The chapter also presents projects that developed systems with resource co-allocation support, and the thesis positioning in relation to existing research.

2.1 Introduction

When users require resources from multiple places, they submit requests called *metajobs*—also known as multi-site jobs, multi-cluster jobs, or co-allo-cation requests—to a *metascheduler*, which in turn contacts *local schedulers* to acquire resources (Figure 2.1). These metajobs are decomposed into a set of jobs or requests to be submitted to resource providers. In this thesis, a provider contains a cluster with a set of processors and therefore we use provider and cluster interchangeably. Site is the physical location of the provider. 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 metascheduler is a software that runs either in the user desktop machine or in a remote server. The local scheduler runs in the front-end node of each provider and is responsible for managing a scheduling queue to control resource access. The scheduling queue contains *jobs*, also known as *requests*, coming from local or remote sites. These jobs are specifications given by users containing the required number of processors and estimated usage time. The empty spaces in the scheduling queue are called *fragments*. When the metascheduler asks for resources, the local schedulers look for free *time slots*,

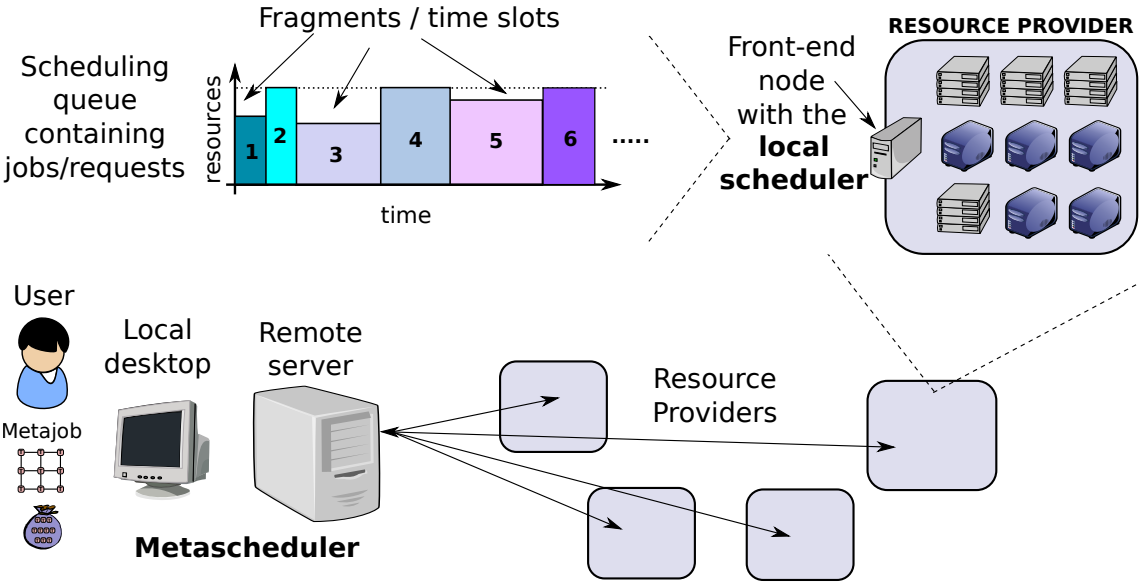


Figure 2.1: Computing architecture and main elements.

which are fragments in the scheduling queue and empty spaces after the last expected completion time.

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 jobs from the same metajob. An alternative is to associate to each job a list of the resource providers holding the other jobs from the same application. 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 centralised entity becomes a bottleneck when managing large number of providers. The second approach has opposite advantages and drawbacks.

When co-allocating resources for message passing applications, the metascheduler uses the free time slots to make advance reservations, whereas for BoT applications, the metascheduler uses the free time slots as an indicator on the number of tasks to be placed in each provider.

From the moment users request resources to the moment applications start execution, four challenges have to be addressed regarding resource co-allocation: distributed transactions, fault tolerance, inter-site network overhead, and schedule optimisation [90].

Distributed transactions is the first challenge discussed in this chapter. Resource co-allocation involves the interaction of multiple entities, namely clients and resource providers. More than one client may ask for resources at the same time from the same providers. This situation may generate deadlocks if the resource providers use a locking procedure; or livelock if there is a time out associated with the locks. Distributed trans-

actions is the research area focused on avoiding deadlocks and livelocks, and minimising the number of messages during these transactions.

Another common problem in the resource co-allocation field is that a failure in a single resource compromises the entire execution of an application that requires multiple resources at the same time. One approach to minimise this problem is defining a fault tolerance strategy that notifies applications of a problem with a resource. A software layer could then provide the application with a new resource, or discard the failed resource if it is not essential.

From the applications' perspective, one of the main problems when executing them over multiple clusters is the inter-cluster network overhead. Several parallel applications require inter-process communication, which may become a bottleneck due to the high latency of wide-area networks. Therefore, it is important to evaluate the benefits of multi-site execution and develop techniques for mapping application processes considering communication costs.

Scheduling multi-cluster applications is more complex than scheduling single-cluster applications due to the tasks time dependency. In addition, as some applications have more flexibility on how to map tasks to resources, the scheduler has to analyse more mapping options. For parallel applications with inter-process communication, the scheduler also has to take into account the network overhead. Moreover, the scheduling of a co-allocation request depends on the goals and policies of each resource provider.

When implementing and deploying a software system that supports resource co-allocation, developers initially face the first three mentioned problems. Once a system is in production, the schedule optimisation becomes one of the most important issues. Most of the work on co-allocation has focused on schedule optimisation, mainly evaluated by means of simulations.

In the next section, we describe in detail the solutions proposed for these four major problems in resource co-allocation, which mainly focus on message passing parallel applications. We also present relevant work on scheduling of BoT applications, which assist in positioning the thesis regarding this application model. We also give an overview of each project before detailing their solutions. Some projects, especially those with middleware implementation, have faced more than one challenge. For these projects, we have included a section with a comparison of their features and limitations.

2.2 Challenges and Solutions

We have classified the existing work on resource co-allocation according to the four major challenges. Table 2.1 contains a short description and solutions for each research topic.

Table 2.1: Summary of research challenges and solutions for resource co-allocation.

Research Topic	Description	Solutions
Distributed Transactions	Prevention of deadlocks and livelocks; Reduction of messages during transactions	Two- and Three-phase commit protocol; Order-based Deadlock Prevention Protocol; Polling
Fault Tolerance	Software and hardware failures; Coordinated allocation	Advance reservations; Backtracking; User's fault recovery strategy; Flexible resource selection
Network Overhead	Evaluation of inter-site communication; Response time reductions	Topology-aware placement; Use of network information; Proximity of data location to resources
Schedule Optimisation	Increase system utilisation; Reduce user response time	Advance reservations; Network-aware scheduling; Rescheduling and negotiation support

Some of the projects have focused on more than one aspect of resource co-allocation. However, the description of such projects is located in the section of the research topic with their most significant contribution.

2.2.1 Distributed Transactions

The research on the management of Distributed Transactions involves the development of protocols to avoid deadlocks and livelocks that may occur during the co-allocation process. In addition, the protocols aim to minimise the number of messages during these transactions. A deadlock may happen when multiple clients ask for resources at the same time from the same resource providers and these providers work with schedulers that lock themselves to serve requests. Similar to the two conditions of a deadlock, a livelock happens when the schedulers in the resource providers have a timeout associated with the locks. The distributed transactions research field has been quite active in database communities [17]. However, this section describes projects interested in this area focusing on resource co-allocation for large-scale systems such as Grid Computing. Table 2.2 summarises the methods and goals used by the researchers on this topic.

The two-phase commit protocol consists in sending *prepare* and *commit* messages to

Table 2.2: Summary of methods and goals for distributed transactions.

Method	Goals
Two-phase commit protocol	Prevent gathering partial number of resources
Polling	Prevent deadlocks and livelocks; Remove requirement of ordering resources; Support asymmetric communication
Order-based Deadline Prevention protocol	Prevent deadlocks and livelocks
Three-phase commit protocol	Prevent deadlocks and livelocks; Support messages to be lost and delayed

local schedulers. The prepare message holds the resources, whereas the commit message allocates the resources. There are variations of this protocol to enhance its functionalities as described below.

Kuo et al. [70] proposed a co-allocation protocol based on the two-phase commit protocol to support cancellations that may occur at any time. Their protocol supports nested configuration, i.e. a resource can be a co-allocator for other resource sets. However, it has no support for atomic transactions. Therefore, a transaction may reach a state where a reservation executes on some resources, while other reservations are cancelled. They deal with race conditions on the request phase and propose a non-blocking protocol with a time out mechanism.

Takefusa et al. [115] extended the two-phase commit protocol by including polling from the client to the server. The authors argued that although there is a communication overhead between the client and server due to the polling, this non-blocking approach allows asymmetric communication, and hence, the client does not need a global address. Moreover, it eliminates firewall problems, avoids hang-ups because of server or client side troubles, and enables the recovery of each process from a failure.

Deadlocks and livelocks are problems that may occur during a distributed transaction depending on the allocation protocol and computing environment. Park [94] introduced a decentralised protocol for co-allocating large-scale distributed resources, which is free from deadlocks and livelocks. The protocol is based on the Order-based Deadlock Prevention Protocol ODP^2, but with parallel requests in order to increase its efficiency. The protocol uses the IP address as the unique local identifier to order the resources. Another approach to avoid deadlock and livelock is the exponential back-off mechanism, which

does not require the ordering of resources. Jardine et al. [62] investigated such a mechanism for co-allocating resources.

Service Negotiation and Acquisition Protocol (SNAP) is a well-known protocol aimed at managing access to and use of distributed computing resources in a coordinated fashion by means of Service Level Agreements (SLAs) [36]. SNAP coordinates the resource management through three types of SLAs, which separate task requirements, resource capabilities, and biding of tasks to resources. From the moment users identify target resources to the moment when they submit tasks, other users may access the chosen resources. This happens because information obtained from the providers may be out-of-date during the selection and actual submission of tasks. In order to solve this problem, Haji et al. [55] developed a Three-Phase commit protocol for SNAP-based brokers. The key feature of their protocol is the use of *probes*, which are signals sent from the providers to the candidates interested in the same resources to be aware of resource status' changes.

Maclaren [78] also proposed a Three-Phase Commit Protocol, which is based on Paxos consensus algorithm [53]. In this algorithm, the coordinator responsible for receiving confirmation answers from resource providers is replaced with a set of replicated processes called *Acceptors*. A leader process coordinates the acceptor processes to agree on a value or condition. Any acceptor can act as the leader and replace the leader if it fails. This algorithm allows messages to be lost, delayed or even duplicated. Therefore, the Paxos Commit protocol is a valuable algorithm when considering the fault tolerance for distributed transactions in order to co-allocate resources in Grids.

Jobs can also allocate resources in a pull manner, which is the approach described by Azougagh et al. [9], who introduced the Availability Check Technique (ACT) to reduce the conflicts during the process of resource co-allocation. The conflicts are generated when multiple jobs are trying to allocate two or more resources in a crossing way simultaneously, resulting in deadlocks, starvations, and livelocks. Rather than allocating resources, jobs wait for updates from resource providers until they fulfil their requirements.

2.2.2 Fault Tolerance

Hardware and software failures are common in large-scale systems due to their complexity in terms of resource autonomy, heterogeneity, and number of components. Improper configuration, network error, and authorisation difficulties are examples of problems that affect the execution of an application. For a multi-site application, a failure in a single resource may compromise the entire execution [34]. This section describes some of the projects related to fault tolerance for multi-site applications. Table 2.3 summarises the main methods used for fault tolerance in resource co-allocation.

Table 2.3: Summary of methods and goals for fault tolerance.

Method	Goals
Flexible resource selection	Ignore optional resources; Specify alternative resources
Advance reservations	Ensure all resources are available at required time
Backtracking	Replace failed/unavailable resources
User's fault recovery strategy	Users specify their own recovery strategy

Czajkowski et al. [35] proposed a layered architecture to address *failures* for co-allocation requests. The architecture has two co-allocation methods: *Atomic Transaction* and *Interactive Transaction*. In the atomic transaction, all the required resources are specified at the request time. The request succeeds if all resources are allocated. Otherwise, the request fails and none of the resources is acquired. The user can modify the co-allocation content until the request initialises. In the interactive transaction method, the content of a co-allocation request can be modified via *add*, *delete*, and *substitute* operations. Resources can be classified in three categories: *required* (failure or time out of this type of resource causes the entire computation to be terminated—similar to atomic operation); *interactive* (failure or time out of a resource results in a call-back to the application, which can delete or substitute to another resource—i.e. the resource is not essential or it is easy to find replacements); *optional* (failure or time out is ignored). Similar approach was explored by Sinaga et al. for the DUROC system [108]. The authors extended DUROC to keep trying to schedule jobs until they could get all the required resources, or until the number of tries achieved a certain threshold.

The Globus Architecture for Reservation and Allocation (GARA) was one of the first co-allocation systems that considered Quality of Service (QoS) guarantees [51]. The main goal of GARA was to provide resource access guarantees by using advance reservations. GARA had the concept of backtracking, in which when a resource fails, it is possible to try other resources until the request succeeds or fails.

Röblitz and Reinefeld [97] presented a framework to manage reservations for applications running concurrently on multiple sites and applications with components that may be linked by temporal or spatial relationships, such as job flows. They defined and described co-reservations along with their life cycle, and presented an architecture for processing co-reservation requests with support for fault tolerance. When handling confirmed co-reservations, as part of the requested resources may not be available, alternative ones

should substitute them. If it is not possible, a best-effort option could be followed or the request should be cancelled. Users define such a behaviour through a fault recovery strategy in the request specification. The authors also discussed the concept of virtual resources to provide the user with a consistent view on multiple reservations. Therefore, it is possible to have modifications of the reservations in a transparent way for users.

2.2.3 Network Overhead for Inter-Cluster Communication

One of the main problems when executing message passing parallel applications over different clusters is the network overhead [43]. The wide-area networks may degrade the performance of these parallel applications, thus generating a considerable execution time delay. Therefore, it is important to evaluate the benefits of multi-site executions and investigate techniques for mapping application processes considering communication costs. Several researchers have investigated the benefits of multi-site executions using different methods and testbeds. Network overhead has also been investigated for co-allocation data and processors. Table 2.4 summarises the main methods for evaluating network overhead for multi-site parallel executions.

Table 2.4: Summary of methods and goals for network overhead evaluations.

Method	Goals
Application specific	Evaluate specific application properties
Data-intensive applications	Consider transfer of large amounts of data
Simulation-based evaluation	Evaluate wide range of parameters and scenarios
Real-testbed-based evaluation	Evaluate network in real conditions
Topoloy-aware placement	Consider network heterogeneity to map tasks

The Message Passing Interface (MPI) has been broadly used for developing parallel applications in single site environments. However, executing these applications on multi-site environments imposes different challenges due to network heterogeneity. Intra-site communication has much lower latency than inter-site communication. There are several MPI implementations, such as MPICH-VMI [93], MPICH Madeleine [7], and MPICH-G2 [65], that take into account the network heterogeneity and simplify the application development process.

Ernemann et al. [45] studied the benefits of sharing jobs among independent sites and executing parallel jobs in multiple sites. When co-allocating resources, the scheduler looks for a site that has enough resources to start the job. If it cannot find it, the scheduler sorts the sites in a descending order of free resources and allocates those resources in this order to minimise the number of combined sites. If it is not possible to map the job, the scheduler queues the job using Easy Backfilling [85]. The authors varied the network overhead from 0 to 40% and concluded that multi-site applications reduce average weighed response time when the communication overhead is limited to about 25%. This threshold has been used for most of the following work that considers network overhead for co-allocation.

Bucur et al. [20] investigated the feasibility of executing parallel applications across wide-area systems. Their evaluation has as input parameters the structure and size of jobs, scheduling policy, and communication speed ratio between intra- and inter-clusters. They investigated various scheduling policies and concluded that when the ratio between inter- and intra-cluster is 50, it is worth co-allocating resources instead of waiting for all resources to be available in a single cluster.

Jones et al. [64] proposed scheduling strategies that use available information of the network link utilisation and job communication topology to define job partition sizes and job placement. Rather than assuming a fixed amount of time for all inter-cluster communication or assigning execution time penalties for the network overhead, the authors considered that inter-cluster bandwidth changes over time due to the number and duration of multi-site executions in the environment. Therefore, they explored the scheduling of multiple co-allocation jobs sharing the same computing infrastructure. As for the co-allocation strategies, the authors investigated:

- First-Fit, which performs resource co-allocation by assigning tasks starting with the cluster having the largest number of free nodes and does not use any information of neither the job communication characterisation nor network link saturation;

- Link Saturation Level Threshold Only, which is similar to First-Fit but discards clusters with saturated links;

- Link Saturation Level Threshold with Constraint Satisfaction, which tries to put jobs into a large portion of a single cluster (e.g. 85% of resources); and Integer Constraint Satisfaction, which uses jobs' communication characterisation and current link utilisation to prevent link saturations.

Jones et al. [64] concluded that it is possible to reduce multi-site jobs' response time by using information of network usage and jobs' network requirements. In addition, they

concluded that this performance gain depends heavily on the characteristics of the arriving workload stream.

Mohamed and Epema [83] addressed the problem of co-allocating processors and data. They presented two features of their metascheduler, namely different priority levels of jobs and incrementally claiming processors. The metascheduler may not be able to find enough resources when jobs are claiming for resources. In this case, if a job j claiming for resources has high priority, the metascheduler verifies whether the number of processors used by low priority jobs is enough to serve the job j. If it is enough, the metascheduler preempts the low priority jobs in a descending order until enough resources are released. The metascheduler moves the preempted jobs into the low priority placement queue. The metascheduler uses the Close-to-Files (CF) job-placement algorithm to select target sites for job components [82]. The CF algorithm attempts to place the jobs in the sites where the estimated delay of transferring the input file to the execution sites is minimal.

2.2.4 Schedule Optimisation

Most of the existing work in resource co-allocation focuses on schedule optimisation of multi-site applications. Scheduling co-allocation requests is more complex than scheduling single site requests due to the tasks' time dependency. Moreover, some parallel applications are flexible on how they can be decomposed to run in multiple sites. For parallel applications with process communication, the scheduler has to take into account the network overhead. Table 2.5 summarises the main methods and environments for optimising the schedule of co-allocation requests.

Snell et al. [112] investigated the importance of using advance reservations for scheduling Grid jobs in multi-site environments in contrast to periodically blocking resources dedicated to Grid usage. They defined three scheduling strategies for co-allocation requests: *Specified co-allocation*, where users specify the resources and their location; *General co-allocation*, in which users do not specify the resource location; and *Optimal scheduling*, in which the metascheduler tries to determine the best location for every required resource in order to optimise cost, performance, response time or any other metric specified by the user. In order to co-allocate resources, the metascheduler queries the local schedulers for time slots. The query specifies the required time and resources, and the response is whether it is possible to allocate or not. They performed experiments to evaluate the impact of using advance reservations for metajobs against reserving periods for external usage. They concluded that the former approach is a viable solution for co-allocating resources for Grid jobs.

Alhusaini et al. [3, 4] proposed a two-phase approach for scheduling tasks requiring resource co-allocation. The first phase is an off-line planning where the scheduler assigns

Table 2.5: Summary of methods and scenarios for schedule optimisation.

Method	Goals
Advance reservation	Ensure all resources are available at the same time
Non-advance-reservation	Support for middleware without advance reservations; educe fragmentation in scheduling queues
Global queue	Simplify evaluation; focus on small scale environments
Autonomous queues	Consider local load and scheduling policies for multiple resource providers
Network-aware scheduling	Consider inter-site network overhead
On-line scheduling	Scheduling decisions based only on already accepted requests
Batch-mode scheduling	Make decisions knowing all requests a priori
Negotiation support	Achieve common goals of users and resource providers
Rescheduling support	Reduce fragmentation and user response time; increase system utilisation

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 where the scheduler maps tasks according to the actual computation and communication costs, which may differ from the estimated costs used in the first phase. In addition, applications may release a portion of the resources before they finish. The authors considered the scheduling of a set of applications rather than a single one (batch mode). Their optimisation criterion was to minimise the completion time of the last application, i.e. the makespan. They modeled the applications as Directed Acyclic Graphs (DAGs) and used graph theory to optimise the mapping of tasks.

Ernemann et al. [46] studied the effects of applying constraints for job decomposition when scheduling multi-site jobs. These constraints limit the number of processes for each site (*lower bound*) and number of sites per job. When selecting the number of processors used in each site, they sort the sites list by the decreasing number of free nodes in order to minimise the number of fragments for the jobs. The decision of using multi- or single-site to execute the application is automatic and depends on the load of the clusters. In their study, a *lower bound* of half of the total number of available resources appeared to be

beneficial in most cases. Their evaluation considers the network overhead for multi-site jobs. They summarised the overhead caused by communication and data migration as an increase of the job's run time.

Azzedin et al. [10] proposed a co-allocation mechanism that requires no advance 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 times are fixed. Consequently, advance reservations generate fragments that schedulers cannot utilise. Furthermore, the authors argued that a resource provider can reject a co-allocation request at any time in favour of internal requests, and hence the co-allocation would fail. Their scheme, called synchronous queuing (SQ), synchronises jobs at scheduling cycles, or more often, by speeding them up or slowing them down.

Li and Yahyapour [75] introduced a negotiation model that supports co-allocation. They extended a bilateral model, which consists of a negotiation protocol, utility functions or preference relationships for the negotiating parties, and a negotiation strategy. For the negotiation protocol, the authors adopted and modified the Rubinstein's sequential alternating offer protocol. In this latter protocol, players bargain at certain times. For each period, one of the players proposes an agreement and the other player either accepts or rejects. If the second player rejects, it presents an agreement, and the first player agrees or rejects. This negotiation continues until an agreement between the parties is established or the negotiation period expires. They evaluated the model with different input parameters for prices, negotiation behaviors, and optimisation weights.

Sonmez et al. [113] presented two job placement policies that take into account the wide-area communication overhead when co-allocating applications across multiple clusters. The first policy is *Cluster Minimisation* in which users specify how to decompose jobs and the scheduler maps the maximum job components in each cluster according to their processor availability (more processors available first). The second policy is *Flexible Cluster Minimisation* in which users specify only the number of required processors and the scheduler fills the maximum number of processors in each cluster. The main goal of these two policies is to minimise the number of clusters involved in a co-allocation request in order to reduce the wide-area communication overhead. The authors implemented these policies in their system called KOALA and evaluated several metrics, including average response time, wait time and execution time of user applications. Their policies do not use advance reservations, so at time intervals, the scheduler looks for idle nodes in the waiting queues of co-allocation requests.

Bucur et al. [21, 22] investigated scheduling policies on various queuing structures for resource co-allocation in multi-cluster systems. They evaluated the differences between having single global schedulers, only local schedulers and both schedulers together, as

well as different priorities for local and meta jobs. They used First Come First Serve in the scheduling queues. They have concluded that multi-site applications should not spend more than 25% of their time with wide-area communication and that there should be restrictions on how to decompose the multi-site jobs in order to produce better schedules.

Elmroth and Tordsson [44] modelled the co-allocation problem as a bipartite graph-matching problem. Tasks can be executed on specific resources and have different requirements. Their model relies on advance reservations with flexible time intervals. They explored a relaxed notion of simultaneous start time, where jobs can start with a short period of difference. When a resource provider cannot grant an advance reservation, it suggests a new feasible reservation, identical to the rejected one, but with a later start time. They presented an algorithm to schedule all the jobs within the start window interval, which tries to minimise the jobs' start time.

Decker and Schneider [40] investigated resource co-allocation as part of workflow tasks that must be executed at the same time. They extended the HEFT (Heterogeneous Earliest-Finish-Time) algorithm to find a mapping of tasks to resources in order to minimise the schedule length (makespan), to support advance reservations and co-allocation, and to consider data channel requirements between two activities. They observed that most of the workflows were rejected because no co-allocation could be found that covered all activities of a synchronous dependency or because there was not enough bandwidth available for the data channels. Therefore, they incorporated a backtracking method, which uses not only the earliest feasible allocation slot for each activity that is part of a co-allocation requirement, but all possible allocation ranges as well.

Siddiqui et al. [105] have introduced a mechanism for capacity planning to optimise user QoS requirements in a Grid environment. Their mechanism supports negotiation and is based on advance reservations. A co-allocation request contains sub-requests that are submitted to the resource providers, which in turn send counter-offers when users and resource providers cannot establish an agreement.

2.3 Systems with Resource Co-Allocation Support

The previous section described some of the existing solutions for each challenge on resource co-allocation. Although several projects have focused on one aspect of resource co-allocation, some research groups have faced more than one challenge, in particular groups that developed middleware systems with resource co-allocation support. This section presents a brief description of the main systems that support resource co-allocation and compares them according to their features and limitations based on the four co-allocation challenges.

GARA (The Globus Architecture for Reservation and Allocation) enables applications to co-allocate resources, which include networks, computers, and storage. GARA uses advance reservations to support co-allocation with Quality-of-Service and uses backtracking to handle resource failure [51]. GARA was one of the first projects to consider QoS for co-allocation requests.

OAR is the batch scheduler that has been used in Grid'5000 [25, 26]. OAR uses a simple policy based on all-or-none approach to co-allocate resources using advance reservations. One of the main design goals of OAR is the use of high level tools to maintain low software complexity.

KOALA is a grid scheduler that has been deployed on the DAS-2 and the DAS-3 multicluster systems in the Netherlands [83, 84]). KOALA users can co-allocate both processors and files located in autonomous clusters. KOALA supports malleable jobs, which can receive messages to expand and reduce the number of processors at application run time, and has fault tolerance mechanisms based on flexible resource selection.

HARC (Highly-Available Resource Co-allocator) is a system for reserving multiple resources, which can be processors and network light-paths, in a coordinated fashion [78]. HARC has been deployed in several computing infrastructures, such as TeraGrid, LONI (Louisiana Optical Network Initiative), and UK National Grid Service. One of the main features of HARC is its Three-Phase Commit Protocol based on Paxos consensus algorithm [53], which increases fault tolerance during the allocation process.

GridARS (Grid Advance Reservation-based System) is a co-allocation framework based advance reservation, which utilises WSRF/GSI (Web Services Resource Framework/Grid Security Infrastructure). GridARS can co-allocate both computing and network resources. One of the main aspects of GridARS is its Two-Phase Commit (2PC) Protocol based on polling [115]. The framework was evaluated on top of Globus by co-allocating computing and network resources from 7 sites in Japan and 3 sites in US. For the resources in US, GridARS used a wrapper on top of the HARC [78].

JSS (Job Submission Service) is a tool for resource brokering designed for software component interoperability [44]. JSS has been used in NorduGrid and SweGrid, and relies on advance reservations for resource co-allocation. These advance reservations are flexible, i.e. users can provide a start time interval for the allocation. JSS also considers time prediction for file staging when ranking resources to schedule user applications. JSS does not access the resource provider scheduling queues to decide where to place the advance reservations. Thus, the co-allocation is based on a set of interactions between the metascheduler and resource providers until the co-allocation can be accomplished.

Table 2.6 summarises the main features of each system according to each co-allocation challenge. The systems have used relied on different methods to deal with distributed transactions and fault tolerance problems. Most of them have no support to schedule applications considering network overhead. Therefore, it is the user who has to deal with this problem from the application level. Regarding schedule optimisation, most of the systems rely on advance reservations.

Table 2.6: Summary of systems with resource co-allocation support.

System	Distributed Transactions	Fault Tolerance	Network Overhead	Schedule Optimisation
GARA	Two-phase commit protocol	Use of alternative/ optional resources; backtracking	User pre-selects set of resources	Advance reservations based scheduling
KOALA	None (processors claimed incrementally)	Flexible selection until application receives resources	Close-to-File policy	Scheduling based on incremental processor claiming; no use of advance reservations
HARC	Three-phase commit protocol based on Paxos consensus algorithm	Only on the allocation transaction phase	User pre-selects set of resources	Advance reservations based scheduling using timetables offered by providers; Reservations can be modified
OAR	One-phase: All-or-nothing approach	Unavailable for co-allocation requests	User pre-selects set of resources	Advance reservation based scheduling using first fit
GridARS	Two-phase commit protocol with polling	On the allocation transaction phase with roll-back	User pre-selects set of resources	Advance reservation based scheduling
JSS	Negotiation with multiple interactions without blocking resources	Only on the moment of finding feasible schedule	Network-aware scheduling based on ranking	Advance reservation based scheduling with interactions between provider and metascheduler

2.4 Scheduling of BoT Applications

Bag-of-tasks applications are commonly referred as applications composed of independent tasks, and hence they do not require co-allocation. However, when scheduling these applications with completion time guarantees, each task of a bag has to be mapped to a resource such that to consider the overall application completion time. Therefore, resource co-allocation is important for these applications as well. This section provides an overview of existing work in the area of BoT scheduling.

Iosup et al. [57] have performed an extensive analysis of BoT scheduling in large-scale distributed systems. To evaluate the scheduling algorithms, they have considered different resource management architectures, task selection policies, and task scheduling policies. Among all the policies they have presented, three support a certain level of quality of service. These policies mainly work with priorities, without considering the expected task completion time. Casanova et al. [28] have also performed an extensive evaluation of policies for BoT scheduling, but they have assumed more detailed information on the characteristics and load of all resources in the system is available.

Beaumont et al. [13] have investigated scheduling policies for BoT applications considering both CPU and network. Their goal is to maximise the throughput of user applications in a fair way. Viswanathan et al. [125] have proposed scheduling strategies for large compute intensive loads that are arbitrarily divisible and have deadline constraints. They use a pull-based scheduling strategy with an admission control to ensure the application deadlines are satisfied. Their work assumes a coordinator node that knows all the tasks in the system. Users do not receive any feedback when they submit their tasks, since the resource providers ask for tasks from the coordinator node according to their available capacity. In addition, users do not receive a new estimation of completion time when deadlines cannot be satisfied.

Benoit et al. [14] have presented scheduling techniques for multiple BoTs. One of the strategies presented in their work uses the Earliest Deadline First policy for the resources to choose the next task to be executed among those they have received. The deadlines have been used as a priority to select tasks, without giving any feedback and completion time estimations to the users.

Kim et al. [68] have proposed and compared eight heuristics that consider priorities and deadlines for independent tasks. Different from our work, tasks submitted to the system are not part of BoTs, therefore their deadlines are independent. Moreover, users submit the tasks directly to a centralised entity, i.e. resource providers have no local load. Abramson et al. [1, 2] have proposed a broker to schedule *parameter sweep applications* (a subclass of BoT applications) with deadline constraints. Yeo and Buyya [131] have also

investigated scheduling of tasks with deadline constraints, but focusing on single cluster environments and with no feedback when deadlines are not possible to be satisfied.

2.5 Thesis Scope and Positioning

This thesis investigates four main aspects that have not been explored extensively in resource co-allocation: rescheduling, completion time guarantees, inaccurate run time estimates, and limited information access from providers to the metascheduler.

Rescheduling message passing applications. As in many management systems, initial schedules have to be updated to attend specification requirements [124]. Rescheduling applications waiting for resources is necessary due to inaccurate usage estimations, request cancellations and modifications, and resource failures. The main benefits of rescheduling are the reduction of application response time and increase in system utilisation. The closest work from our rescheduling policies comes from Alhusaini et al. [4] (Section 2.2.4) who proposed a two-phase co-allocation approach. After mapping the tasks to resources (first phase), 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 part of resources before the completion of the execution. Similar to our work, Alhusaini et al. have considered the inaccurate estimation of job requirements and the need of a rescheduling phase to overcome this problem. However, they have assumed 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.

Rescheduling BoT applications. Existing work on BoT application mostly focuses on the initial scheduling. However, inaccurate run time estimates have impact on the initial schedules, which require task rescheduling. Therefore, one of the thesis contributions is a coordinated rescheduling algorithm for BoT applications and an analysis of the impact of run time estimates when scheduling these applications across multiple providers.

Limited information access to the metascheduler. Resource providers can use execution offers when they are not willing to disclose private information such as local load, resource capabilities, and scheduling strategies. Bag-of-tasks are one of the main application models that can execute across multiple providers. Co-allocation for BoT applications is important since the results produced by all tasks constitute the solution of a single problem. The closest work from our execution offer-based scheduling comes from: Elmroth and Tordsson [44], who proposed a co-allocation algorithm that relies on the interaction between resource providers and metascheduler; and from Singh et al. [109] who

developed a multi-objective Genetic Algorithm mechanism for provisioning resources, in which resources publish their available time slots, or offers, so that applications can use them to meet to their requirements.

Considering the four resource co-allocation challenges described in this chapter, we mainly focus on schedule optimisation. For the rescheduling on message-passing applications, we also consider network overhead of inter-cluster communication, whereas for execution offers we consider issues on distributed transactions. Rescheduling can also be used as a mechanism to improve fault tolerance when applications are waiting for resources.

2.6 Conclusions

This chapter described the main research efforts in the area of resource co-allocation. These efforts involve four research directions: distributed transactions, fault tolerance, evaluation of inter-site network overhead, and schedule optimisation. We have presented existing work for each of these research directions. We have also described and compared six systems that support resource co-allocation.

When implementing real systems to deploy in production environments, the support for managing distributed transactions properly becomes an important issue. In terms of fault tolerance, co-allocation systems have been supporting the notion of optional and alternative resources that allows the scheduler to remap the application to other resources in case of failures. As for wide-area network overhead, most of existing work that considers it has used 25% of the execution time as the threshold to perform experiments. In addition, researchers have been considering location of data and computing resources to schedule multi-site applications. This is particularly necessary when scheduling data-intensive applications.

Most of the work on resource co-allocation focuses on schedule optimisation, which has been mainly based on the use of advance reservations. When scheduling multi-site applications, there are several factors to take into account. Apart from the network overhead and fault tolerance aspects, scheduling relies on the amount of information available for finding a placement for jobs. The use of a global queue or autonomous queues has a considerable influence on the scheduling strategies. Fortunately, several researchers have been considering the use of autonomous queues to perform their experiments, which is a fundamental characteristic of a large-scale computing environments.

Although there are several researchers working on scheduling policies for co-allocation requests, we have observed that most groups that developed middleware systems with

co-allocation support use simple scheduling techniques. That is because there are still several technical difficulties before more advanced scheduling policies can be deployed. Some of these technical problems are interoperability between metaschedulers and resource providers middleware, inter-site network overhead, and the autonomous policies of each resource provider.

This chapter also presented the thesis position in relation to existing work. The main research direction of this thesis is the rescheduling of co-allocation requests for message passing and bag-of-tasks applications. The next chapters describe in detail our contributions for this research direction, starting by understanding the impact of rescheduling advance reservations in a single provider.

Chapter 3

Flexible Advance Reservations

Advance reservations are important building blocks for coordinated allocation of resources hosted by multiple providers. This chapter presents a detailed study on reservations that have flexible time intervals, having system utilisation as the main performance metric. This study involves the evaluation of four scheduling algorithms along with measurements considering the time reservations are flexible and alternative offers from resource providers when reservations cannot be granted. The results show the importance of rescheduling advance reservations in single-site environments, which motivates a further study for multi-site applications in the following chapters.

3.1 Introduction

Advance reservations are an allocation mechanism to ensure resource availability in future. Resources can be display devices for a meeting, network channels required for data transmission, and computers from multiple clusters to execute a parallel application.

When a provider accepts an advance reservation, the user expects to be able to access the agreed resources at the specified time. However, changes may arise in the scheduling queue between the time the user submits the reservation to the time the user receives the resources. There are a number of reasons for such changes including: users cancelling and modifying requests, resource failures, and errors in estimating usage time. When reservations are not rescheduled, scheduling queues increase their fragmentation. This fragmentation reduces the potential scheduling opportunities and results in lower utilisation. Indeed, fragmentation also limits the positions in which other jobs can be scheduled. In order to minimise fragmentation due to advance reservations, researchers in this area have introduced and investigated the impact of flexible time intervals for advance reservations [30, 47, 67, 86, 98].

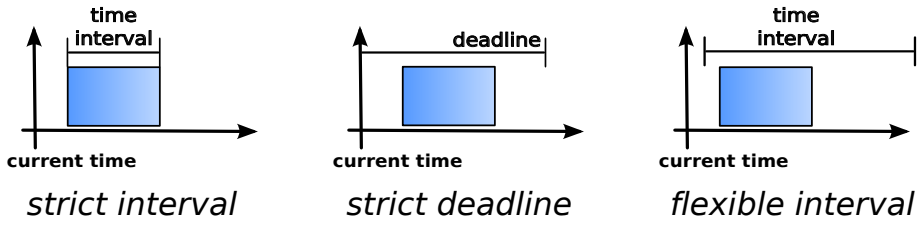


Figure 3.1: Time restrictions for allocating resources.

We extended the existing solutions on advance reservations and contributed to the research field by:

- Introducing the concept of adaptive time QoS parameters, in which the flexibility of these parameters are not static but adaptive according to the user needs and resource provider policies (Section 3.2);
- Presenting heuristics for scheduling advance reservations (Section 3.3);
- Performing experiments through extensive simulations to evaluate the advance reservations with flexible and adaptive time QoS parameters (Section 3.4). We show the results on the impact of system utilisation using different scheduling heuristics, workloads, time intervals, inaccurate estimation of execution times, and other input parameters. Moreover, we investigate cases when users accept an alternative offer from the resource provider on failure to schedule the initial request.

3.2 Reservation Specification

This section defines the set of parameters required for an advance reservation request. Following are the three time restrictions for allocating resources (Figure 3.1):

1. **Strict interval:** Users require resources at the same amount of time as the interval length and hence 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. **Strict deadline:** Users require that the execution completes prior to a deadline. This scenario typically applies when there are subsequent dependencies on the results of a given computation.
3. **Flexible interval:** There is a strict start and 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.

3.2.1 Scheduling Issues and Incentives

The 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 rigid advance reservations fragments the availability of the resource, which may result in wasted computation time, increased rejections, and reduced system utilisation.

The idea of having flexible intervals for advance reservations is to make it possible to modify or reallocate existing advance reservations when new jobs are submitted to the scheduler. To illustrate the advantage of flexible intervals, consider a workflow composed of three jobs that need to execute in sequence (Figure 3.2). Users have to estimate the execution time for each job and data transfer time between workflow stages. In order to prevent the execution being lost in case of an overrun, users overestimate resource usage time. However, as the actual execution may take less time, jobs can be rescheduled to start before the expected, thus reducing the workflow response time considerably.

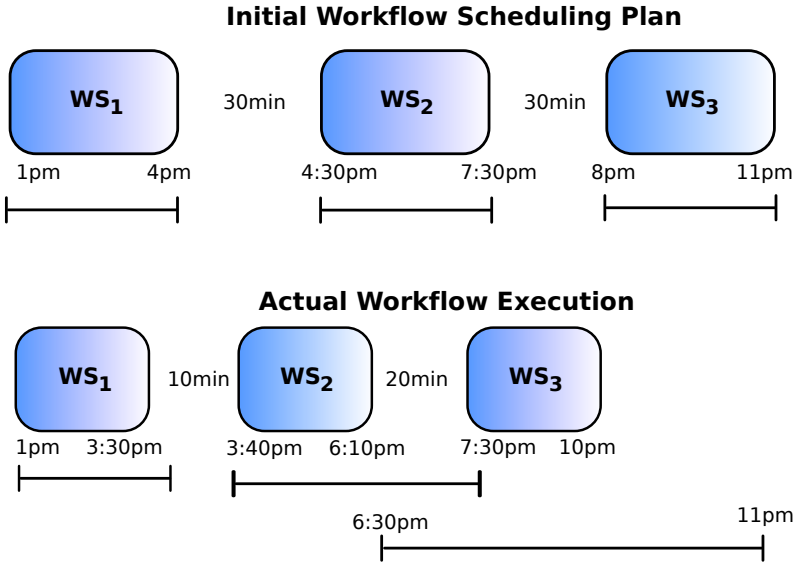


Figure 3.2: Reschedule of workflow tasks due to inaccurate run time estimations.

3.2.2 Reservation Parameters

Advance reservations contain a set of timing constraints and computational resources. Following is the notation and parameter definitions for a job j, which can be either rigid or moldable [48] (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 that specifies the relation between the number of resources and execution time T_j^e;
- T_j^s: job start time—time determined by the scheduler;
- T_j^r: job ready time—minimum start time determined by the user;
- T_j^c: job completion time—defined as $T_j^s + T_j^e$;
- D_j: job deadline—defined by the user.

3.3 Scheduling and Rescheduling of Reservations

The scheduling of a job consists in finding a free time slot that meets the job requirements. Rather than providing the user with the resource provider's scheduling queue information, we consider 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 users and other resource providers may exploit this commercially sensitive information.

Scheduling takes place in two stages. First, all jobs that are currently awaiting for execution on the machine are sorted based on some criterion. Then this list is scheduled in order, and if the new job can be scheduled, the reservation is accepted. If the job cannot be scheduled, then the scheduler can return a set of schedulable alternative times.

3.3.1 Sorting

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 $Q^w = \{j_1, ..., j_n\} \mid n \in \mathbb{N}$ [88]. The first queue contains jobs already in execution that 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 sorting techniques: Shuffle, First In First Out (FIFO), Biggest Job First (BJF), Least Flexible First (LFF), and Earliest Deadline First (EDF). The only sorting criterion that needs explanation is LFF, which sorts the jobs according to their flexibility in terms of time intervals. This approach is based on the work from Wu et al. [127], but considers only the time intervals. We define the time flexibility of a job j as follows:

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

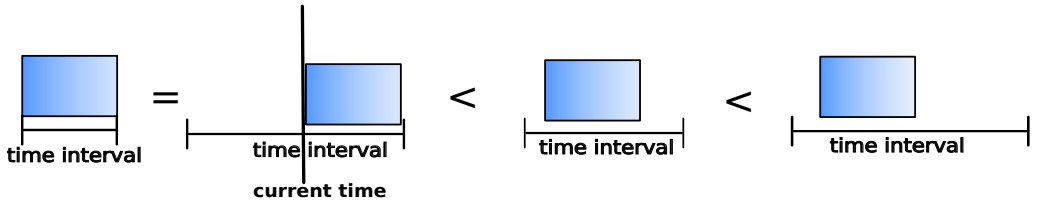


Figure 3.3: Sorting jobs using Least Flexible First criterion.

3.3.2 Scheduling

Algorithm 1 represents the pseudo-code for scheduling a new job j_k at the current time CT, returning true if it is possible to scheduled it, or false and a list of alternative possible schedules otherwise. The list of jobs in Q^w is sorted by a given criterion (e.g. EDF or LFF). Before scheduling a new job, the state of the system is consistent, which means that the current schedule of all jobs meets the users' QoS requirements. Therefore, during the scheduling, if a job j_i is rejected there are two options: (i) $j_i = j_k$, the new job cannot be scheduled; or (ii) $j_i \neq j_k$, the new job is scheduled but generates 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 $fixqueue$. Each job is scheduled by using the first fit approach with conservative backfilling—the first available time slot is assigned to the job [85]. 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 $[T_j^r, D_j - T_j^e]$.

When job j_k is rejected, all 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 (Figure 3.4). The list Ψ contains scheduling options defining start time, duration, and number of processors. 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 sorts it according to the percentage difference ϕ between the original T_j^s and D_j values and the alternative scheduler suggested options $OPTT_j^r$ and $OPTD_j$:

$$\phi_{opt} = \begin{cases} \frac{OPTD_j - D_j}{T_j^e}\text{ : option generated by placing } j_k \text{ after } j_i \\ \frac{OPTT_j^r - T_j^r}{T_j^e}\text{: option generated by placing } j_k \text{ before } j_i \end{cases}$$

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

```
1  Q^w ← Q^w ⋃ {j_k}
2  Sort Q^w according to a criterion (e.g. EDF or LFF)
3  k ← new index of j_k
4  jobscheduled ← true
5  for ∀j_i ∈ Q^w | i ≥ k and jobscheduled = true do
6      if schedulejob (j_i, Q^w, Q^r) = false then
7          jobscheduled ← false
8  if jobscheduled = false then
9      if i ≠ k then
10         fixqueue(Q^w, i, k) { update index of j_k (k ← i)}
11     return reschedule ∀j_i ∈ Q^w | i ≥ k
12 return true
```

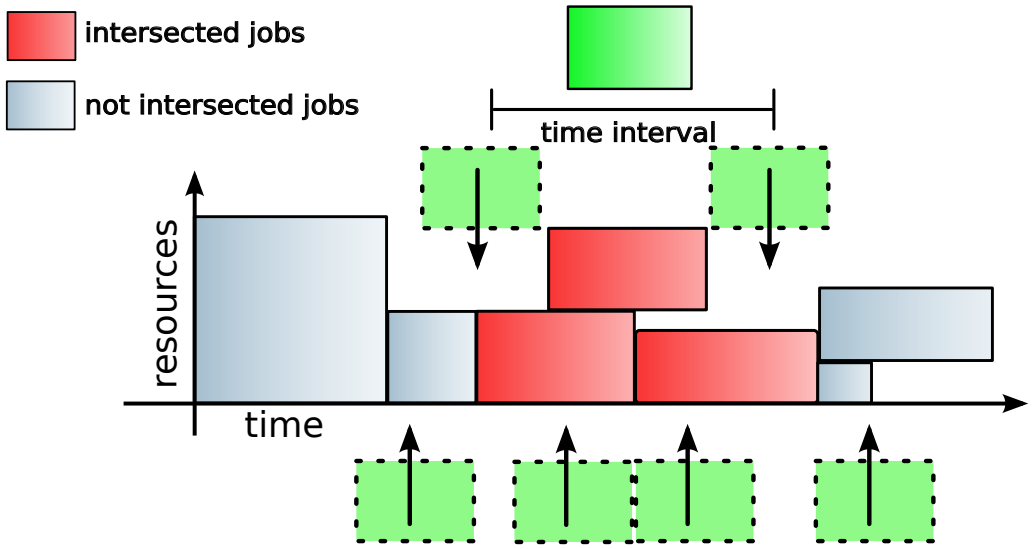


Figure 3.4: Example of alternative options generated when an advance reservation cannot be granted.

Once defined the possible positions of the new job j_k, all jobs in Q^w after j_k (including j_k) are rescheduled. If a job j_i is rejected, there are again two options: (i) $j_i = j_k$, the new job cannot be scheduled; or (ii) $j_i \neq j_k$, the new job is scheduled but delays another job $j_i \in Q^w$. In contrast to 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 is 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_k^r and D_k are set to the original values, and the rest of Q^w is again rescheduled. This process finishes when there are no more scheduling options to test. The first successful option is enough for a user who does not require an advance reservation.

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

```
1  OT_k^r ← T_k^r, OD_k ← D_k {keep original values}
2  for ∀OPT ∈ Ψ do
3      jobscheduled ← true
4      for ∀j_i ∈ Q^w | i ≥ k and jobscheduled = true do
5          if j_i = j_k then
6              Set T_k^r and D_k with option OPT
7          jobscheduled ← schedule(j_i)
8      if jobscheduled = false then
9          if i ≠ k then
10             fixqueue(Q^w, i, k)
11             T_k^r ← OT_k^r, D_k ← OD_k {restore original values}
12             return reschedule ∀j_i ∈ Q^w | i ≥ k
13         else
14             return false {already tested new options for j_k}
15     else
16         {valid option OPT in Ψ—inform user about this possibility}
17 if ∃ OPT ∈ Ψ | OPT generates a possible scheduling then
18     return true
19 return false
```

3.4 Evaluation

The goal of the evaluation is to observe the impact of rescheduling advance reservations on the system utilisation. We evaluated the use of flexible QoS parameters for advance reservations on the PaJFit simulator.

3.4.1 Experimental Configuration

We used the workload trace from the IBM SP2 system, composed of 128 homogeneous processors, located at the San Diego Supercomputer Center (SDSC)[5] 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. Therefore, for each job j, $D_j = T_j^{sub} + T_j^e * p$, where p is a random number defined by a Poisson distri-

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

bution with λ=5 (mean of delay factor), and T_j^{sub} is the request submission time defined in the workload traces. As we are working with advance reservations, we defined the release time of jobs as $T_j^r = D_j - T_j^e$. 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 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 still continue to specify strict deadlines.

3.4.2 Results and Analysis

For the first experiment, we evaluated the importance of sorting the jobs in the waiting queue according to a specific criterion. Figure 3.5 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 conservative backfilling. 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 system utilisation. It is worth noting that the results are not load sensitive, shown as the load increases—from normal (top graph) to high (bottom graph) in Figure 3.5. 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 $\pm$ 3.3 %, 50.9 $\pm$ 3.5 %, and 54.7 $\pm$ 3.7 %.

Using the EDF heuristic, we then evaluated the impact of the flexible time interval *duration* on system utilisation. We observe in Figure 3.6 that the longer the interval size, the higher the utilisation; longer interval sizes provide the scheduler with more options for fitting (juggling) advance reservations and thereby minimising the resource fragmentation.

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 by a Poisson distribution with λ=80, we assume the users in general overestimate the execution time [31, 73].

Compared to the results in Figure 3.6, we observe in Figure 3.7 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 advance reservations with rigid time QoS requirements.

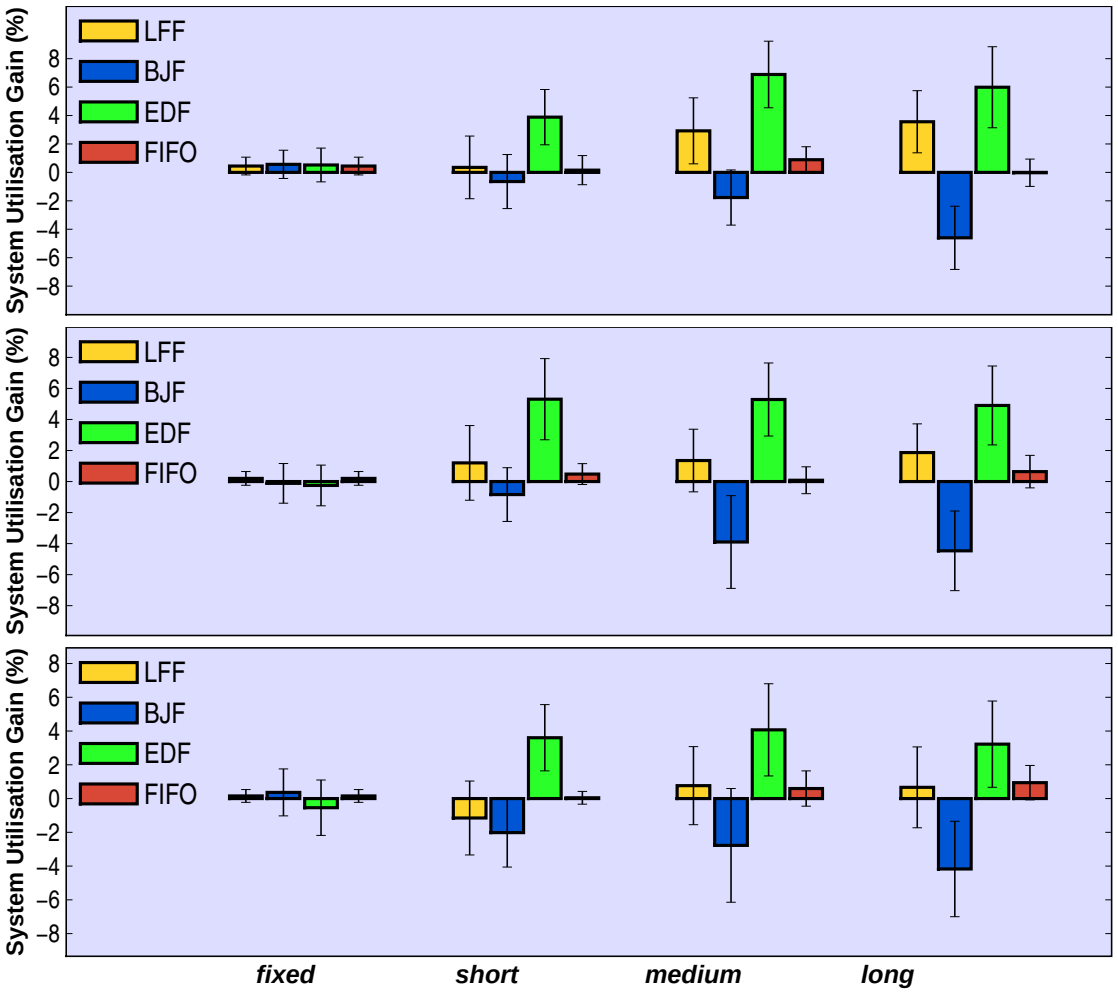


Figure 3.5: Impact of sorting criterion on system utilisation.

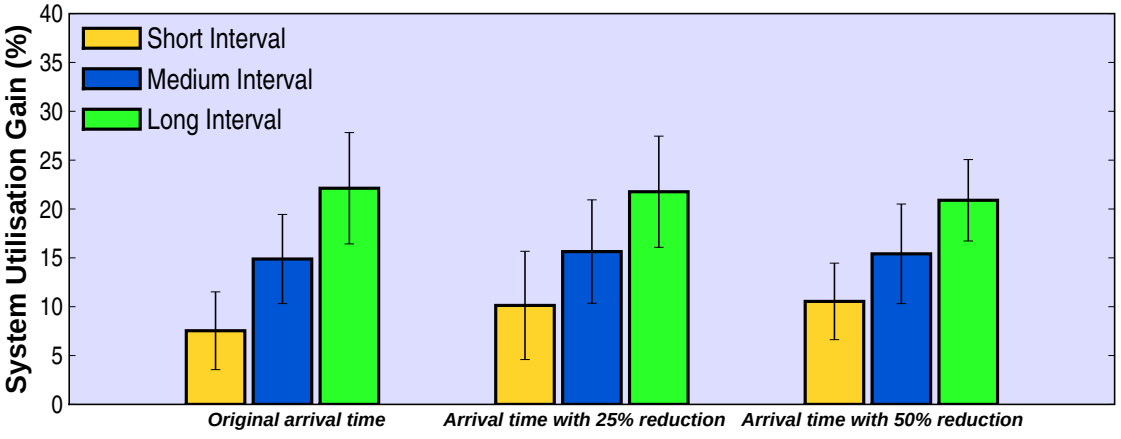


Figure 3.6: Impact of time interval size on resource utilisation.

Users may want to know with some assurance when their jobs will execute. They can ask the resource provider to fix their jobs when the time to receive the resources

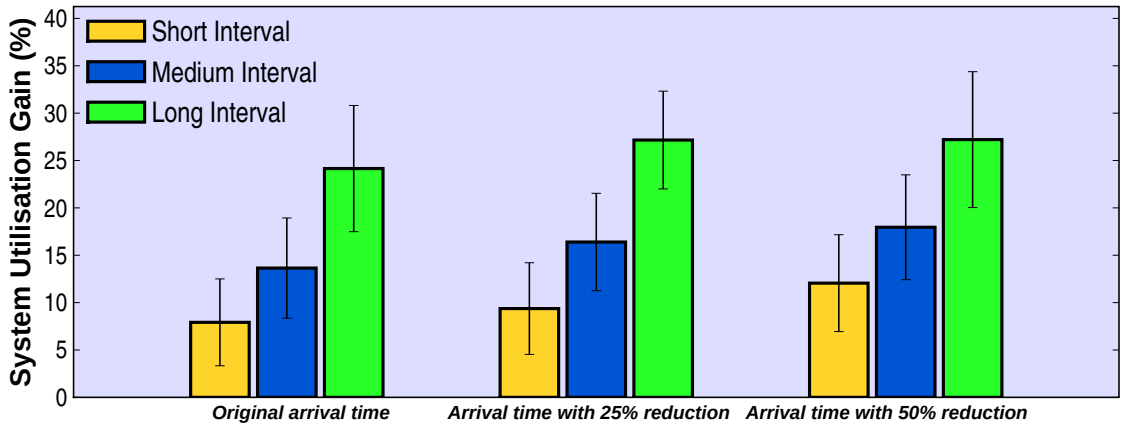


Figure 3.7: Impact of time interval size on resource utilisation with inaccurate run time estimations.

gets closer, i.e. remove the time interval flexibility by renegotiating the reservation. 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 after the job is accepted.

As in the first set of experiments (Figure 3.5), we performed runs for different workloads. However in this case the results for all workloads are similar, therefore we only present the graph for the medium workload in Figure 3.8. We observe that the longer users wait to fix their job requirements, the better the system utilisation since the scheduler has more opportunities to reschedule the workload.

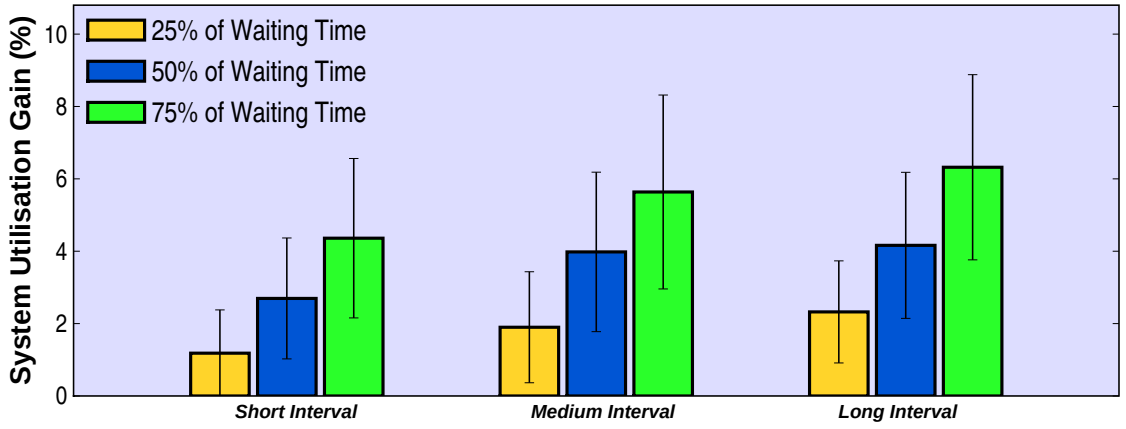


Figure 3.8: Effects of the duration the advance reservations are flexible.

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 user. When the resource provider cannot schedule a job j with the required starting time,

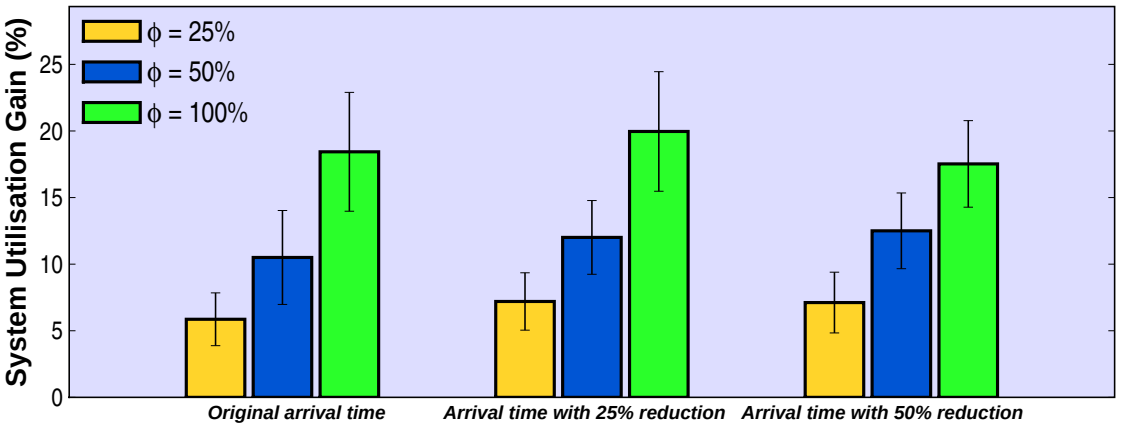


Figure 3.9: System utilisation using suggested option from resource provider.

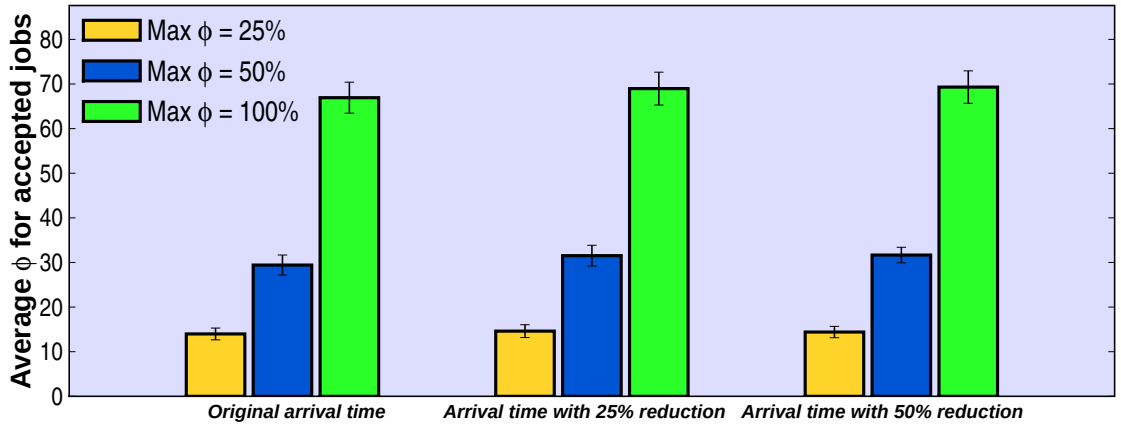


Figure 3.10: Average actual ϕ of jobs accepted through resource provider's suggestion.

it provides the user with other options (if possible) before and after the interval $[T_j^r, D_j]$. We selected the lowest difference ϕ of the options for each job j, given a threshold of 25%, 50% and 100%. Figure 3.9 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 user with an alternative option is a useful technique for users who cannot accept flexible intervals.

We also measured the difference between the actual and the thresholds ϕ for the jobs accepted through the option suggested by the resource provider. From Figure 3.10 we observe that in average, the value of actual ϕ is significantly less than the maximum ϕ defined by the resource provider. This means that even when users let providers choose options that are not close to the original request, the scheduler has flexibility to find alternative options better than the user threshold.

3.5 Conclusions

In this chapter we outlined user 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 cannot 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 user 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 to 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 application run time.

The results presented in this chapter are a solid foundation for scheduling applications that require co-allocation using advance reservations. Different from single site advance reservations, when rescheduling applications on multiple sites, resource providers have to keep all the advance reservations of a co-allocation request synchronised. In spite of its complexity, we will see in the following chapter that there are benefits of rescheduling applications that require multiple advance reservations.

Chapter 4

Adaptive Co-Allocation for Message Passing Applications

This chapter proposes adaptive co-allocation policies based on flexible advance reservations and process remapping. Metaschedulers 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, process remapping increases the chances of placing the tasks of multi-site jobs into a single cluster, thus eliminating the inter-cluster network overhead.

4.1 Introduction

Most of the current resource co-allocation solutions rely on advance reservations [40, 44, 51, 78, 97]. Although advance reservations are important to guarantee that resources are available at the expected time, they reduce system utilisation due to the inflexibility introduced in scheduling other jobs around the reserved slots [112]. To overcome this problem, several researchers have been working with flexible (or elastic) advance reservations, i.e. requests that have relaxed time intervals [47, 67, 86, 98, 99]. Nevertheless, the use of these flexible advance reservations for resource co-allocation has been barely explored [66].

By introducing flexibility to the advance reservations of co-allocation requests [66], schedulers can hence reschedule them to increase system utilisation and reduce response time of both local and multi-site jobs. This is particularly necessary due to the wrong estimations provided by users [73, 85, 118].

Little research has been devoted to resource co-allocation with rescheduling support

[3, 4]. Therefore, the major contributions of this chapter are:

- A *resource co-allocation model based on flexible advance reservations and process remapping, which allows the rescheduling of multi-site parallel jobs*. The flexible advance reservations are used to shift the start time of the job components, whereas the process 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;
- An *evaluation of scheduling co-allocation requests considering both user-estimated and actual run times, as well as response time guarantees for local and multi-site jobs*. Current research on co-allocation assumes accurate estimation of application run times 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 analysed several metrics to have a better understanding of the improvements achieved here. We also discuss issues on deploying the co-allocation policies on real environments.

4.2 Environment and Application Model

A metascheduler books resources across multiple autonomous sites to execute parallel jobs (Figure 4.1). Each site has its own scheduling queue and policies to manage both local and external jobs. As resource providers rely on inaccurate run time estimations, they must update their queues to fill fragments in order to produce better schedules. Therefore, they may also need to modify parts of a co-allocation request. However, all jobs from the same application must be synchronised, 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.

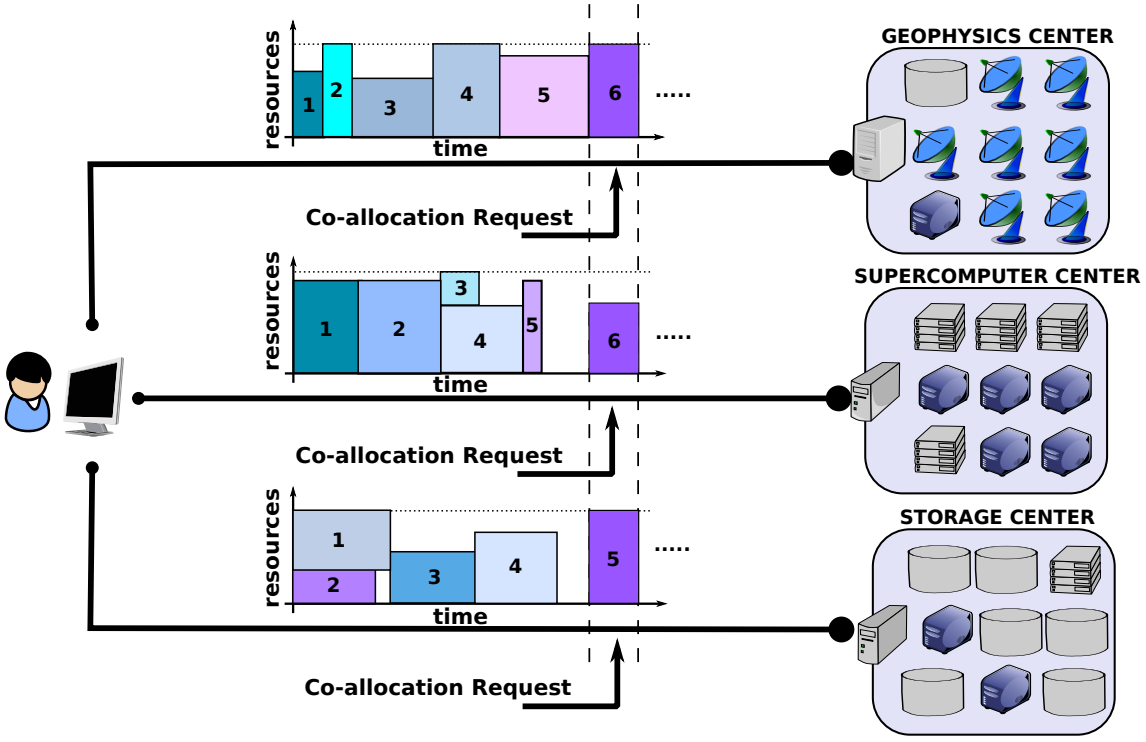


Figure 4.1: Metascheduler co-allocating resources from three providers.

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 optimise 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 [85].

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. A sub-request has a limited number of processors, which is dictated by the capacity of the machine m_i. However, the number of processes of each application component is determined by the user, which can exceed the number of processors.

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

4.3 Flexible Resource Co-Allocation

The flexible resource co-allocation (FlexCo) model proposed here is inspired by existing work on flexible advance reservations [47, 67, 86, 87, 98, 99]. A request, or a metajob, 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 jobs that are submitted to different machines. Each job may have a different number of resources with different capabilities. The following parameters and notations represent a metajob j based on the FlexCo model:

- $R_j^{m_k}$: number of processors required in each machine m_i, where k is the total number of jobs of the metajob 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 4.2): (i) *Start time shifting:* changes the start time according to the relaxed time interval—the change must be the same for all jobs coming from the same metajob; and (ii) *Processor remapping:* changes the number of required resources of two or more jobs. Combining both operations is also important for the scheduler. In Figure 4.2, we observe that after using the process remapping operation, it is possible reduce the metajob response time by shifting the jobs associated with it. The schedulers perform these operations while jobs are waiting for resources, and not during run time. The idea here is to redefine the request specifications, not to migrate jobs [81].

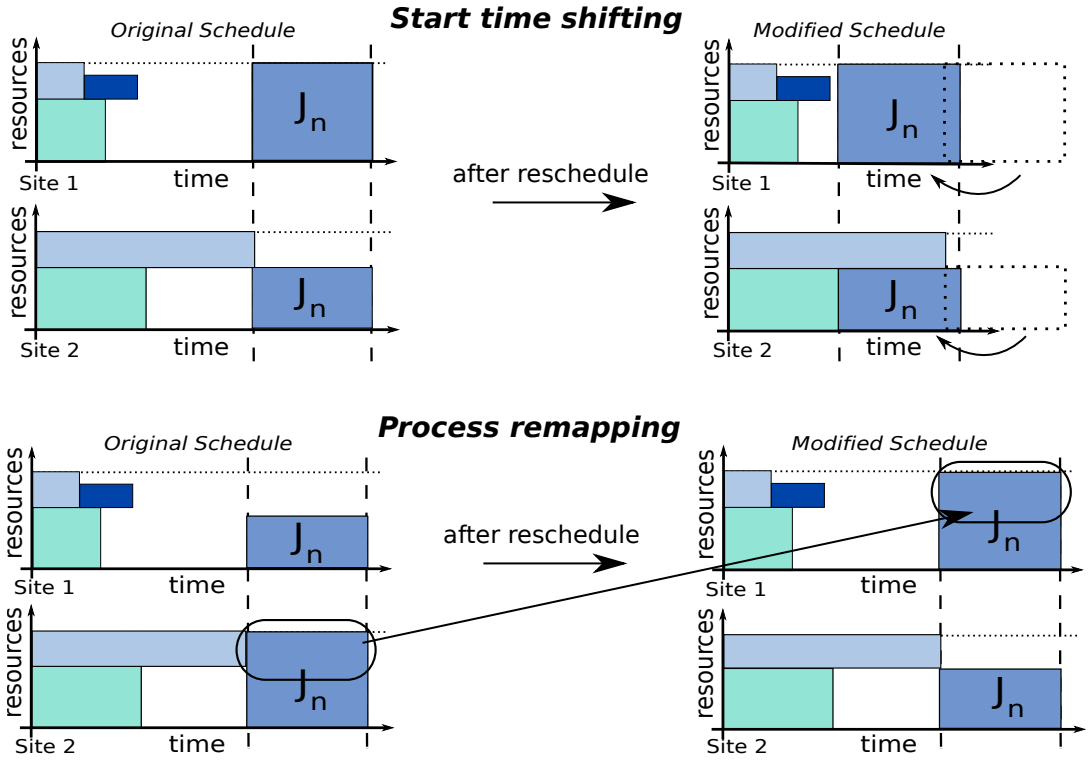


Figure 4.2: Operations of a FlexCo request.

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 application independent in the sense that it is only a shift on the start time of the user application.

Process Remapping (Remap): A user requiring a certain number of resources tends to decompose the metajob 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 processes once the jobs 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 jobs. Branch-and-bound-based solvers for optimisation 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 clusters required by a co-allocation request, which has a direct impact on the network utilisation.

4.4 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. The scheduling involves the manipulation of time slots, which are data structures composed of four values: ts^{id}: identification; ts^{s}: start time; ts^{c}: completion time; and ts^{n}: number of resources available in this time slot.

4.4.1 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 multi-site jobs into a single site over time due to the rescheduling.

4.4.2 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

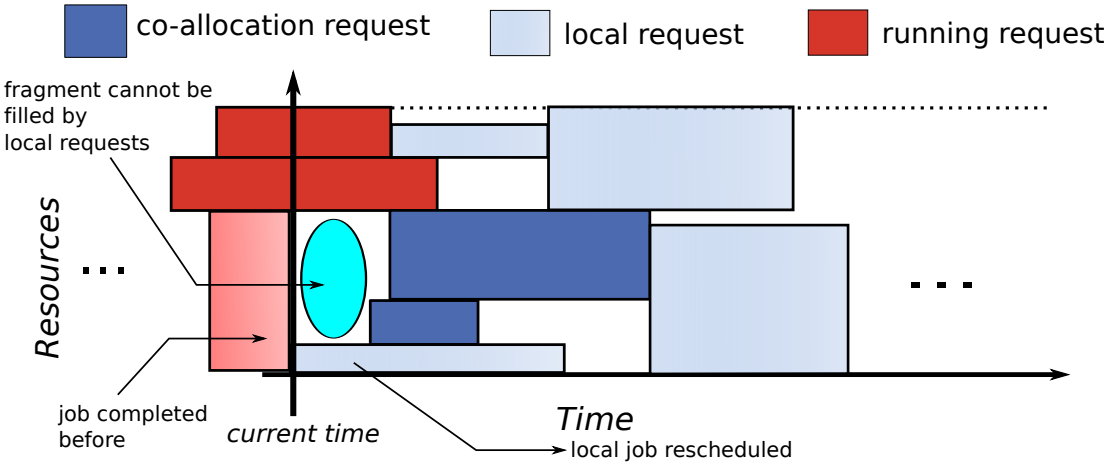


Figure 4.3: Reschedule of co-allocation jobs to fill queue fragment.

Algorithm 3: Pseudo-code for rescheduling jobs, which is executed on the local schedulers when a job completes before the expected time.

```
1  coallocRescheduled ← 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  while there are idle resources do
7      for ∀ multisite jobs j_i in Q^w do
8          Contact metascheduler to reschedule j_i
9          if T_{j_i}^c ≤ previousT_{j_i}^c then
10             coallocRescheduled ← true
11 if coallocRescheduled = true then
12     for ∀ j_i ∈ Q^w do
13         if j_i is local job then
14             Schedule job with backfilling
```

to optimise 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 fragments generated due to the early completion of a job (Figure 4.3). The local schedulers use Algorithm 3 to reschedule jobs whenever a job completes before its estimated time. The rescheduling is based on the *compressing* method described by Mu'alem and Feitelson [85], 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

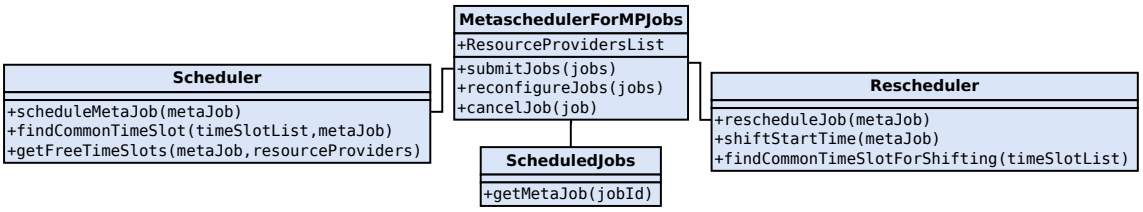


Figure 4.4: Class diagram for the metascheduler of message passing applications.

a list of jobs sorted according to start time instead of sorting them when rescheduling (Line 2).

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 4.4.1). The main differences are that (i) for the **Shift** operation, the metascheduler asks for time slots only from those resource providers that hold the jobs 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.

4.4.3 Implementation Issues

Local schedulers usually handle jobs by using their IDs. In order to implement the co-allocation policies, jobs require an additional ID, which is composed of the metascheduler ID that necessary to contact the respective metascheduler managing the job, and the ID used internally by the metascheduler, which assists the metascheduler to identify the jobs. The co-allocation request also needs an estimation of the network overhead required to execute the application in multiple providers.

Figure 4.4 represents a simplified version of the class diagram for the metascheduler. There are four main components: the metascheduler main class, scheduler, rescheduler, and a list of scheduled jobs. The main responsibilities of the metascheduler class are to submit jobs to resource providers, and reconfigure and cancel jobs in case of Remap operation is used. The complexity of implementing the scheduler and rescheduler classes lies on finding the common time slot (Figure 4.5), which has to consider the requirements of the metajob. The metascheduler also contains a list of scheduled jobs to keep track where each job of a meta job is scheduled and to find metajobs based on job IDs given by providers at the rescheduling phase.

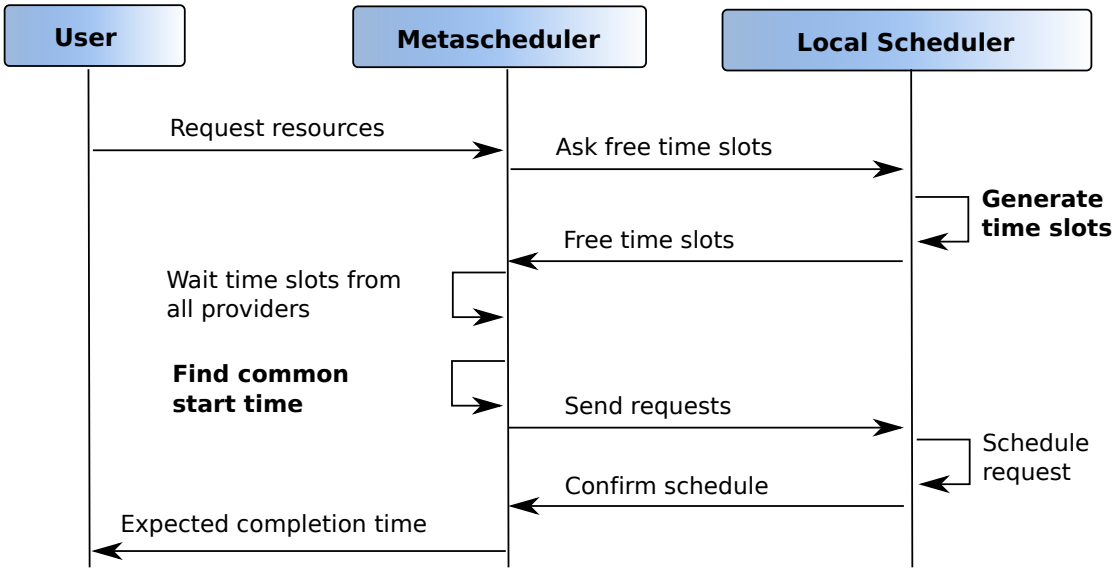


Figure 4.5: Sequence diagram for the initial co-allocation.

4.5 Evaluation

We used the simulator PaJFit and real traces from supercomputers available at the Parallel Workloads Archive 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 description of the environment set up and metrics followed by the results and our analysis.

4.5.1 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 30 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.[1] Table 4.1 sum-

[1]To vary the load we used a strategy similar to that described by Shmueli and Feitelson to evaluate their backfilling strategy [104], but we fixed the time interval and included more jobs from the trace.

Table 4.1: Summary of workloads.

Location	Procs	Jobs	Actual	—	Estimated Load
Cluster 1	430	6,668	57%	—	192%
Cluster 2	240	1,632	54%	—	107%
Cluster 3	128	2,815	51%	—	288%
External	798	180	16%	—	55%
External	798	288	31%	—	91%

marises the workload characteristics. We 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. [45] shows that co-allocation is advantageous when the penalty for network overhead is up to 25%. Therefore, we limited the network overhead under this value.

We evaluated the system utilisation and response time, i.e. the difference between the job completion time and submission time. In addition, we analysed the behaviour of multi-site jobs due to rescheduling. We investigated these metrics according to the run time estimation precision of all jobs in the system. We used the original values, original plus 25% and 50% of increase in accuracy. The metrics are also a function of the external load.

4.5.2 Results and Analysis

Figure 4.6 presents the global response time reduction as a function of the run time estimation precision and external load. In order to analyse these results, we have separated them for local and external jobs; Figures 4.7 and 4.8 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, 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 have this problem, and therefore, by shifting the multi-site jobs, some local jobs can be shifted as well. We also observe that in most scenarios, Shift+Remap provides better schedules than only the Shift operation. That is because of the higher flexibility the Shift+Remap gives to the scheduler and by

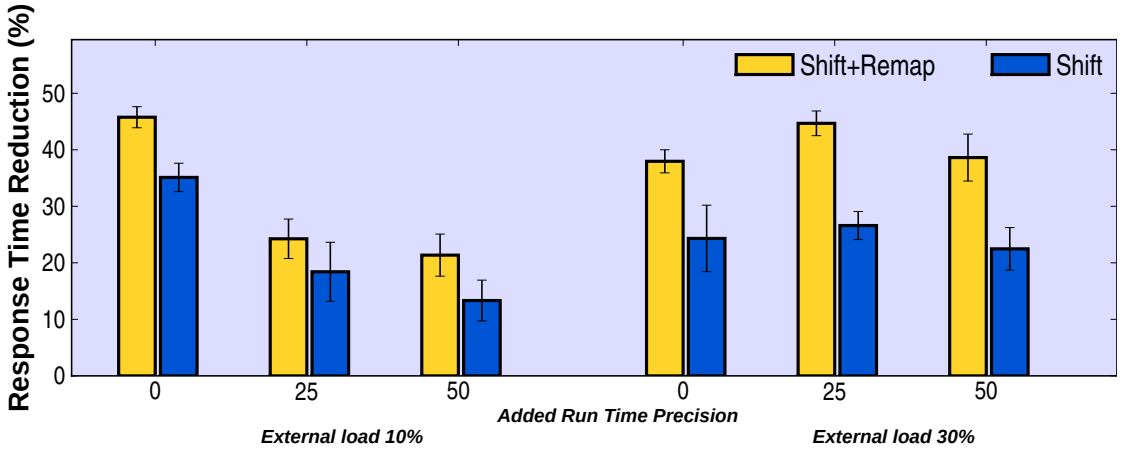


Figure 4.6: Global response time reduction as a function of the run time estimation precision and external load.

the fact that some multi-site jobs can be remapped to a single cluster, which reduces up to 25% of the execution time for these jobs.

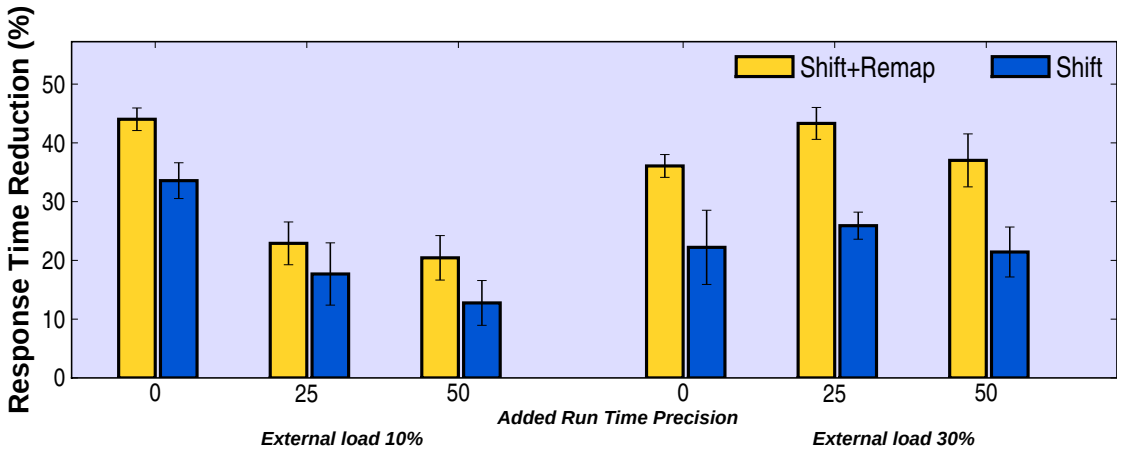


Figure 4.7: Response time reduction of local jobs as a function of the run time estimation precision and external load.

Filling the gaps using FlexCo requests has a direct impact on the system utilisation, as can be observed in Figure 4.9. For system utilisation, we 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. However, for external load of 10% and increase of precision for 25% the difference between Shift+Remap and Shift is minimum, and for precision increase of 50%, utilisation for Shift+Remap is lower than for Rigid Advance Reservations. This difference does not reflect the improvement of Shift+Remap observed in the response time, and happens because Shift+Remap reduces system utilisation when multi-sites jobs remapped to

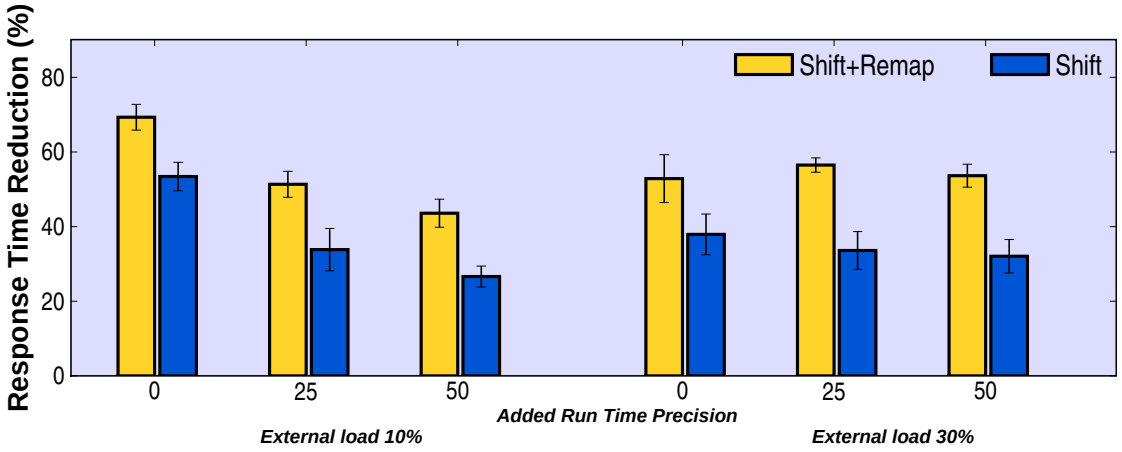


Figure 4.8: Response time reduction of external jobs as a function of the run time estimation precision and external load.

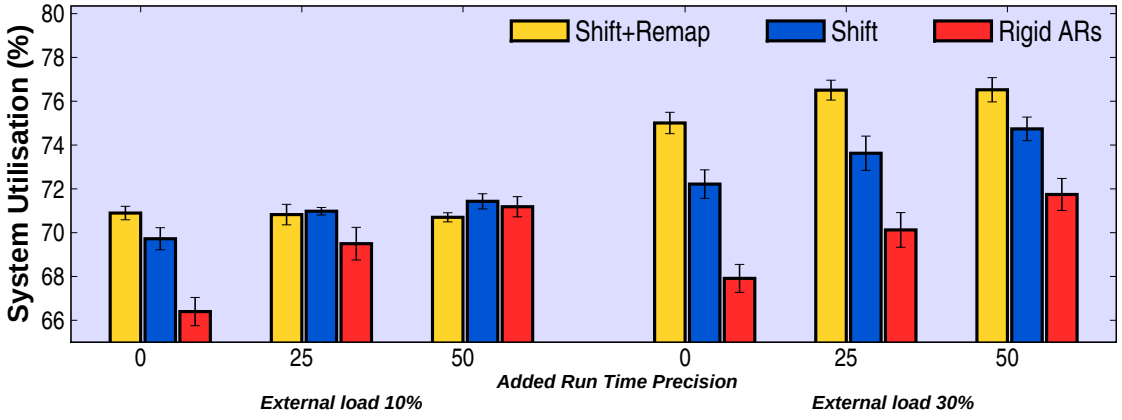


Figure 4.9: Global utilisation as a function of the run time estimation precision and external load.

a single-site, which eliminates network overhead. For both response time and system utilisation, we observe that the higher the imprecision on run time 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 metajobs remapped to a single cluster due to the rescheduling. From Figure 4.10, we observe that approximately 15% of multi-site jobs, that otherwise would use inter-cluster communication, were migrated to a single cluster. Different from the utilisation and response time, this metric does not present a smooth behavior; moving jobs to a single site is highly dependent on the characteristics and packing of the jobs.

Figure 4.13 illustrates the percentage of multi-site jobs that are initially submitted to more than one site and 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

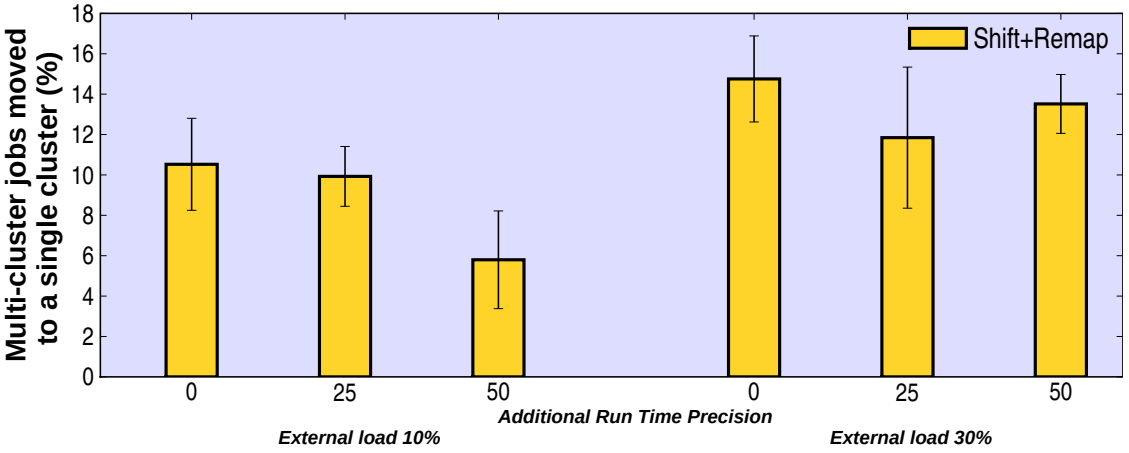


Figure 4.10: Percentage of multi-site jobs moved to a single cluster as a function of the run time estimation precision and external load.

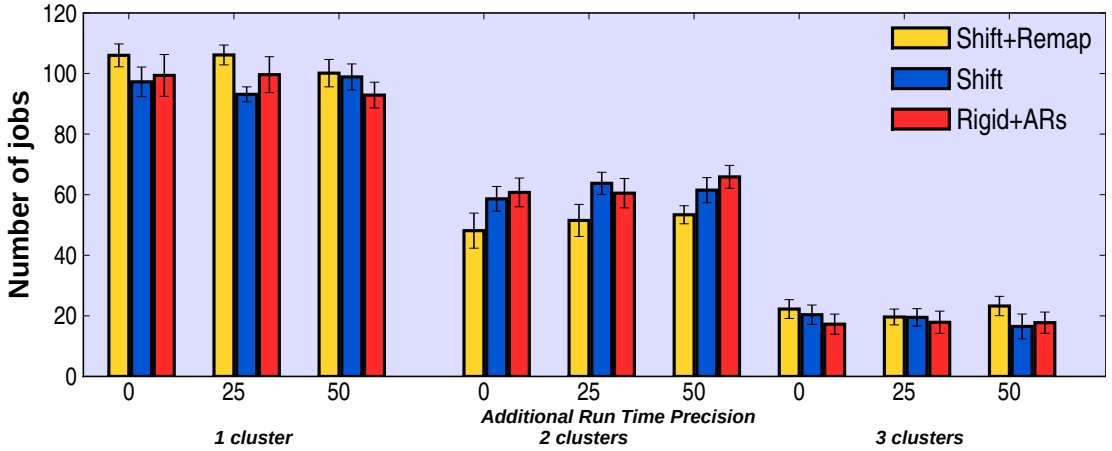


Figure 4.11: Number of clusters used by job with system external load=10%.

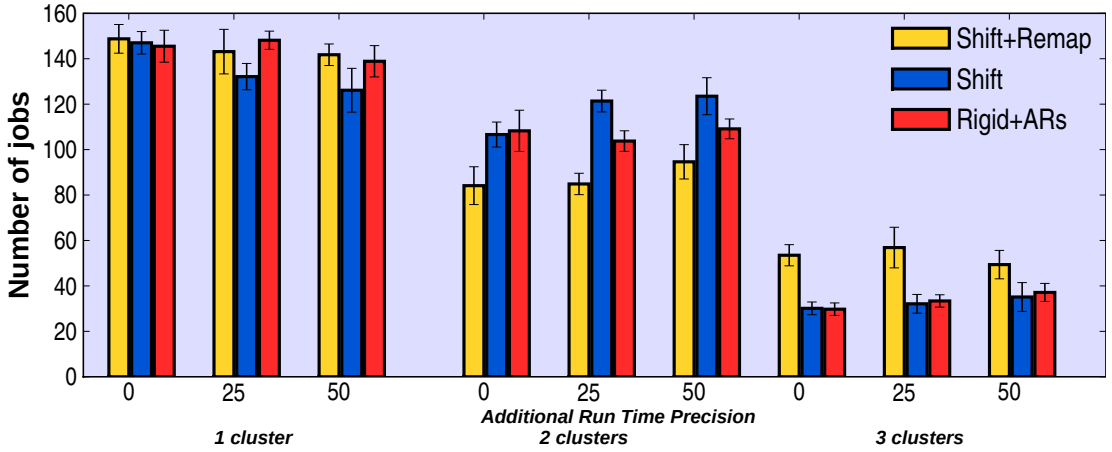


Figure 4.12: Number of clusters used by job with system external load=30%.

already showed, Shift+Remap provides a higher impact on the improvement.

Figure 4.11 and 4.12 illustrate the total number of clusters used by the external jobs.

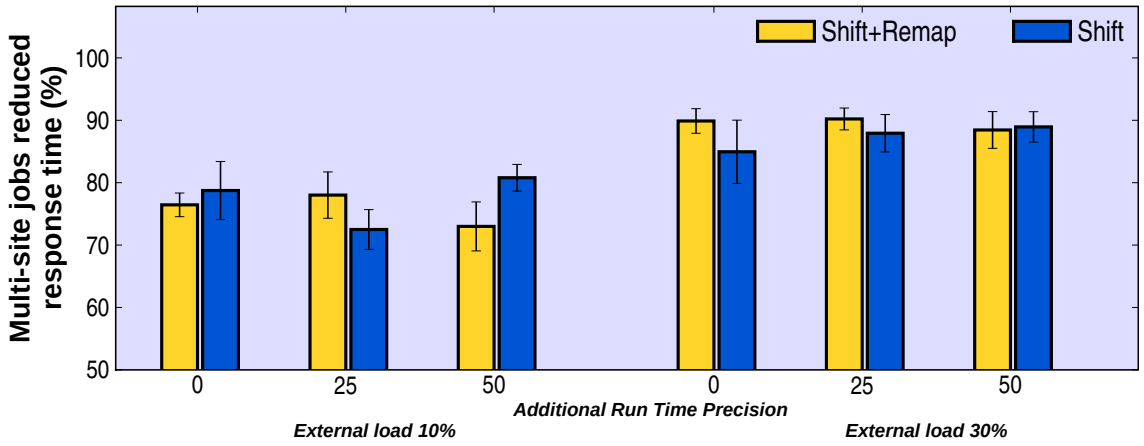


Figure 4.13: Percentage of multi-site jobs that reduce their response time due to rescheduling as a function of the run time estimation precision and external load.

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 fragments 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.

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.

4.6 Conclusions

This chapter has shown the impact of rescheduling co-allocation requests in environments where resource providers deal with inaccurate run time 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 process 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 optimisation 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.

This chapter meets two objectives proposed in Chapter 1 for message passing applications: understand the impact of inaccurate run time estimates, and design and evaluate co-allocation policies with rescheduling support. Process remapping has practical deployment difficulties, especially for environment containing clusters with different resource capabilities. Therefore, next chapter discusses how to overcome these difficulties and the kind of applications that can benefit from the process remapping operation.

Chapter 5

Implementation of Automatic Process Mapping using Performance Predictions

This chapter presents a performance prediction mechanism to enable automatic process mapping for iterative parallel applications. Iterative applications have been used to solve a variety of problems in science and engineering. Metaschedulers can use performance predictions for both the initial scheduling and the rescheduling of applications. We performed experiments using an iterative parallel application, which consists of benchmark multiobjective problems, with both synchronous and asynchronous communication models on Grid'5000. The results show that it is possible to generate performance predictions with no access to the user application source code. In addition, metaschedulers using predictions can reschedule applications to faster or slower resources without asking users to overestimate execution times.

5.1 Introduction

Co-allocating resources from multiple clusters is difficult for users, especially when resources are heterogeneous. Users have to specify the number of processors and usage times for each cluster. Apart from being demanding to estimate application run times, these static requests limit the initial scheduling due to the lack of resource options given by users to metaschedulers. In addition, static requests prevent rescheduling of applications to other resource sets; applications may be aborted when rescheduled to slower resources; unless users provide high run time overestimations. When applications are rescheduled to faster resources, backfilling [85] may not be explored if estimated run times are not reduced. Therefore, performance predictions play an important role for automatic scheduling and rescheduling.

The use of performance predictions for scheduling applications have been extensively studied [56, 63, 101, 103]. However, predictions have been used mostly for single-cluster applications and require access to the user application source code [15, 16, 102, 130]. Parallel applications can also use multiple clusters and performance predictions can assist their deployment. One application model for multi-cluster environments is based on iterative algorithms, which has been used to solve a variety of problems in science and engineering [11], and has also been used for large-scale computations through the asynchronous communication model [42, 76].

This chapter proposes a resource co-allocation model with rescheduling support based on performance predictions for multi-cluster iterative parallel applications. Iterative applications with regular execution steps can have run time predictions by observing their behavior with a short partial execution. This chapter also proposes two scheduling algorithms for multi-cluster iterative parallel applications based on synchronous and asynchronous models. The algorithms can be utilised to co-allocate resources for iterative applications with two heterogeneity levels: the computing power of cluster nodes and process sizes.

We performed experiments using an iterative parallel application, which consists of benchmark multiobjective problems, with both synchronous and asynchronous communication models on Grid'5000. The results using our case study application and seven resource sets show that it is possible to generate performance predictions with no access to the user application source code. By using our co-allocation model, metaschedulers become responsible for run time predictions, process mapping, and application rescheduling; releasing the user from these difficult tasks. The use of performance predictions presented here can also be applied when rescheduling single-cluster applications among multiple clusters.

5.2 Iterative Parallel Applications

Iterative algorithms have been used in a large class of scientific and engineering problems, especially using optimisation techniques such as genetic algorithms, particle swarm and ant colony optimisation algorithms [79, 106]. These algorithms consist of a set of computations inside a loop, which can be partitioned to execute in parallel. The main communication models for these algorithms are synchronous and asynchronous. The second model is becoming popular since it opens the opportunity for another parallel application model for large-scale systems [11, 42, 76, 106]. The reason is that asynchronous model can handle inter-process communication on high latency environments. The next sections present the two execution models.

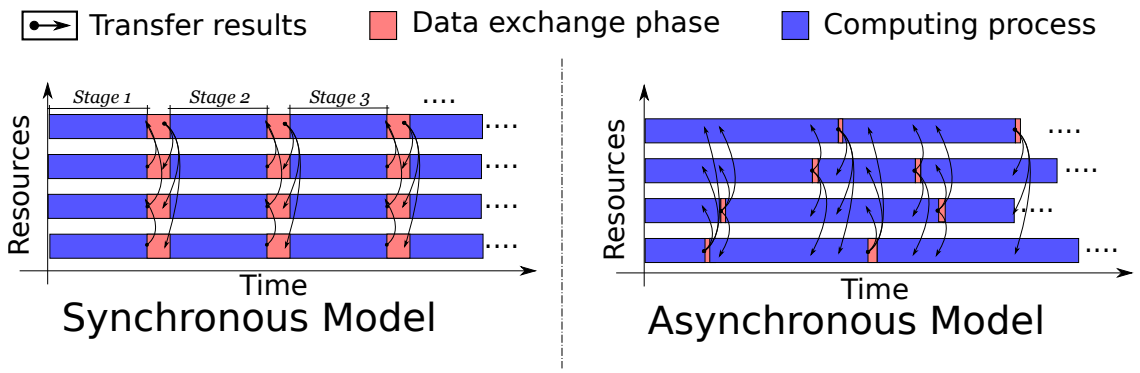


Figure 5.1: Synchronous and asynchronous communication models.

5.2.1 Synchronous Model

In this model, application processes are distributed to machines and results are merged once they have completed the number of iterations specified by the user. We call *stage* a set of processes followed by a merging, or data exchange, process (Figure 5.1). When a stage finishes, its results are redistributed to the machines that then execute the next processes. The execution completes when all processes achieve the total number of iterations specified by the user. Processes may finish at different times due to heterogeneity in the system and application. In order to avoid idle processor time, depending on the application, it is possible to keep iterating all processes in a stage until they reach the minimum number of required iterations. For evolutionary-based optimisation applications, keep iterating processes only improves the results, without negative side effects. Therefore, processes running on faster machines and/or using parameters that require less computing power iterate more than the others.

There are several approaches to synchronise data among the application processes. One of them is to have a master node for each cluster involved in the execution, responsible for merging the results of nodes in its cluster. This master node sends the results to the master node with better aggregate CPU, called global coordinator, which merges all the results and sends the merged result to all clusters. After that, processes start the new stage. Hierarchical data exchange inside a cluster can also be used to optimise the synchronisation phase.

5.2.2 Asynchronous Model

For this model, when a process finishes, it distributes its results to other processes asynchronously, merges its results with the last results from other processes, and continues its execution (Figure 5.1). This prevents any idle time, and provides better support for heterogeneous machines and processor fault tolerance. Note that as processes execute in

multiple clusters, the impact of wide-area communication has to be minimised as much as possible [12]. Indeed, asynchrony masks communication and computation, and therefore minimises this impact.

The application following the asynchronous model can use similar approaches from the synchronous model to distribute data among processes. The difference is that a process does not wait to receive data from other processes.

Resource co-allocation is important for both models since: for the synchronous model, it prevents processes from being idle and thus completes the execution faster; whereas for the asynchronous model, it increases interaction among results of the application processes.

5.3 Co-Allocation based on Performance Predictions

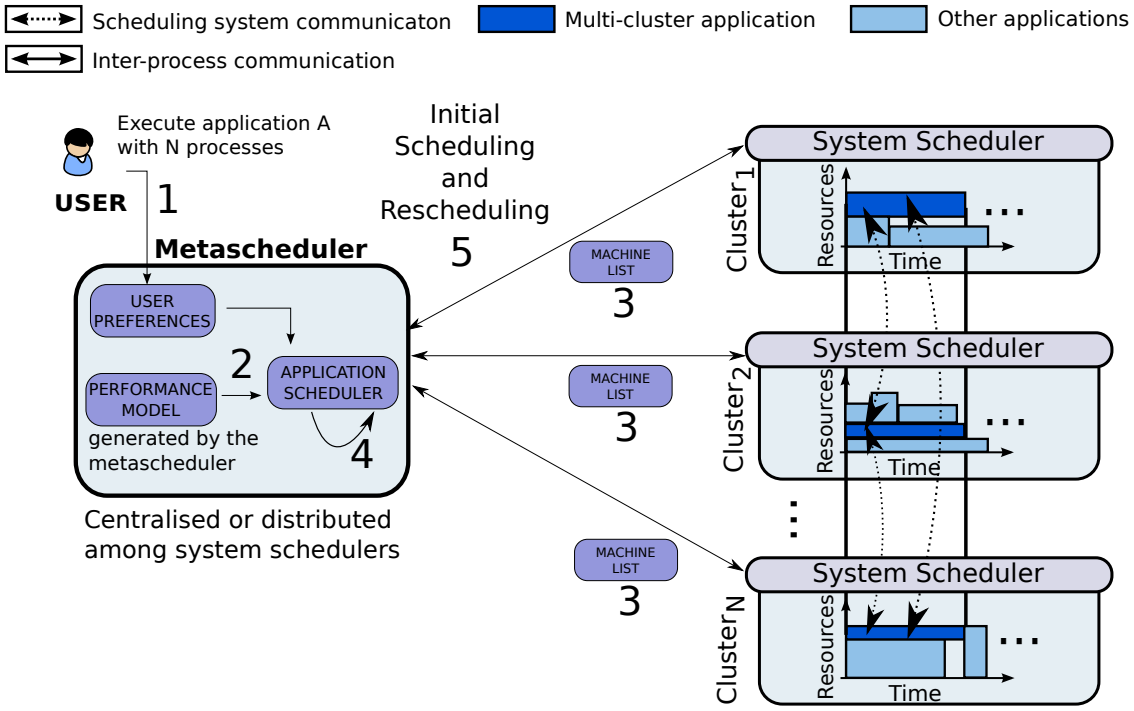


Figure 5.2: Deployment of iterative parallel applications in multiple clusters following synchronous and asynchronous models.

The metascheduler, responsible for co-allocating resources from multiple clusters, relies on four components to enable automatic process selection and rescheduling support. Here we present the sequence of steps to co-allocate resources using performance predictions and an overview of the metascheduler components (component details are presented in the next sections), as illustrated in Figure 5.2:

- **User preferences (Step 1):** users specify the total number of processes and application parameters. Users have to also specify script to determine application throughput on a resource. Section 5.3.1 details how to provide the script.
- **Performance predictions (Step 2):** the metascheduler executes the script to collect application throughput on multiple resource types. Section 5.4 details an example of how to generate predictions.
- **Machine list (Step 3):** the metascheduler contacts system schedulers to obtain a list of resources that can be used by the user application. Section 5.3.3 describes the interaction between metascheduler and system schedulers.
- **Application scheduler (Step 4):** uses the machine list, performance predictions, and user preferences to generate a schedule of the application processes. This component generates a set of scripts used to deploy the application. Section 5.3.2 presents two application schedulers for iterative parallel applications.

A resource co-allocation request based on performance predictions can use different resource sets automatically. This is particularly important when rescheduling applications on multiple clusters. Note that rescheduling is allowed when requests are still in the waiting queues, and hence is different from migrating processes at run time.

The **system scheduler** uses the application scheduler to obtain the application estimated run time. This estimation can be used for both the initial scheduling and the rescheduling (**Step 5**).

5.3.1 Generation of Run Time Predictions

Run time predictions are obtained by executing an application process until the throughput (iterations/second) becomes steady. One way to obtain the throughput is to write a simple script that runs an application process with a given number of iterations from a list (e.g. 1, 3, 5, 10, 50, 100...) and stops when the throughput becomes steady or the list finishes. Another way is to ask the application (if it supports) to write output files for each iteration executed and then check the time interval between iterations. For both ways, no application source code is required. The time to obtain the throughput depends on the application and its input parameters. The more node types the script is executed, the more schedule options can be granted. The metascheduler can ask system schedulers to use the otherwise wasted time of queue fragments to execute the script or submit it as a processor request to each cluster before the actual application execution.

Once the throughput is available, the application scheduler can calculate the execution time of each process by multiplying the throughput by the number of iterations specified

by the user. The application scheduler determines the process locations and the overall execution time, which is described in the following section. Network overhead can also be included in the overall execution time, which can be estimated by the amount of data that has to be transferred among application processes and the network latency among clusters.

5.3.2 Application Schedulers

We developed two application schedulers, one for each communication model. These schedulers are frameworks for iterative parallel applications with two heterogeneity levels: computing power of cluster nodes and process sizes. The latter level is used for applications that have processes with different computing power requirements.

The input parameters for the schedulers are: list of resources, number of iterations per process, number of processes, process sizes, performance predictions for computation and inter-process communication. For the synchronous model (Algorithm 4), the scheduler sorts resources by their computing power, which is determined during the performance prediction phase (Line 1) and assigns processes according to their process sizes (more CPU consuming ones first) for each stage (Lines 6-12). The number of stages is determined by the total number of resources and total number of processes specified by the user (Line 3). As resources and process sizes make the process execution times vary, more iterations can be added to processes that would be waiting for slower processes (Lines 14-15). For the asynchronous model (Algorithm 5), the scheduler assigns one process of each process size to the resource with the earliest completion time (Lines 3-6). The scheduler gives priority to longer processes (Line 1), but each group of processes with the same size receives one resource per round (Line 4).

5.3.3 Scheduling and Rescheduling

The interaction between system and application schedulers takes place during the initial scheduling and rescheduling of a request. The **initial scheduling** is triggered when a metascheduler requests for machines, whereas the **rescheduling** is triggered when a system scheduler wants to update its queue. For both cases, interaction between system and application's scheduler comprises three steps:

1. The application scheduler asks the system scheduler for the earliest n machines available;
2. The application scheduler generates a schedule containing the application estimated execution time;

Algorithm 4: Pseudo-code for generating the schedule using the synchronous model.

```
1  Sort resources by decreasing order of computing power
2  Sort processes by decreasing order of CPU demand
3  numberOfStages ← numberOfProcesses / numberOfResources
4  numberOfProcesssesPerProcessSize ← numberOfResources / numberOfSizes
5  maxCompletionTime ← 0
6  for each numberOfStages do
7      r ← 0 (r: resource id)
8      for each processSize do
9          for 0 to numberOfProcessesPerProcessSize do
10             Schedule a process of this processSize to resource r
11             Update MaxCompletionTime
12             r ← r + 1
13     /* optional /*
14     for each resource r do
15         Make last process on this resource complete at MaxCompletionTime by
           increasing the number of iterations
```

Algorithm 5: Pseudo-code for generating the schedule using the asynchronous model.

```
1  Sort processes by their sizes. Decreasing order of CPU demand
2  n ← 0
3  while n < numberOfProcesses do
4      for each process size do
5          Select resource r with earliest completion time
6          Schedule process to resource r
7      n ← n + numberOfProcessSizes
```

3. The metascheduler, or a system scheduler, verifies with the other system scheduler(s) whether it is possible to commit requests;
4. Step 1 is repeated if it is not possible to commit requests. A maximum number of trials can be specified.

Note that by using this algorithm, resource providers can keep their schedules private [44]. Alternatively, in Step 1, the application scheduler could ask system schedulers for all free time slots (which are available time intervals for resources) and then minimise interactions between the metascheduler and the system scheduler.

Rescheduling frequently occurs when applications finish before the estimated time [89]. Rescheduling can also be necessary when resources fail or users modify/cancel requests. Whenever one of these events arise, the system scheduler triggers the rescheduling process.

5.4 Evaluation

This section describes the experiments to show how much time the metascheduler requires to generate application throughputs, the accuracy of prediction run times, and the impact of these predictions on the rescheduling. We used an iterative parallel application based on evolutionary algorithms, which consists of benchmark multiobjective problems, with both synchronous and asynchronous communication models. We conducted the experiment in Grid'5000, which consists of clusters in France dedicated to large-scale experiments [18]. The clusters are spaced-shared machines shared by multiple applications. From these clusters, we selected seven resource sets to execute the application. These sets are examples of resources that are dynamically chosen by the metascheduler to execute our case study application. Experiments on how the metascheduler selects resource sets, and when it reschedules multi-cluster applications are described in the previous chapter. Before describing the experiment, we provide an overview of the benchmark application.

5.4.1 Case Study of An Iterative Parallel Application

EMO (Evolutionary Multi-objective Optimiser) is an iterative benchmark application based on Genetic Algorithms [38] and uses the concept of topology to drive the evolutionary process [69, 122]. A **topology** is a graph interconnecting individuals of a population and is characterised by: (i) the node degree representing the average number of connections for each individual; and (ii) the path line defining the number of hops to be crossed on average to connect individuals. Individuals are chosen to exchange their information according to the topology links; process that determines how the current solutions of the approximation sets are selected to produce a new generation of solutions. The use of topologies provides better solutions in general but requires a large number of elements in the approximation sets, which becomes compute intensive for non-trivial problems. Moreover, topologies have impact on the execution time: sparsely connected topologies imply faster execution times than fully connected ones, but propagate more slowly the updates in the approximation set. Figure 5.3 presents the topologies used in the experiments.

EMO is an application that runs from the operating system shell and can be controlled by a set of 25 command line parameters. The main input parameters for our case study are: the topology used to produce new solutions, the number of iterations, the size of the approximation set, and the optimisation multi-objective function. As execution outcome, EMO produces: the final solution set (*Pareto Front*) and the approximation set that generated this solution. For the distributed version of EMO, we introduced a coordination layer that reiterates EMO processes by feeding them with updated information on the approximation set. An additional component, called EMOMerge, merges and partitions

the approximation sets generated at each stage of the execution for synchronous model. For the asynchronous model, EMOMerge uses the last results received from the other processes.

Epsilon Indicator. Multi-objective functions identify a multi-dimensional space whose properties are difficult to visualise effectively. It is then necessary to adopt synthetic measures that generally aggregate information about the quality of a solution into one number called indicator. In our case study, we use a quality indicator called *Epsilon* [135], which is based on the distance between a reference solution and the *Pareto front*.

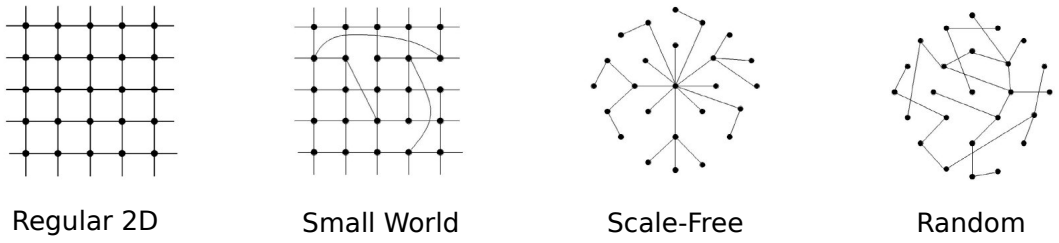


Figure 5.3: Topologies of EMO application.

5.4.2 Experimental Configuration

We used seven clusters located in three cities in France with heterogeneous computing capabilities (Figure 5.4). Table 5.1 presents an overview of the node configurations for these clusters. Machines in the same location share the same file system, which simplifies file transfer between nodes of clusters in the same site.

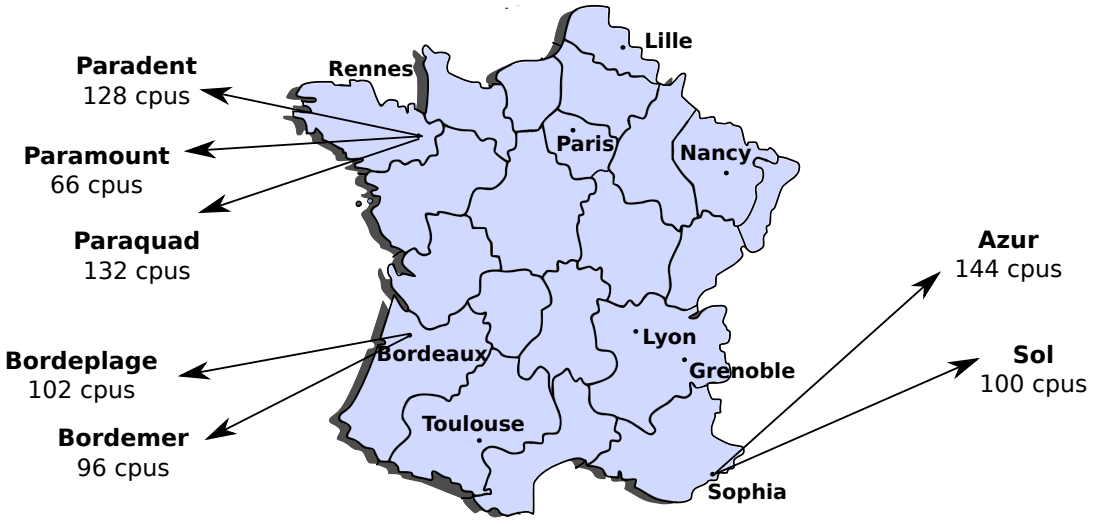


Figure 5.4: Location of resources inside Grid'5000.

Table 5.1: Overview of the node configurations.

Cluster	Location	CPUs' Configuration	Cores per Node	Total CPUs
azur	Sophia	AMD Opteron 246 2.0 GHz	2	144
sol	Sophia	AMD Opteron 2218 2.6 GHz	4	100
bordemer	Bordeaux	AMD Opteron 248 2.2 GHz	2	96
bordeplage	Bordeaux	Intel Xeon EM64T 3 GHz	2	102
paraquad	Rennes	Intel Xeon 5148 2.33 GHz	4	132
paramount	Rennes	Intel Xeon 5148 2.33 GHz	4	66
paradent	Rennes	Intel Xeon L5420 2.5 GHz	8	128

Table 5.2: Resource sets selected by the metascheduler on seven clusters in Grid'5000.

Clusters/Resource Sets	1	2	3	4	5	6	7
paradent	32						
paramount	04		20			04	
paraquad	04	20					
sol		12	12	20	20	12	
bordemer		02	08	20	08	06	20
azur		06			06	10	
bordeplage					06	08	20

Application configuration. We configured EMO to solve the DTLZ6 function with a setup of 10 objectives. This function is one of the most compute intensive functions in the benchmark proposed by Deb et al. [39]. We have used four topologies: Regular 2D, Scale-Free, Small-World, and Random [69]; and 1024 individuals for each EMO process with a minimum of 200 iterations each process. The application was deployed on 40 cores using 480 EMO processes, i.e. 120 processes for each topology in order to optimise 10-objective functions.

Resource sets. We configured the metascheduler to access seven resource sets in Grid'5000. Table 5.2 presents the list of clusters and number of cores used in each resource set. These resource sets are examples of resources chosen dynamically by the metascheduler for the EMO application. The clusters are space-shared machines, and hence the resource sets are dedicated to the application, which is a common set up for existing HPC infrastructures.

Inter-process communication overhead. Communication is based on file transfer between processes during the merging phases. The files transferred among the sites are 500 Kbytes on average. Therefore, the cost of transferring the files is minimum compared to the total application execution time, which takes minutes. However, file transfer relies on secure copy (scp) command, which requires authentication. Therefore, we used inter-site file transfer as 800 ms.

Metrics. To evaluate co-allocation based on performance predictions and their impor-

tance on rescheduling, we measured the *time to generate predictions* and analyze the *difference between the actual and the predicted execution times*. The prediction for each resource set assists schedulers to know whether they can reschedule the new sub requests into the scheduling queues of other schedulers. Therefore, we measured the impact of predictions for the application rescheduling. To understand the application output on different resource sets, we also measured *execution times* and the *Epsilon indicator* for the synchronous and asynchronous models.

5.4.3 Results and Analysis

Performance predictions generation. The metascheduler executed an independent process in six nodes with different computing power and the throughputs became steady before 250 iterations for all processes. Table 5.3 presents the throughputs for each cluster and topology. We observe that sparsely connected networks, such as the Regular 2D and the Scale-Free networks, imply a faster iteration time than more connected networks, such as the Random topology. For the Random topology, the value of the path line was around five times smaller than the path line of the Regular 2D. This means that, on average, the selection of the individual to exchange information requires traversing a list five times smaller than for the Random topologies; and this reflects the execution time difference. Table 5.4 shows the time spent to obtain the throughput for each node type and topology. Most of the throughput values took less than one minute to be obtained. The total CPU time to generate the predictions is only 2% of the overall CPU time of the longest experiment. As the predictions can be re-utilised or used by longer executions or executions with more processes, the cost for generating predictions tends to be zero.

Table 5.3: Throughput (iterations/sec.) for each machine type and topology.

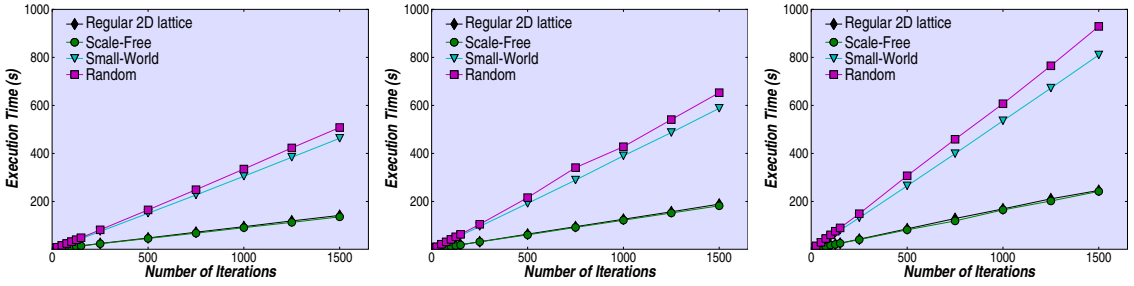
Cluster/Topology	Regular 2D	Scale-Free	Small-World	Random
paradent	10.87	11.36	3.57	3.29
paramount	10.00	10.42	3.33	3.05
sol	9.26	9.62	3.09	2.81
bordemer	7.81	7.81	2.60	2.38
azur	7.14	7.35	2.34	2.14
bordeplage	5.81	6.10	1.89	1.68

Regular behaviour. In order to show the regularity of the throughput, we configured the meta-scheduler to collect the throughput data until 1500 iterations. As we can see in Figure 5.5, which shows the execution times as a function of the topologies and number of iterations for three machine types in Grid'5000, EMO has a regular throughput over the iterations. This happens because EMO processes the similar amount of work in each

Table 5.4: Time in seconds to obtain the throughputs for each machine type and topology.

Cluster/Topology	Regular 2D	Scale-Free	Small-World	Random
paradent	23	14	42	46
paramount	25	15	45	49
sol	27	16	49	54
bordemer	32	19	58	63
azur	35	21	64	70
bordeplage	43	26	80	90

iteration, which is common for several iterative applications. Figures 5.6-5.9 represent the throughput (iterations/second) of an EMO execution using each topology on a single core of seven machine types as a function of number of iterations.



(a) Paramount. (b) Bordemer. (c) Bordeplage.

Figure 5.5: Execution times as a function of the topologies for three machine types in Grid'5000.

Accuracy of predictions. Figure 5.10 presents the predicted and actual execution times for synchronous and asynchronous models. Actual execution times are averages of five executions for each resource set. We observe that the execution time for the asynchronous model is shorter than the synchronous model for all resource sets. For the asynchronous model, all EMO processes execute the minimum required number of iterations, whereas for the synchronous model, EMO processes may execute more iterations in order to wait for processes that take longer. In addition, the difference between actual and predicted execution is on average 8.5% for synchronous and 7.3% for the asynchronous model. These results highlight that it is possible to reschedule processes on multiple clusters since schedulers can predict the execution time for different resource sets. Note that the predictions for the asynchronous model is slightly better than for the synchronous model. This reason is that the asynchronous model requires less accurate inter-process communication predictions than the synchronous model since the network overhead impact in the first model is minimum.

For the quality of the predictions (Figure 5.10), resource sets 2 and 3 present better results compared to the other sets for the synchronous model. This happens because

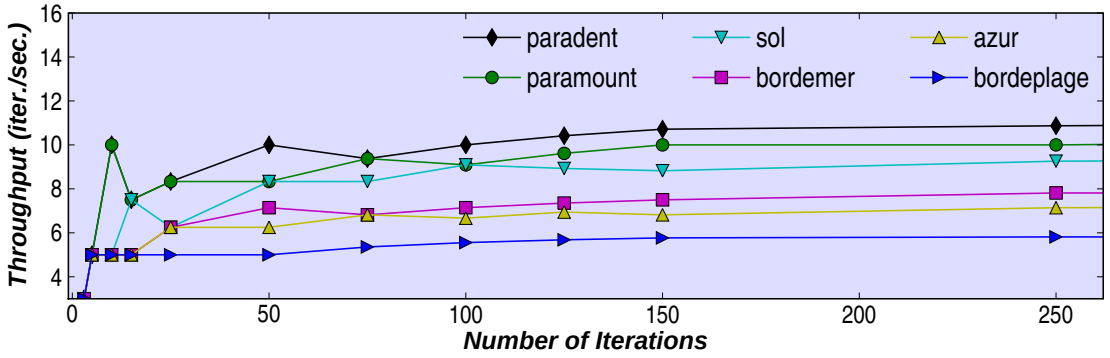


Figure 5.6: Throughput for the Regular 2D topology.

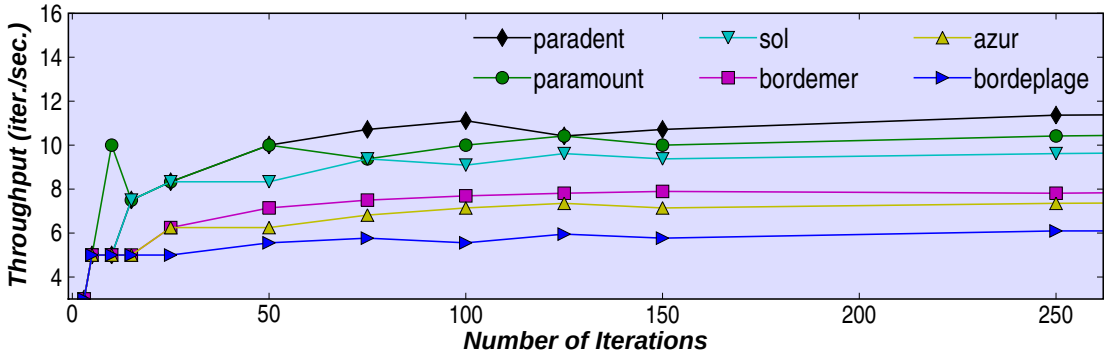


Figure 5.7: Throughput for the Small-World topology.

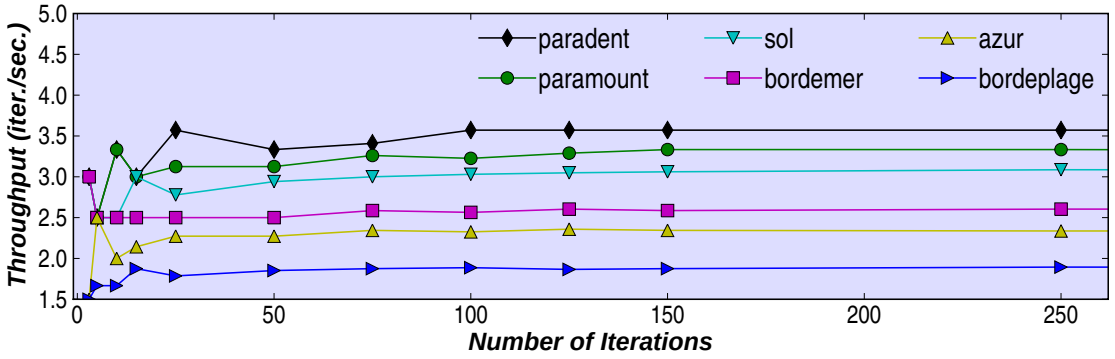


Figure 5.8: Throughput for the Scale-Free topology.

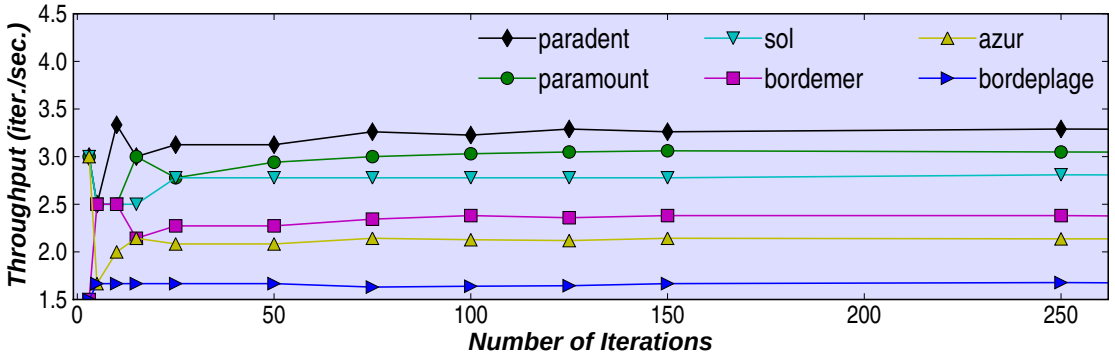


Figure 5.9: Throughput for the Random topology.

the merging phase is split by sites (locations in France). For these sets, three sites are used, and therefore the load for merging results is well balanced. Resource set 6 also comprises three sites, but only four resources in one of the sites. For the asynchronous model, the worst prediction is for resource set 7 since 20 resources from the worst cluster (*bordeplage*) are used, which makes the merging process slower.

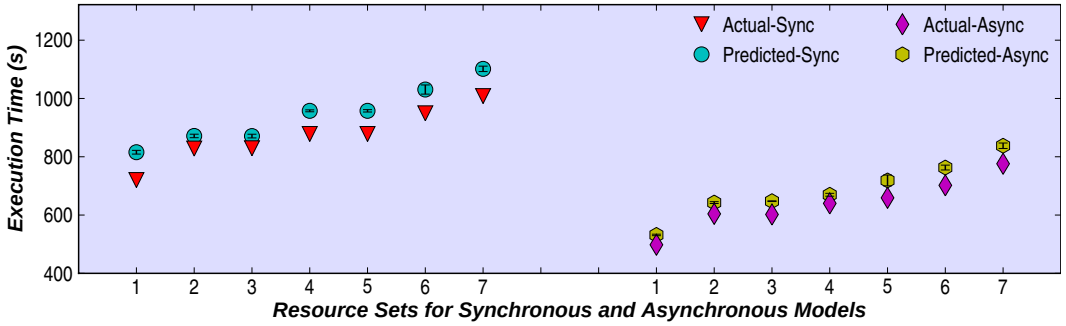


Figure 5.10: Comparison of predicted and actual execution times for synchronous and asynchronous models.

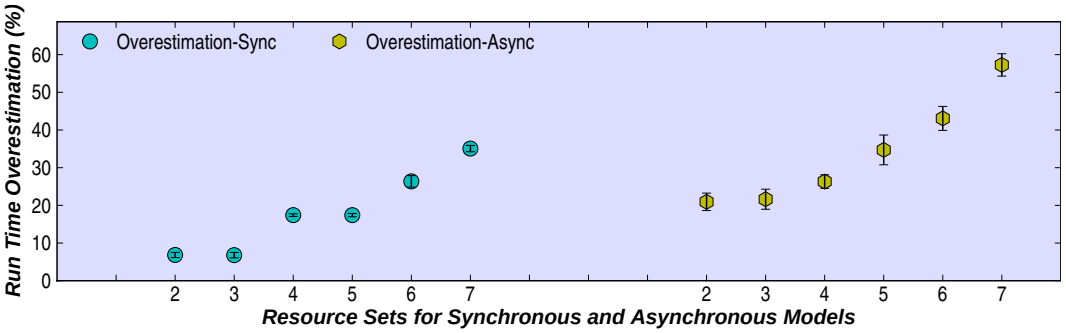
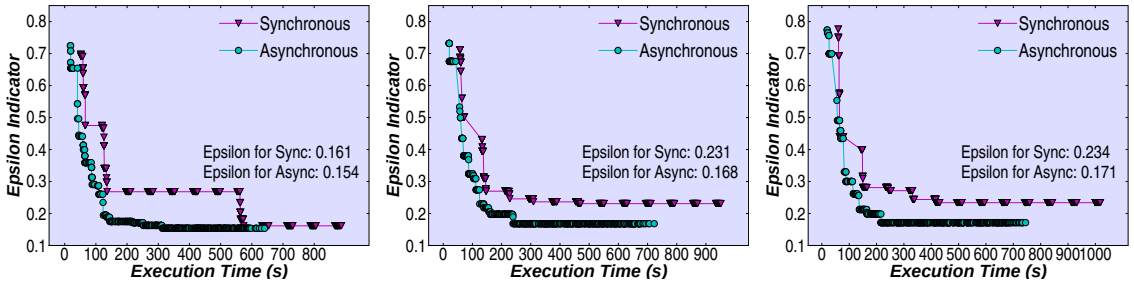


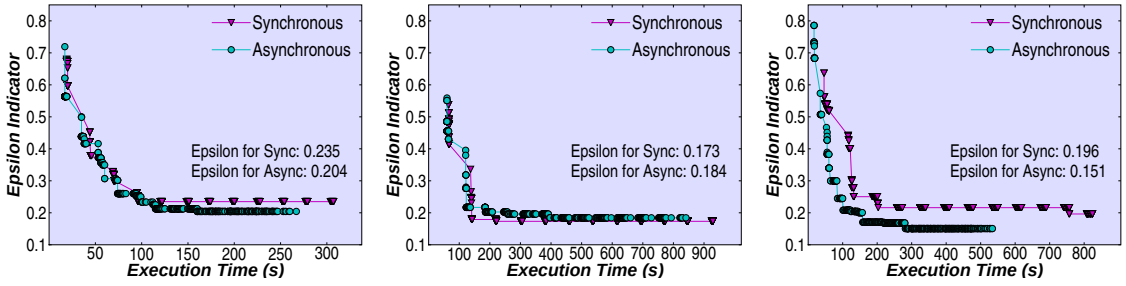
Figure 5.11: Run time overestimations required to avoid application being aborted due to rescheduling from resource set 1 to other sets without co-allocation based on performance predictions.

Importance of predictions to rescheduling. When remapping processes from one resource set to another, the application run time may remain the same, increase or decrease. When it remains the same, schedulers just have to redefine the number of resources in each cluster; which can be performed by the metascheduler or by the system schedulers themselves. This is the case for remapping processes from, for example, resource set 1 to 2 and 2 to 3 or 4 for synchronous and asynchronous model respectively. When the run time increases, the prediction generated by the application-scheduler may avoid the application to be aborted due to underestimations. A rescheduling from a shorter to longer execution time is desired when the application can start earlier than the initial schedule predicted. This is the case for remapping processes from resource set 1 to 7. Figure 5.11 shows that



(a) Resource set 3. (b) Resource set 5. (c) Resource set 6.

Figure 5.12: Epsilon indicator for three resource sets on both communication models.



(a) Only Scale-Free topology. (b) Only Random topology. (c) Four topologies together.

Figure 5.13: Epsilon indicator for resource set 1 showing the importance of mixing topologies for both communication models.

overestimation is required to avoid the application to be killed when rescheduling from resource set 1 to the other resource sets without the use of predictions (the higher the value the more useful is our metascheduler based on predictions). For the synchronous model, 35% of overestimation is required, whereas for the asynchronous model 57%. When rescheduling a request from a longer to shorter run time, predictions assist schedulers to increase the chances of backfilling sub requests [85]. This happens because longer jobs tend not to fill the fragments in the scheduling queues [120]. This is the case when rescheduling processes from resource set 7 to 1 for synchronous and asynchronous model.

Synchronous versus asynchronous models. In order to understand the output produced by the application, we have also compared the quality of the optimisation results between synchronous and asynchronous models. Figure 5.12 shows the Epsilon indicator (the lower the better) for synchronous and asynchronous models under three resource sets. The asynchronous model converges faster and produces better results than the synchronous model. This happens because the asynchronous model is able to mix more results from different EMO processes, which might have different topologies, in relation to the synchronous model. For resource set 3, the synchronous model produces similar result for the Epsilon indicator as the asynchronous model, but the Epsilon values get closer after a considerable execution time. Figure 5.13 illustrates the importance of mixing re-

sults from different topologies. The results show that although Random, which is the most CPU consuming topology, has the greatest impact on the Epsilon indicator, the less CPU consuming Scale-Free topology contributes to the function optimisation. Moreover, even for one-topology executions, asynchronous produces better optimisation results and it converges faster than its synchronous counterpart. Similar results were obtained for the other resource sets. The comparison results between synchronous and asynchronous model showed here corroborate the results presented by Desell et al. [42] with their application in the astronomy field; i.e. asynchronous model has better convergence rates, especially when heterogeneous resources are in place.

5.5 Conclusions

Resource co-allocation ensures that applications access processors from multiple clusters in a coordinated manner. Current co-allocation models mostly depend on users to specify the number of processors and usage time for each cluster, which is particularly difficult due to heterogeneity of the computing environment.

This chapter presented a resource co-allocation model with rescheduling support based on performance predictions for multi-cluster iterative parallel applications. Due to the regular nature of these applications, a simple and effective performance prediction strategy can be used to determine the execution time of application processes. The metascheduler can generate the application performance model without requiring access to the application source code, but by observing the throughput of a process in each resource type using a short partial execution. Predictions also enable automatic rescheduling of parallel applications; in particular they prevent applications from being aborted due to run time underestimations and increase backfilling chances when rescheduled to faster resources.

From the experiments using an iterative benchmark parallel application on Grid'5000, we observed run time predictions with an average error of 7% and prevention of up to 35% and 57% of run time overestimations for synchronous and asynchronous models, respectively. The results are encouraging since automatic co-allocation with rescheduling support is fundamental for multi-cluster iterative parallel applications; in particular because these applications, based on asynchronous communication model, are used to solve problems in large-scale systems.

This chapter meets the third objective presented in Chapter 1, which is investigation of technical difficulties to deploy the co-allocation policies in real environments, in particular for the process remapping operation. Another application model that has been used for large-scale distributed systems is the bag-of-tasks; and therefore we present co-allocation policies for this model in the next two chapters.

Chapter 6

Offer-based Co-Allocation for BoT Applications

Metaschedulers can distribute parts of a Bag-of-Tasks (BoT) application among various resource providers in order to speed up its execution. When providers cannot disclose private information such as their load and computing power, which are usually heterogeneous, the metascheduler needs to make blind scheduling decisions. This chapter describes three policies for composing execution offers to schedule deadline-constrained BoT applications. Offers act as a mechanism in which resource providers advertise their interest in executing an entire BoT or only part of it without revealing their load and total computing power. In addition, evaluation results show the relation between the amount of information resource providers need to expose to the metascheduler and its impact on the scheduling.

6.1 Introduction

The execution of a BoT application on multiple utility computing facilities is an attractive solution to meet user deadlines. This is because more tasks of a single BoT application can execute in parallel and these facilities have to deliver a certain QoS level, otherwise the providers are penalised. A service provider containing a metascheduler is responsible for distributing the tasks among resource providers according to their load and system configuration. However, allocating resources from multiple providers is challenging because these resource providers cannot disclose much information about their local load to the metascheduler. Workload is private information that companies do not disclose easily since it may affect the business strategy of competitors.

Much work has been done on scheduling BoT applications [19, 74, 80]. However,

little effort has been devoted to schedule these applications with deadline requirements [14, 68, 125], in particular considering *limited load information available* from resource providers.

This chapter introduces three policies for composing offers to schedule BoT applications. Offers are a mechanism in which resource providers expose their interest in executing an entire BoT or only part of it without revealing their local load and system capabilities. Whenever providers cannot meet a deadline, they generate offers with another feasible deadline. For the offer generation within resource providers, we leverage the work developed by Islam et al. [60, 61], whereas the concept of combining offers for executing an application on multiple resource providers is inspired by the provisioning model of Singh et al. [109] and the capacity planning of Siddiqui et al. [105]. In addition, this chapter shows the impact of the amount of information resource providers need to expose to the metascheduler and its impact on the scheduling.

6.2 Architecture and Scheduling Goal

A metascheduler receives user requests to schedule BoTs on multiple autonomous resource providers in on-line mode. Users provide the number of tasks in the bag, their estimated required time, and a deadline to execute the entire BoT. Resource providers are responsible for scheduling both local and external jobs using the Earliest Deadline First. The local jobs can be both sequential and message passing parallel applications, whereas the external jobs are BoT applications. The metascheduler has no access to the scheduling queues.

As illustrated in Figure 6.1, the scheduling of a BoT application consists of 5 steps. In step 1, the metascheduler advertises the application requirements to the resource providers. In step 2, the resource providers generate a list of offers that can serve the entire BoT or only part of it. Once the resource providers generate the offers, they send them to the metascheduler (step 3), which composes them according to the user requirements (step 4), and submits the tasks to resource providers (step 5).

Scheduler's Goal. Our goal is to meet users' deadlines, and when not possible, try to schedule jobs as close as possible to these deadlines. The challenge from the metascheduler's point of view is to know how much work to submit to each resource provider such that it meets the BoT user's deadline, whereas from the resource providers' point of view is to know how much work they can admit without violating the deadlines of already accepted requests.

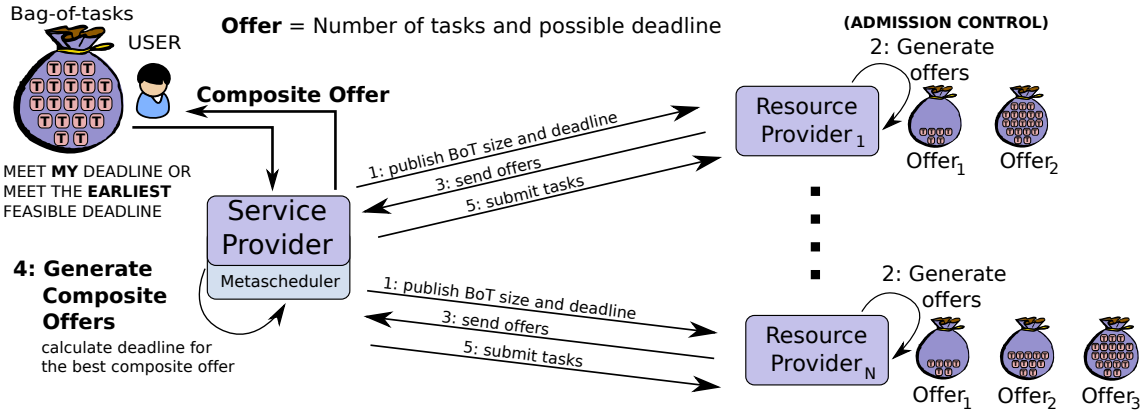


Figure 6.1: Components interaction for scheduling a bag-of-tasks using offers from multiple providers.

6.3 Offer Generation

6.3.1 Deadline-aware Offer Generation

An offer consists of a number of tasks, and the maximum time the resource provider can complete the work. The resource provider could follow different policies to generate a list of offers. For instance, a resource provider could generate offers that are more profitable [59, 96]; provide some slack in case of resource failures or to increase the chances of admitting more jobs in future; or that do not violate the deadline of already scheduled tasks and the new task. In this work, we considered the third approach.

The offer generation uses the BoT information provided by the metascheduler, which includes estimated execution time of each task, number of tasks, and deadline. Different from scheduling based on FIFO with conservative backfilling for example, where it is possible to identify time slots by simply calculating the start and completion time of the scheduled tasks, an Earliest Deadline First based queue cannot follow such an approach. The reason is that the offer generation involves the rescheduling of the already accepted jobs, and hence the free time slots depend on the new submitted job.

In order to generate the list of offers Φ, the resource provider:

1. Defines a set of possible number of tasks Δ it is willing to accept. It creates this list by calculating a percentage of the total number of tasks in the BoT application, e.g. $\Delta \leftarrow \{BoT^s, 0.75 * BoT^s, 0.50 * BoT^s, 0.25 * BoT^s, 0.10 * BoT^s\}$, where s is the BoT size, i.e. total number of tasks.

2. A procedure *genOffer* generates an offer for each BoT size. To generate an offer, the resource provider creates a temporary job j_k that has the BoT specifications,

which include deadline and estimated execution time, but with the different number of tasks, defined in Δ. The algorithm used is the Earliest Deadline First.

3. The *genOffer* procedure returns the completion time of that offer. This time can be the deadline provided by the user (in the case of a successful schedule), or a completion time that is longer than the deadline (when the scheduler cannot meet the user deadline).

4. The algorithm compares the current offer with the previous offer in Φ. If the completion time is the same, the previous offer simply has its number of tasks increased. If the completion time is longer, the resource provider includes the current offer in the list Φ.

FEEDBACK. In order to provide users with feedback when it is not possible to meet their deadlines, we use the approach proposed by Islam et al. [61]. The idea is to use a binary search that has as its first point the deadline defined by the user and its last point as the longest feasible deadline, i.e. the one when the job is placed in the last position of the scheduling queue.

6.3.2 Load-aware Offer Generation

We use a policy based on free time slots presented by Singh et al. [109] (**FreeTimeSlots** policy) as base for comparison in our evaluations. In this policy, the resource provider uses the current schedule containing running and pending jobs. The resource provider generates windows for each processor that represent their time availability, also known as *time slots*. Different from Singh et al. [109], we analyse and provide the time slots for the entire scheduling queue, not only part of it.

6.4 Offer Composition

The metascheduler is responsible for composing offers from resource providers and giving the user a single offer with a feasible deadline that can be met. The offer composition determines how much work the metascheduler should send to each resource provider; and when these tasks should receive the resources. The goal of the metascheduler is to meet their users' deadlines, or get as close as possible to the deadlines.

Once the metascheduler receives the offers from resource providers, it analyses whether it is possible to meet user's deadline. We have developed one policy for when the user's deadline cannot be met (*OffersNoLB*), and two policies for when it is possible to meet the user's deadline (*OffersWithPLB* and *OffersWithDBLP*). These last two policies try to

balance the load distributed among the resource providers according to the information available to the metascheduler.

6.4.1 When User Deadline Cannot Be Met

For the **OfferNoLB** policy, after collecting the offers from resource providers, the metascheduler uses the following algorithm to compose offers:

1. Create a list with all the offers;
2. Sort the list such that all offers that meet the user's deadline come before those that do not meet. For those that meet, the offers are sorted in the decreasing order of number of tasks. For those that do not meet, the offers are sorted in the ascending order of completion time;
3. Remove all offers after the first offer that is able to execute the entire BoT;
4. Create a list L that is dynamically updated with possible composite offers. The creation of L is based on the order of the offers. For each offer analysed, the metascheduler updates the number of remaining requested tasks and the last completion time of each list that uses that offer. Note that what makes this algorithm simple is the list pre-processing, i.e. sorting and filtering;
5. Return the first composite offer in L that provides the earliest possible deadline.

Table 6.1 illustrates an example on how the metascheduler composes offers. The example considers a list of offers $\Phi = \{(s_1, d_1)_{rid_1}, ..., (s_n, d_n)_{rid_n}\}$ where rid is the resource provider id, s is the BoT size, and d is the offer's deadline, and a BoT with deadline = 40 time units and number of tasks = 512. As we can observe, the choice of the offers is not greedy. From the example, the metascheduler does not use the offer *(256, 40)*$_1$, which is the best offer, because the remaining resource providers would end up completing the BoT by 200 time units rather than 100 time units, which is the next best offer from resource provider 3 that can accept enough tasks ((512, 200)$_3$).

6.4.2 Balancing Load When Possible to Meet User Deadline

When it is possible to meet the user's deadline, the metascheduler tries to balance the number of tasks to be submitted to each resource provider. This balance allows jobs local to resource providers to meet more deadlines. The policy for load balancing depends on the amount of information the metascheduler has about the resource providers.

We have developed two policies for load balancing:

Table 6.1: Example of offer composition with the *OfferNoLB* policy for a BoT with number of tasks = 512 and deadline = 40 time units.

Operation	**Offers (size, deadline)$_{\text{provider}}$**
Original Offers	(256, **40**)$_1$, (512, 100)$_1$, (128, **40**)$_2$, (512, 200)$_2$, (64, **40**)$_3$, (512, 200)$_3$
Sorted Offers	(256, **40**)$_1$, (128, **40**)$_2$, (64, **40**)$_3$, (512, 100)$_1$, (512, 200)$_2$, (512, 200)$_3$
Filtered Offers	(256, **40**)$_1$, (128, **40**)$_2$, (64, **40**)$_3$, (512, 100)$_1$
Composite Offers (List L)	{(128, **40**)$_2$, (64, **40**)$_3$, (320, 100)$_1$} OR
	{(256, **40**)$_1$, (128, **40**)$_2$, (64, **40**)$_3$ (not enough tasks)}
Selected Composite Offer	(128, **40**)$_2$, (64, **40**)$_3$, (320, 100)$_1$

- **OffersWithPLB** (Proportional Load Balancing) balances the load according to the size of the offers;

- **OffersWithDPLB** (Double Proportional Load Balancing) balances the load according to the size of the offers *and* the total computing power of resource providers.

These policies work only with offers that meet user's deadlines; all the other offers are discarded. Therefore, each resource provider has only one offer that meets the deadline.

The *OffersWithPLB* policy uses the offer size in order to balance the number of tasks submitted to each resource provider. The proportional parameter P is calculated per offer as follows: $P \leftarrow$ *OfferSize* / *totalOfferedTasks*, where *OfferSize* is the number of tasks in an offer, and *totalOfferedTasks* is the total number of tasks from all offers. For each offer, the number of tasks is multiplied by P. As the metascheduler does not know the load and the total computing power of resource providers, the offer size serves as an indicator of how much work a resource provider should receive in relation to the others.

For the *OffersWithDPLB* policy (Algorithm 6), the parameter P is calculated by the group of offers with the same number of tasks (Line 3). The additional parameter *Double Proportional (DP)* is used to distributed the load to a given group of resource providers that contains offers with the same number of tasks (offer size) according to their capabilities (e.g. total number of resources a provider hosts) (Line 12). It is not always possible to distributed the tasks of a given offer size exactly proportionally to the resource capabilities. For this reason, we sort the offers of the same size in a decreasing order of providers' total computing power (Line 1) and we adjust the number of tasks to the resource provider when necessary (Lines 14-15).

Algorithm 6: Pseudo-code for composing offers using the *OffersWithDPLB* policy.

```
1  Sort offers by decreasing order of their size (offers of same size are sorted by decreasing
   order of resource provider's total computing power)
2  for each offer size in offers list do
3      P ← totalTasksOfferedThisSize / totalOfferedTasks
4      remainingNTasksSameSize ← P * BoTSize
5      totalCPower ← total RPs' computing power of offers of this size
6      currentSize ← offer size
7      for all offers with size = currentSize do
8          if last offer from this group then
9              nTasks ← remainingNTasksSameSize
10         else
11             nTasks ← P * BoTSize
12             DP ← offer.RPCPower / totalCPower
13             nTasks ← DP * nTasks
14             if nTasks > offer.nTasks then
15                 nTasks ← offer.nTasks
16         offer.setNTasks(nTasks)
17         remainingNTasksSameSize.decrement(offer.nTasks)
18         compositeoffer.add(offer)
```

6.5 Evaluation

We have evaluated the scheduling policies by means of simulations to observe their effects in a long-term usage. We have used our event-driven simulator PaJFit and real traces from supercomputers available at the Parallel Workloads Archive and extended them according to our needs.

We have evaluated the following scheduling policies:

- **FreeTimeSlots:** scheduling based on free time slots, i.e. the metascheduler has a detailed access to the load available (Section 6.3.2);
- **OffersWithPLB:** scheduling based on offers. The scheduler composes offers based on their sizes (Section 6.4.2);
- **OffersWithDPLB:** the metascheduler considers offer sizes and the total computing power of resource providers (Section 6.4.2);
- **OffersWithDPLBV2:** an extension of *OffersWithDPLB* in which the metascheduler has access to the resource providers' load to be processed (not the free time slots). This information is used in the same way to calculate the parameter DP described in Section 6.4.2;

For the offer-based policies, i.e. *OffersWithPLB*, *OffersWithDPLB*, and *OffersWithDPLBV2*, when no offer can meet the user deadline, the metascheduler follows the **OffersWithNoLB** policy described in Section 6.4.1.

6.5.1 Experimental Configuration

TRACES. We have modeled an environment composed of five clusters with their own schedulers and loads, and one metascheduler that receives external (BoT) jobs that can be executed in either a single or multiple clusters. For the local jobs, we have 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, the 128-node IBM SP2 from The San Diego Supercomputer Center (SDSC SP2 v3.1), and the 70 dual 3GHz Pentium-IV Xeons from LPC Clermont-Ferrand in France (LPC-EGEE v1.2). We have used two parts of the trace HPC2N, from different years, to simulate two clusters. For the external jobs, we have used the trace of a bigger machine from 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 64 processors (SDSC BLUE v3.1). We have simulated 60 days of these traces.

Table 6.2: Summary of workloads used to perform the experiments.

Location	Trace	Procs	Jobs	Load	Job Req Procs	Job Req Time
Cluster 1	CTC SP2v2.1	430	3,478	49%	$1 \leq$ procs ≤ 306	1h $\leq$ time $\leq$ 18h
Cluster 2	HPC2N v1.1	240	959	56%	$1 \leq$ procs ≤ 128	1h $\leq$ time $\leq$ 120h
Cluster 3	HPC2N v1.1	240	4,913	48%	$1 \leq$ procs ≤ 128	1h $\leq$ time $\leq$ 120h
Cluster 4	SDSC SP2 v3.1	128	1,088	54%	$1 \leq$ procs ≤ 115	1h $\leq$ time $\leq$ 18h
Cluster 5	LPC-EGEE v1.2	140	6,574	52%	$1 \leq$ procs ≤ 1	2h $\leq$ time $\leq$ 72h
External	SDSC BLUE v3.1	1178	969	54%	$64 \leq$ procs ≤ 632	1h $\leq$ time $\leq$ 36h

LOAD. Regarding the load used in the resource providers, approximately 50% comes from the multi-site BoTs (*external load*), which could be executed in any cluster or multiple clusters, and approximately 50% comes from users submitting parallel or sequential jobs directly to a particular cluster (*local load*). The *global load* is therefore the external load plus the local load submitted to the clusters. We have chosen the same load for local and external loads in order to be able to compare the impact of the scheduling policies on local and external jobs in a fair manner. We varied the loads using a strategy similar to that described by Shmueli and Feitelson to evaluate their backfilling strategy [104], in which they modify the jobs' arrival time. However, we fixed the simulation time interval and modified the number of jobs in the traces. Table 6.2 summarises the workload

characteristics. More details on the workloads can be found at the Parallel Workloads Archive.

DEADLINES. To the best of our knowledge, there are no traces available with deadlines. Therefore, we have incorporated deadlines in the existing traces using the following function: T_j^s+T_j^r+k, where T_j^s is the job submission time, T_j^r is the job estimated run time, and k is a parameter that assumes three values according to two **Deadline Schemata**. For *Deadline Schema 1*, k assumes the values 18 hours, 36 hours, and 10 days, and for *Deadline Schema 2*, k assumes the values 12 hours, 1 day, and 1 week. Therefore, Deadline Schema 2 has more jobs with tighter deadlines than Deadline Schema 1. We have used a uniform distribution for the values of k for all jobs in each workload. Note that k is not a function of job size. Modeling k independently of job size allowed the environment to have both small and big jobs with relaxed and tight deadlines. We have generated 30 workloads for each original trace varying the seed for the deadlines. By having 60 days of simulated time, 30 workloads with different deadlines, and 2 deadline schemas, we believe that we have been able to evaluate the policies under various conditions.

METRICS. We have assessed five metrics:

1. *Jobs Delayed:* number of jobs that are not able to meet their deadlines;

2. *Work Delayed:* amount of work (processors x execution time) of the jobs that are not able to meet their deadlines;

3. *Total Weighted Delay:* weighted difference between jobs' deadlines and their new deadline given by the system;

$$TWDelay = \sum_{D_j^N > D_j} R_j * \left(\left(\frac{D_j^N - T_j^s}{D_j - T_j^s} \right) - 1 \right) * 100 \tag{6.1}$$

   Where D_j is the job deadline, D_j^N is the new job deadline provided by the system, R_j is the number of tasks of the job, and T_j^s is the job submitted time. We used the value R_j as the weight for the percentage difference between the time of the original and new deadline.

4. *Clusters per BoT:* number of clusters used by BoTs;

5. *System Utilisation:* global system utilisation.

For utility computing environments, the first two metrics represent the loss in revenue due to possible rejections, whereas the third metric could represent penalties for not meet-

ing user's demand. The last two metrics allow us to verify how jobs are spread across the clusters and the impact of the policies on the system utilisation.

GOAL. Assess the impact of the information available to the metascheduler for scheduling deadline-constrained jobs.

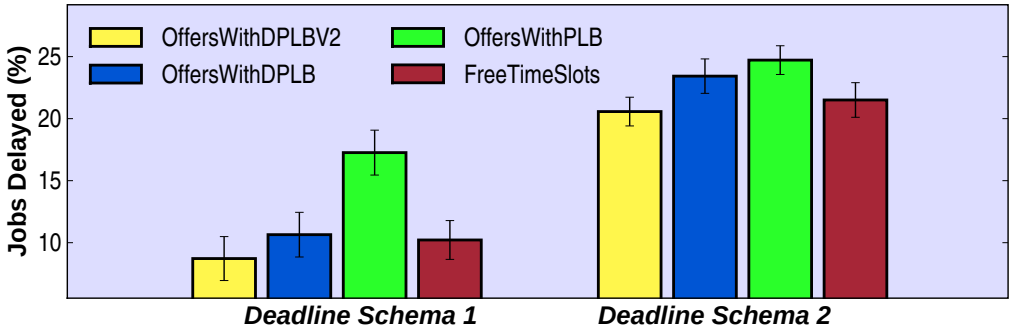
6.5.2 Results and Analysis

The graphs we show in this section contain the averages of 30 simulation runs, each with different workloads, along with their standard deviations. We show the results for the external load, local load and both together since external and local load have different characteristics. We first analyse the jobs that are not able to meet their deadlines when submitted to the system. Figures 6.2 and 6.3 represent number of jobs and their respective amount of work (time x number of tasks) delayed in relation to the total number of jobs in the system and total amount of work respectively.

Deadline tightness. We observe that the difference in the results among the offer-based policies increases when jobs have more relaxed deadlines (Schema 1 has more relaxed deadline jobs than Schema 2). That is because as jobs have more relaxed deadlines in Schema 1, resource providers can reschedule more jobs in order to generate better offers. Therefore, the metascheduler has more options to schedule BoT applications. When it is not possible to meet a user deadline, which is the more frequent in Schema 2, the load balancing policies cannot be used. The metascheduler has to use the *OffersWithNoLB* policy (Section 6.4.1) for most of the jobs.

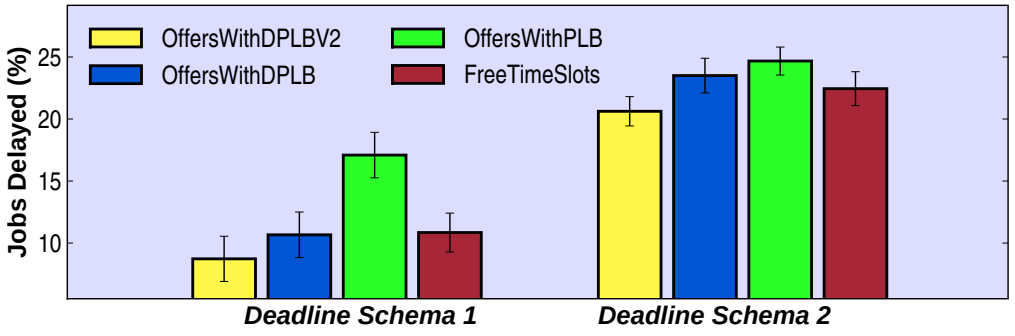
Information access. In relation to the differences between the *FreeTimeSlots* policy and offer-based policies, we observe that the former policy handles local jobs better or similar to offer-based policies. That is because the metascheduler has more detailed load information using the *FreeTimeSlots* policy, and hence it can better distribute the load among resource providers. As local jobs do not have the option to choose the resource providers, they enjoy more benefits using this policy. The *OffersWithDPLBV2* policy generates similar results as *FreeTimeSlots* for local jobs. This happens because in *OffersWithDPLBV2*, the metascheduler has rough access to the local loads, which is enough to balance the load and gives local jobs equal opportunity. Finally, we observe that for these two metrics, having access to the providers' total computing power is enough to provide as good results as the *FreeTimeSlots* policy, in which the metascheduler has a detailed access to the resource providers' loads, i.e. the free time slots. If rough load information is also available (*OffersWithDPLBV2*), it is possible to get even better results.

Job delays. Figure 6.4 illustrates total weighted delay for not meeting the deadlines of



(a) Local Load.

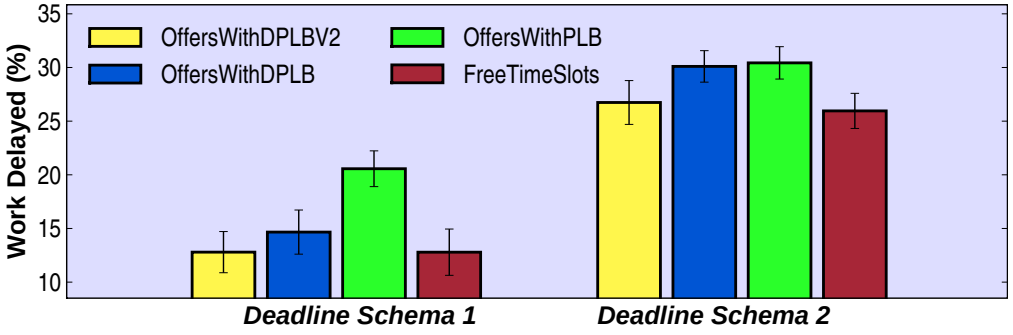
(b) External Load.



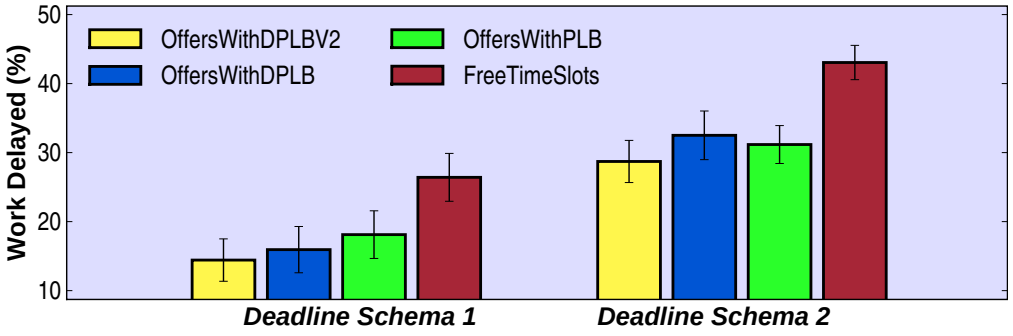
(c) Global Load.

Figure 6.2: Number of jobs delayed for local, external, and global load.

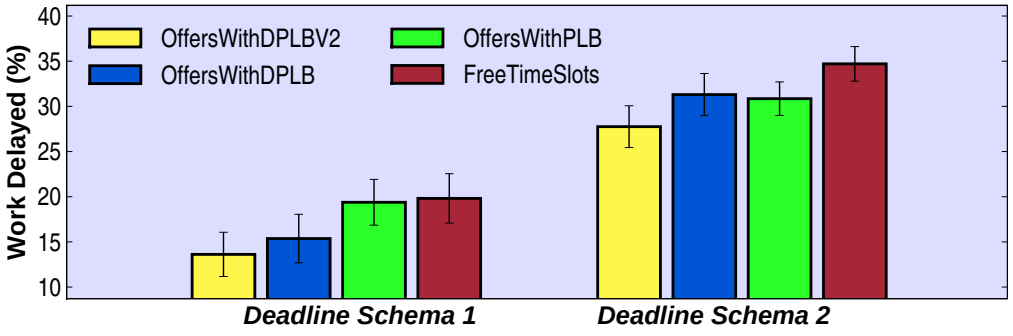
local, external, and global load. This metric is interesting because it shows the difference between what users asked and what the system provides. The behavior of this metric is similar to the previous metrics, except that the delayed external jobs suffer much more in the *FreeTimeSlots* policy. In this policy, the providers disclose their free time slots to the metascheduler, which has no knowledge of the deadlines of the already accepted jobs. Therefore, the metascheduler makes blind decisions in terms of deadlines, which have a considerable impact on external jobs. Thus, even though resource providers disclose detailed information of their local load to the metascheduler using the *FreeTimeSlots*



(a) Local Load.



(b) External Load.



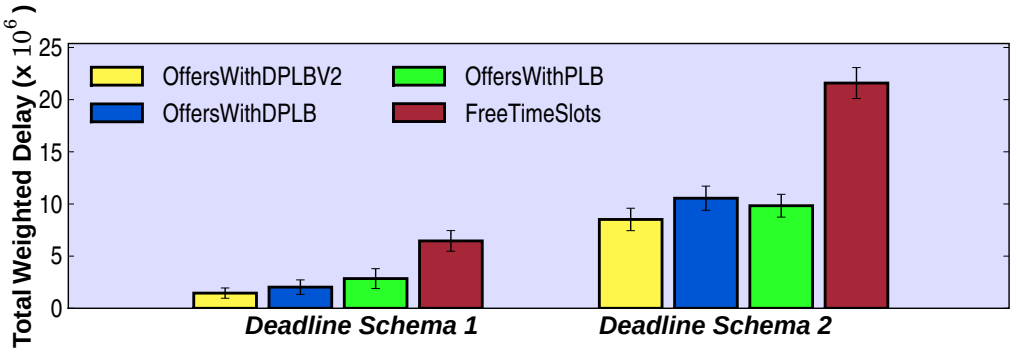
(c) Global Load.

Figure 6.3: Amount of work delayed for local, external, and global load.

policy, such a policy produces much worse results than the simplest offer-based policy, i.e. *OffersWithPLB*. In this offer-based policy, the metascheduler uses only the offers, without knowing the resource providers' total computing power and load. This reveals that indeed, the offer sizes are a good indicator to balance load among resource providers without accessing their private information.

Load and system utilisation. Another important factor is the *Number of Clusters* used by the BoT applications. Figure 6.5 shows the number of clusters used by BoT applications for each policy. We observe that the tighter the deadlines, the fewer options the

(a) Local Load.



(b) External Load.

(c) Global Load.

Figure 6.4: Total Weighted Delay for local, external, and global load.

metascheduler has to distribute the tasks of BoT applications among resource providers. That is because there is more unbalance in the load when jobs have tighter deadlines, and hence the metascheduler tends to use fewer options in offer-based policies, and fewer time slots in the *FreeTimeSlots* policy. In addition, the offer-based policies with double proportional load balancing tend to distribute the load better than the other two policies. Therefore, they allow more scheduling options for the next jobs arriving into the system. Regarding the *System Utilisation* (Figure 6.6), we observe that the difference between the policies is minimal, i.e. less than 1 percent. *OffersWithDPLB* has a minimal decrease in

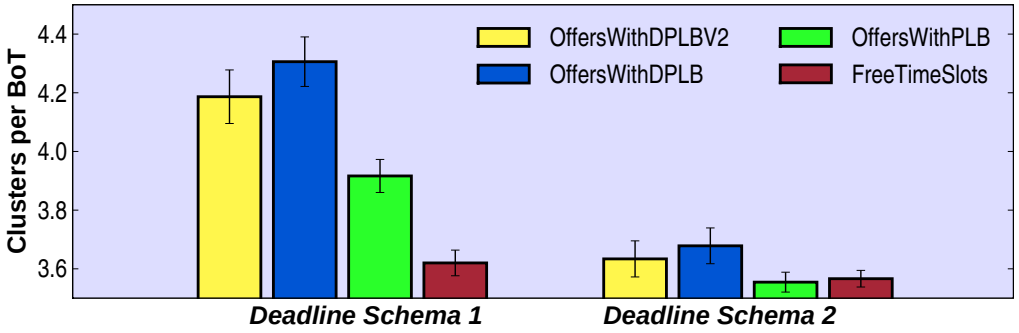


Figure 6.5: Clusters per BoT.

Figure 6.6: Global system utilisation.

relation to the other policies because it considers the total computing power of providers without their actual loads. Therefore, a provider that has less computing power than the others may receive fewer tasks even if the other big providers have higher load. This situation happens only when all offers that meet the deadline have the same size.

6.6 Conclusions

This chapter described three policies for composing resource offers from multiple providers to schedule deadline-constrained BoT applications. These offers express the interest of resource providers in executing an entire BoT or only part of it without revealing their local load and total system capabilities. When the metascheduler receives enough offers to meet user deadlines, it decides how to balance the tasks among the resource providers according to the information it has access, such as resource providers' total computing power and their local loads. Whenever providers cannot meet a deadline, they generate offers with another feasible deadline. The metascheduler is then responsible for composing the offers and providing users with a feedback containing the new deadline.

From our experiments, we observed that by using the free time slots of resource

providers, BoT applications cannot access resources in short term even when local jobs could be rescheduled without violating their deadlines. The only benefit of publishing the free time slots to the metascheduler is that it can balance the load among resource providers, which makes more local jobs meet deadlines. However, when using offer-based policies, more BoTs can meet deadlines and the delays between the user deadline and the new deadline assigned by the system is much lower (in some cases 50% lower) in comparison to the policy that uses free time slots (*FreeTimeSlots*).

We also observed that the simplest offer-based policy (*OffersWithPLB*) produces schedules that delay fewer jobs in comparison to the *FreeTimeSlots* policy. However, *OffersWithPLB* rejects more local jobs than *FreeTimeSlots*. This happens because *OffersWithPLB* cannot balance the load among resource providers. If resource providers also publish the total computing power (the *OffersWithDPLB* policy), the metascheduler can balance the load and have similar acceptance rates as the *FreeTimeSlots* policy for local jobs. If the resource providers can make their load available (*OffersWithDPLBV2*), the metascheduler can reduce even more the number of jobs delayed; however the benefit is not significant. Therefore, our main conclusions are: (i) offer-based scheduling produces less delay for jobs that cannot meet deadlines in comparison to scheduling based on load availability (i.e. free time slots); thus it is possible to keep providers' load private when scheduling multi-site BoTs; and (ii) if providers publish their total computing power they can have more local jobs meeting deadlines.

This chapter meets part of the second objective of the thesis, which is design, implementation, and evaluation of co-allocation policies. It considered co-allocation as offer-composition for BoT applications with deadline constraints. This chapter is a building block for the next chapter, which deals with inaccurate run time estimates, rescheduling, and implementation issues, thus meeting the three objectives of this thesis for BoT applications.

Chapter 7

Adaptive Co-Allocation for BoT Applications

The expected completion time of the user applications is calculated based on the run time estimates of all applications running and waiting for resources. However, due to inaccurate run time estimates, initial schedules are not those that provide users with the shortest completion time. This chapter proposes a coordinated rescheduling algorithm and evaluates the impact of this algorithm and system-generated predictions for bag-of-tasks in multi-cluster environments. The coordinated rescheduling defines which tasks can have start time updated based on the expected completion time of the entire BoT application, whereas system-generated predictions assist metaschedulers to make scheduling decisions with more accurate information. We performed experiments using simulations and an actual distributed platform, Grid'5000, considering three main variables: time to generate run times, accuracy of run time predictions, and time users are willing to wait to schedule their applications.

7.1 Introduction

Metaschedulers can distribute parts of a BoT application among various resource providers in order to speed up its execution. The expected completion time of the user application is then calculated based on the run time estimates of all applications running and waiting for resources. A common practice is to overestimate execution times in order to avoid user applications to be aborted [72, 73]. Therefore, initial completion time promises are usually not accurate. In addition, when a BoT application is executed across multiple clusters, inaccurate estimates increase the time difference between the completion of its first and last task, which increases average user response time in the entire system. This

time difference, which we call *stretch factor*, increases mainly because rescheduling is performed independently by each provider.

System generated predictions can reduce inaccurate run time estimates and prevent users from having to specify these values. Several techniques have been proposed to predict application run times and queue wait times. One common approach is to analyse scheduling traces; i.e. historical data [91, 111, 119]. Techniques based on trace analyses have the benefit of being application independent, but may have limitations when workloads are highly heterogeneous. Application profiling has also been vastly studied to predict execution times [63, 101, 103, 130]. Application profiling can generate run time predictions for multiple environments, but usually requires application source code access.

This chapter proposes a coordinated rescheduling strategy for BoT applications running across multiple resource providers. Rather than providers performing independent rescheduling of tasks of a BoT application, the metascheduler keeps track of the expected completion time of the entire BoT. This strategy minimises the stretch factor and reduces user response time. We also show that on-line system generated predictions, even though require time to be obtained, can reduce user response time when compared to user estimations. Moreover, with more accurate predictions, providers can offer tighter expected completion times, thus increasing system utilisation by attracting more users. We performed experiments using simulations and an actual distributed platform, Grid'5000, on homogeneous and heterogeneous resources. We also provide an example of system-generated predictions using POV-Ray, which is a ray-tracer tool to generate three dimensional images to produce animations.

7.2 Scheduling Architecture

Similar to the architecture presented in Chapter 6, a metascheduler receives user requests to schedule BoTs on multiple autonomous resource providers in on-line mode. The two main differences are that users provide the number of required resources along with either a run time estimation or an application profiler. In addition, a rescheduling component is placed into the metascheduler and resource providers.

As illustrated in Figure 7.1, the scheduling of a BoT application consists of 6 steps. In step 1, the metascheduler advertises the application profiler or user estimations to the resource providers. In step 2, the resource providers execute the profiler (if available) and generate a list of offers that can serve the entire BoT or only part of it. An offer consists of a number of tasks, and their expected completion time. In this work, resource providers generate offers aimed not to violate the expected completion time of already

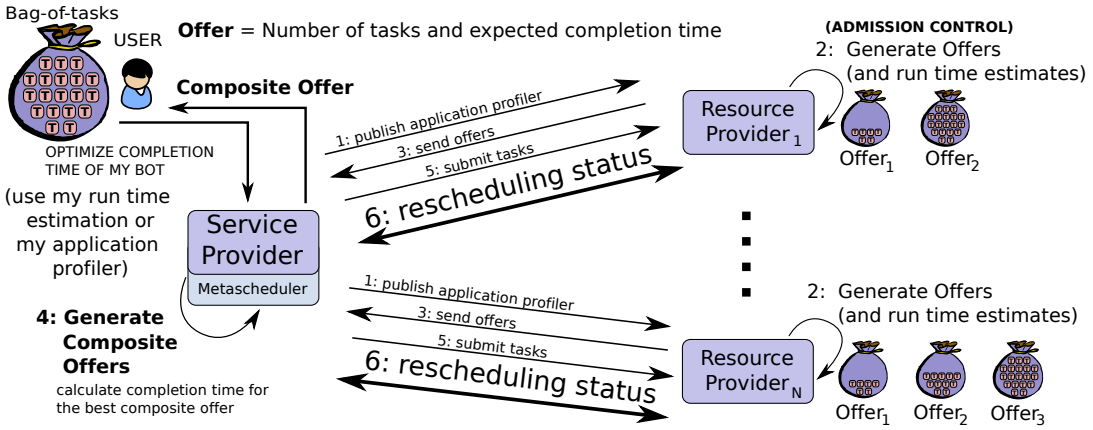


Figure 7.1: Components' interaction to schedule a Bag-of-Tasks using coordinated rescheduling and system generated predictions.

scheduled tasks and to consider the tasks' run time estimation errors. Once the resource providers generate the offers, they send them to the metascheduler (step 3), which composes them according to the user requirements (step 4), and submits the tasks to resource providers (step 5). After the tasks of a BoT are scheduled, resource providers contact the metascheduler for rescheduling purposes (step 6).

Due to system heterogeneity and the different loads in each resource provider, offers arrive at different times to the metascheduler. Once the metascheduler receives all offers, some of them may not be valid any more since other users submitted applications to the providers. To overcome this problem, we use an approach similar to the one developed by Haji et al. [55], who introduced a Three-Phase commit protocol for SNAP-based brokers. We used *probes*, which are signals sent from the providers to the metaschedulers interested in the same resources to be aware of resource status' changes.

Figure 7.2 represents a simplified version of the class diagram for the metascheduler. There are four main components: the metascheduler main class, scheduler, rescheduler, and a list of scheduled jobs. The main responsibilities of the metascheduler class are to submit jobs to resource providers, and keep a table with the expected completion time of the BoTs. The complexity of implementing the scheduler lies on composing the offers. The rescheduler is responsible for updating BoT completion times in the scheduling queues of resource providers. Figure 7.3 illustrates the sequence diagram for the initial co-allocation of a BoT application.

The **schedulers' goal** is to provide users with expected completion time and reduce such a time as much as possible during rescheduling phases.

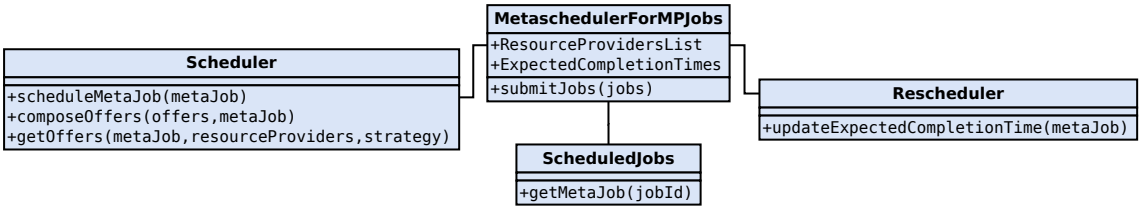


Figure 7.2: Class diagram for the metascheduler of BoT applications.

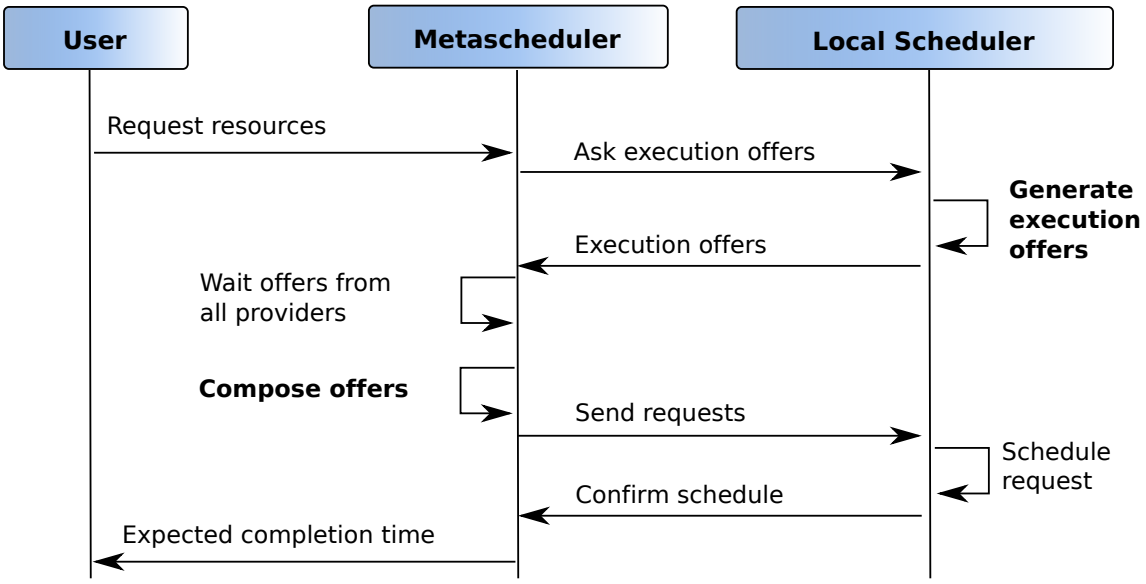


Figure 7.3: Sequence diagram for the initial co-allocation.

7.3 Coordinated Rescheduling

Once the metascheduler provides the user with an expected completion time of his/her BoT application, the application tasks start execution immediately or after resources become available. For the second case, it means tasks are placed in a waiting queue and can be rescheduled to start before expected when tasks from other applications have inaccurate run time estimates.

Figure 7.4 illustrates the difference between the traditional rescheduling strategy, which considers all tasks independently, and the coordinated rescheduling, which considers the tasks of a BoT as being part of a single application. For the coordinated rescheduling, the local scheduler reschedules BoT applications considering their global completion time, rather than their local completion time information.

Whenever a job completes before the expected time, local schedulers execute Algorithm 7 to reschedule the waiting queue. The first step is to sort the jobs in the waiting queue by increasing order of their expected completion times. Jobs from the same BoT are sorted by increasing order of expected start time individually. Later, jobs are resched-

uled one by one. For each job j_i being part of a BoT, the scheduler verifies whether j_i holds the expected completion time of the entire BoT. Both BoT jobs and other type of jobs are then rescheduled using FIFO with conservative backfilling. If a BoT job holds the expected completion time of the entire job and receives a new completion time due to rescheduling, the algorithm keeps this job in a structure call $newCompletionTimes$, which contains the job id and the new completion time. The algorithm is executed again but this time sorting the jobs by their start time. This is done to avoid any fragments in the queue that are not possible to be filled using the sorting by completion time. After all jobs are rescheduled, the local scheduler sends the $newCompletionTimes$ structure to the metaschedulers holding the respective BoTs.

From the metascheduler side, each time it receives the $newCompletionTimes$ structure, it verifies whether the new completion times are local or global. If they are global, the metascheduler sends to the local schedulers holding BoT tasks this new information.

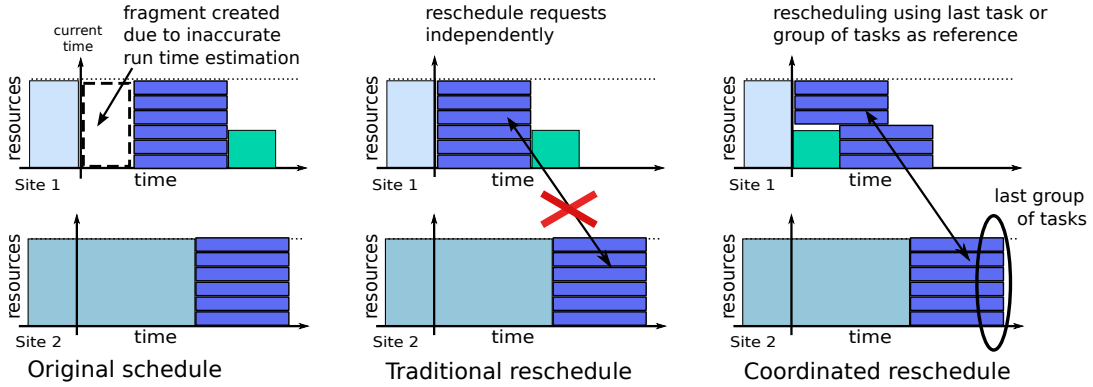


Figure 7.4: Example of schedule using coordinated rescheduling.

7.4 On-line System Generated Predictions

Metaschedulers can better distributed tasks among multiple providers when tasks have precise run time estimates. This has a direct impact on user response time and system utilisation. For example, resource providers can publish offers with tighter response times, thus increasing system utilisation by attracting more users. One approach to generate run time estimations is through the analysis from previous execution of applications with similar characteristics. The main limitation of this approach is that if applications are submitted by several users with different requirements, it is difficult to find patterns to estimate execution times. Another approach to generate run time estimates is through application profiling via execution sampling.

Usually, tasks in a BoT application have similar nature, and therefore by execution

Algorithm 7: Pseudo-code for rescheduling jobs, which is executed on the local schedulers when a job completes before the expected time.

```
1  Sort jobs by expected completion time. BoT tasks are sorted by expected
   completion of the entire BoT. Tasks from the same BoT are sorted by expected start
   time
2  for ∀j_i ∈ waiting queue do
3      isLastTask ← false
4      previousCompletionTime ← completion time of j_i
5      if j_i is part of a BoT then
6          isLastTask ← holds the last expected completion time
7      Reschedule j_i using FIFO with conservative backfilling
8      if previousCompletionTime ≠ newCompletionTime and
       isLastTask = true then
9          add task to possible new completion time list
10 Repeat algorithm execution by sorting jobs in an ascending order of start time.
   Send new completion times to the metaschedulers
```

a few tasks it is possible to estimate the overall application execution time. In addition, depending on the application, it is possible to reduce the problem size in order to speed up the prediction phase. For example, image processing applications can have the problem size reduced by modifying image resolutions. The following sections describe an example of run time generator and discuss when and how to execute the generator.

7.4.1 Example of Run Time Estimator for POV-Ray

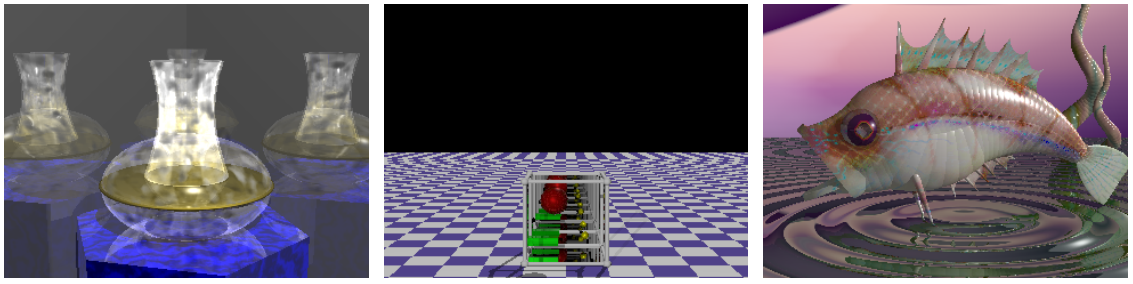
This section describes a run time estimator for POV-Ray[1], a ray-tracer tool that generates three dimensional images for creating animations. Ray tracing is a CPU consuming process that depends on several factors such as image size, rendering quality, and objects and materials in an image. We consider users with animation specifications to generate sets of frames. The execution times are unknown since they depend on several factors of the frames and on the properties of the machine processing the frames. Therefore, application profiling can give an insight on the time cost to generate all the frames, which has an impact on how to schedule the application. We use POV-Ray to create three short animations containing 200 frames each, with a resolution of 2048x1536 pixels (Quad eXtended Graphics Array).

In order to create the animations, we used three examples of images that come with the POV-Ray package, namely Sky Vase, Box, and Fish. Sky Vase consists of a Vase with a sky texture on a table with two mirrors next to it (we replaced the texture *BrightBlueSky*

[1]POV-Ray - The Persistence of Vision Raytracer: http://www.povray.org

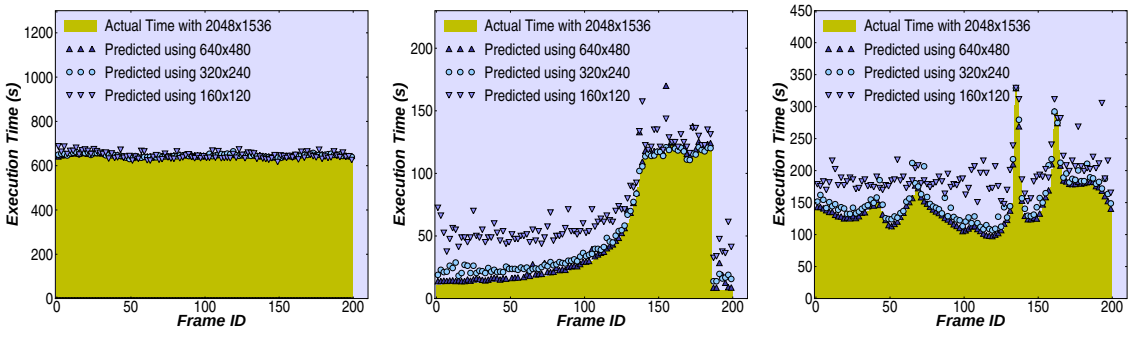
to *Clouds* in order to increase the workload). To generate the animation for Sky Vase, we rotated the vase 360 degrees. Box consists of a chess floor with a box containing a few objects with mirrors inside. We included a camera that gets closer to the box and cross it on the other side. Fish consists of a fish over water that rotates 360 degrees. Different from Sky Vase, Fish has a more heterogeneous animation due to the fish's shape (a vase shape is symmetric vertically).

The animations have different execution time's behaviour. Sky Vase has a steady execution time since the vase is the only object the rotates and its texture has similar work to be processed on each frame. For the Box animation, at the beginning of the animation the box is still far, and hence small, consuming little processing time. However, as the camera approaches the box, more work has to be processed, getting to its maximum when the camera is inside the box. After the camera crosses the box, only the floor has to be rendered. The Fish animation has a very heterogeneous execution time due to the fish's shape, which impacts on the amount of work that needs to be processed when rendering the reflex of the fish on the water. Figure 7.5 illustrates an example of image for each of the three animations.



(a) Sky Vase. (b) Box. (c) Fish.

Figure 7.5: Example of images for each of the three animations.



(a) Sky Vase. (b) Box. (c) Fish.

Figure 7.6: Predicted execution time using most CPU consuming frame as base for estimations.

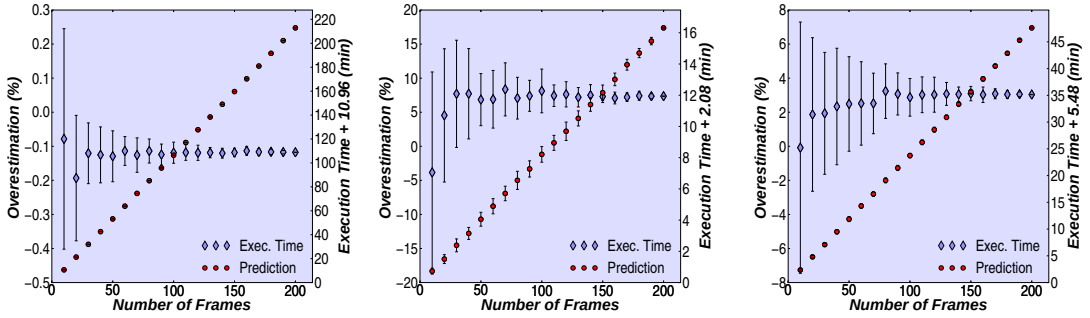
Table 7.1: Time to generate execution time estimates and their accuracy using three rescaled resolutions. The times include the processing of the base frame with the original resolution. Note that the total execution time for animations of Sky Vase, Box, and Fish are 36h, 2.5h, and 8h respectively.

Animation	Resolution	Exec. Time (min)	Perc. of total time	Accuracy
Sky Vase	640x480	223	10.3	0.1% underest.
	320x240	64	2.9	1.3% underest.
	160x120	27	1.2	0.2% underest.
Box	640x480	18.4	12.3	7.30% overest.
	320x240	6.8	4.5	14.00% overest.
	160x120	4.0	2.6	64.90% overest.
Fish	640x480	53.1	11.0	3.03% overest.
	320x240	18.57	3.9	10.47% overest.
	160x120	9.90	2.0	37.64% overest.

When predicting the execution time, one must consider the trade-off between the time spent to predict it and the prediction accuracy. The prediction should be fast enough to allow prompt scheduling decisions to be made and accurate enough to be meaningful for the schedulers. Apart from that, it should be easy to be deployed in practice. One possibility is to render the animation in a much lower resolution and render a *base frame* of the actual animation. Using the execution time of the base frame of the actual and the reduced animation, it is possible to generate a factor to be multiplied on the lower resolution animation to predict the execution actual time of each frame. Figure 7.6 presents predictions using the maximum and execution time frame as base frame. For this experiment, we used 640x480, 320x240, and 160x120 as lower resolutions for generating predictions. If the base frame is the one with maximum execution time, the predictions tend to be overestimated, whereas by using the minimum execution time they tend to be underestimated. We also observe that both 640x480 and 320x240 resolutions using the base frame with maximum execution time provided much better predictions than 160x120. Table 7.1 summarises the execution times to generate the predictions and their accuracies using the base frame with maximum execution time. The results show that good predictions are time consuming since we are using the entire animation.

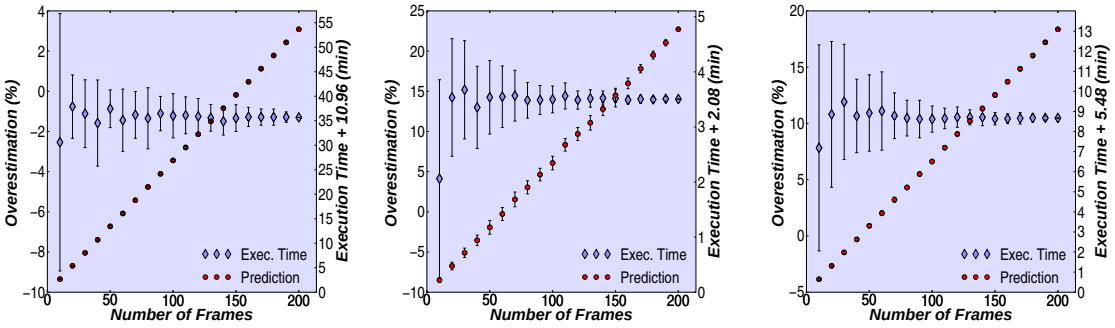
It is possible to reduce the profiling time by sampling a set of frames with a lower resolution rather than using the entire animation. Figures 7.7 and 7.8 show the execution time and the prediction accuracy as a function of the number of frames sampled using resolutions 640x480 and 320x240 respectively. For this experiment, we used the maximum execution time as base frame. The results show that it is possible to considerably

reduce the profiling time keeping a good prediction accuracy level. This happens because in an animation, neighbouring frames have similar content and depending on the case, the variation is minimum during the entire animation, such as for Sky Vase.



(a) Sky Vase. (b) Box. (c) Fish.

Figure 7.7: Predicted execution time partial sampling of 640x480 frames and the most CPU consuming frame as base for estimations.



(a) Sky Vase. (b) Box. (c) Fish.

Figure 7.8: Predicted execution time partial sampling of 320x240 frames and the most CPU consuming frame as base for estimations.

7.4.2 Where to Generate the Estimations

Either users or resource providers can generate run time estimates. If users know the exact resource configuration for each provider, users can execute the run time estimation generator. Also, even if users do not know the configuration, providers that work with virtual machines can make images of these machines available to users for download. Therefore, the user can execute the generator locally. Another option is to allow providers to generate run time estimates at the moment users submit their application requirements to the metascheduler. In this case, providers can have a dedicated set of resources to generate run time estimates, or providers can generate predictions by placing estimators in their shared resources for actual executions.

7.5 Evaluation

This section evaluates the coordinated rescheduling algorithm and the impact of inaccurate run time estimates when scheduling BoT applications on multiple resource providers. We performed experiments using both a simulator and a real testbed. Simulations allowed us to perform repeatable and controllable experiments using various parameters. The experiments in a real testbed allowed us to verify how the scheduler architecture can be used in practice. We used PaJFit, and workloads produced by the Lublin-Feitelson model. For the real experiments, we used an extended version of PaJFit, which works with sockets for communication between modules, on Grid'5000. Following we describe the experiment configuration and the result analysis.

7.5.1 Experimental Configuration

We set up a computing environment with a metascheduler and four clusters, C_{1-4}, with 300 processors each. From this environment, we explored a set of scenarios as described in Table 7.2. The set of experiments with all clusters with the same configuration helps us to analyse the differences between system and user generated estimations and the rescheduling algorithm. The experiment with all clusters with the same configuration but using different policies for run time estimates helps us to understand which resource provider would benefit more from each approach. We also analyse the impact of heterogeneity for scheduling and rescheduling of BoT applications across multiple providers.

Table 7.2: Set up for experiments varying hardware configuration and estimation types. All clusters have 300 each processors.

Hardware	**Estimation Type and rescheduling**
C_1=C_2=C_3=C_4	UE with uncoordinated rescheduling
C_1=C_2=C_3=C_4	SE X and Y more accurate than UEs with uncoordinated rescheduling 5-30min generation time
C_1=C_2=C_3=C_4	UE with coordinated rescheduling
C_1=C_2=C_3=C_4	C_1 and C_2 with UEs and C_3 and C_4 with SEs
C_1=C_2 20 % and 50% faster than C_3=C_4	UEs
C_1=C_2 20 % and 50% faster than C_3=C_4	SEs
C_1=C_2 20 % and 50% faster than C_3=C_4	UEs with coordinated rescheduling

We used the workload model proposed by Lublin and Feitelson [77] to generate traces for both the simulations and the experiments in Grid'5000. We simulated 15 days of the workload and used 15 workloads for each experiment. We also considered 20 run time estimation values. Therefore, for each scenario described in Table 7.2, we have a total

of 300 simulations. For all experiments we set up the system load as 70% by changing the arrival times of the external jobs. To achieve this load we used a strategy similar to that described by Shmueli and Feitelson to evaluate their backfilling strategy [104], but we fixed the time interval and included more jobs from the trace.

We performed our experiments in Grid'5000 by placing a local scheduler in four clusters with access to 300 processors. Table 7.3 presents an overview of the node configurations in which we deployed the local schedulers and the metascheduler. Figure 7.9 illustrates the resource locations in Grid'5000 used in this experiment. We present the results obtained through simulations followed by results from Grid'5000.

Table 7.3: Overview of the node configurations for the experiments in Grid'5000.

Scheduler	Cluster	Location	CPUs' Configuration
metascheduler	sol	Sophia	AMD Opteron 246 2.0 GHz
provider 1	paradent	Rennes	Intel Xeon L5420 2.5 Ghz
provider 2	bordemer	Bordeaux	AMD Opteron 248 2.2 GHz
provider 3	grelon	Lille	AMD Opteron 285 2.6 GHz
provider 4	chicon	Nancy	Intel Xeon 5110 1.6 GHz

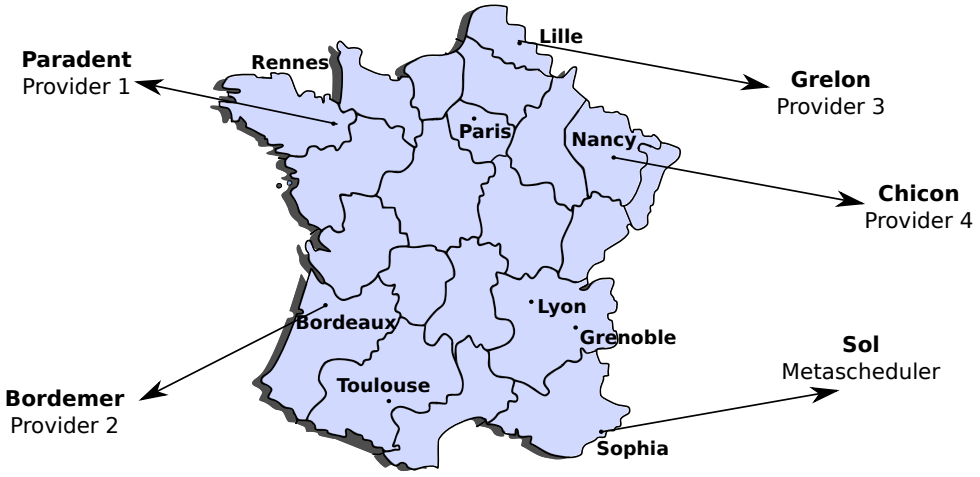
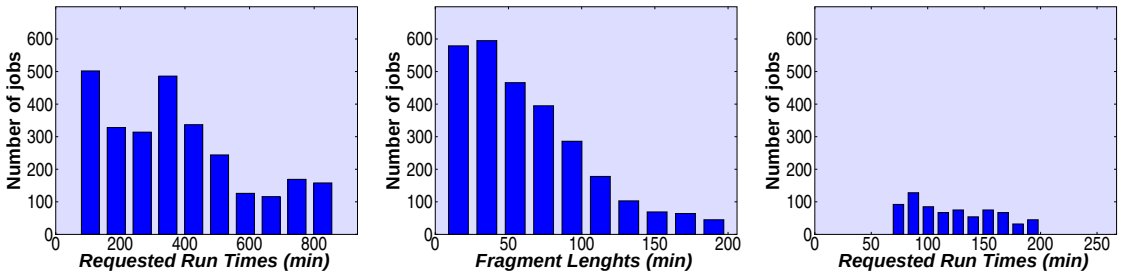


Figure 7.9: Resource location in Grid'5000.

7.5.2 Result Analysis

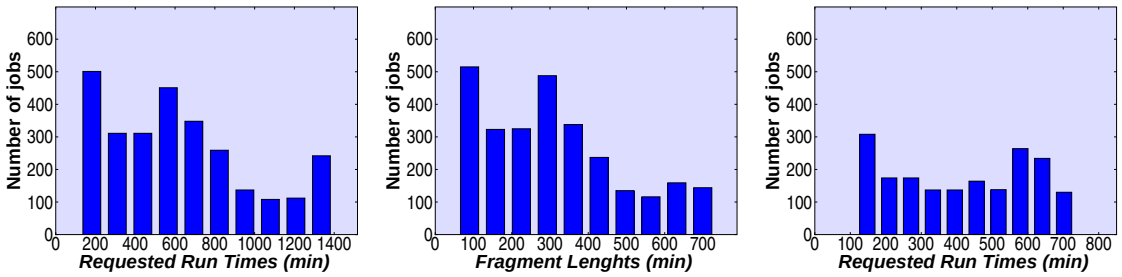
There are two factors related to the reduction of user response times: load balancing and backfilling. Load balancing can be improved by having better run time estimates, since the metascheduler can decide the right amount of work to distribute to each provider, whereas backfilling can fill queue fragments generated by earlier completion times of user requests. These fragments can be filled as long as estimations are shorter or of the same size as the fragments. By increasing user estimations, more fragments are created and therefore more

jobs can be backfilled. However, there is a limit in which backfilling can be explored. Figures 7.10 and 7.11 show the requested run times, fragment lengths, and number of jobs that would fit into the fragments for run time estimates with accuracy of 85% and 50%, respectively. We observe that the higher the accuracy the smaller the number of jobs that have chances of being backfilled. In this example, we are not considering the submission time of the jobs. Figure 7.12 presents the total number of jobs that would have chances of backfilling as a function of run time accuracy. From this figure we notice that there is a limit on the backfilling chances, in particular, after an overestimation of 200% the chances of backfilling become steady due to fragment lengths and requested run times.



(a) Requested run times. (b) Fragment lengths. (c) Chances to be backfilled.

Figure 7.10: Requested run times and fragment lengths for accuracy of 85%.



(a) Requested run times. (b) Fragment lengths. (c) Chances to be backfilled.

Figure 7.11: Requested run times and fragment lengths for accuracy of 50%.

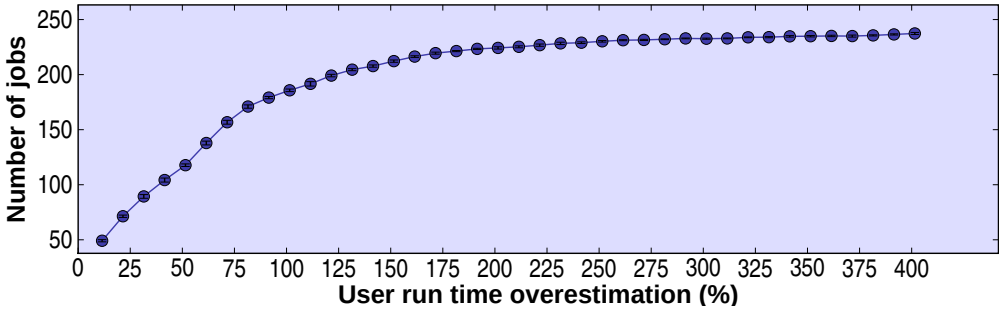
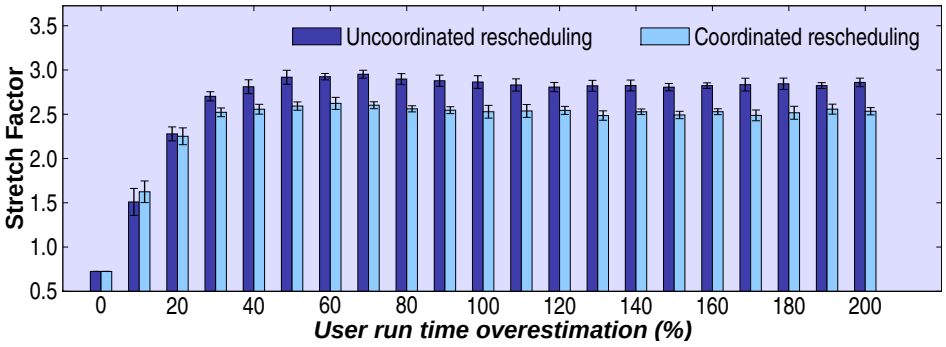


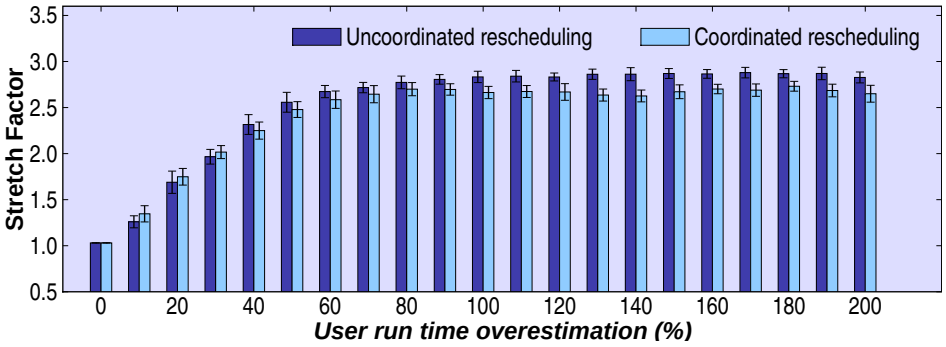
Figure 7.12: Backfilling limit as a function of run time overestimations.

The main motivation for developing the coordinated rescheduling for bag-of-tasks is

the observation that stretch factor increases with the run time overestimations. Figure 7.13 presents the stretch factor for applications scheduled in multiple clusters as a function of run time overestimation for homogeneous and heterogeneous environments. Until 30% of overestimation, there is no difference between the rescheduling strategies. This happens because by this value, just a few jobs have chances of backfilling. However, after 30%, tasks of BoT applications spread over the scheduling queues due to the rescheduling, thus increasing the stretch factor. The coordinated rescheduling minimises this effect, especially when run time accuracy is low.



(a) Homogeneous environment.

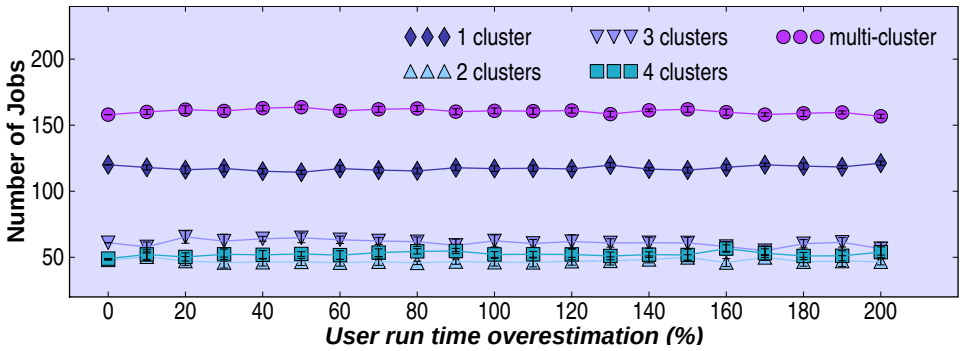


(b) Heterogeneous environment.

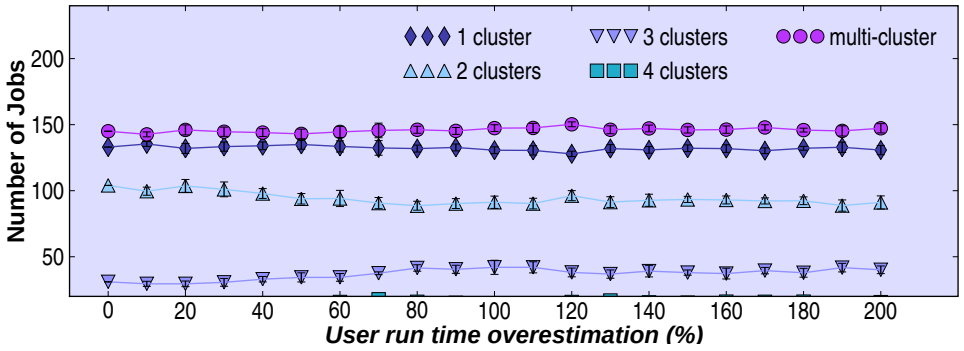
Figure 7.13: Stretch factor variation as a function of the run time estimation accuracy and rescheduling policy.

For the heterogeneous environment, although stretch factor is reduced using coordinated rescheduling over the uncoordinated one, this improvement is slightly lower (Figure 7.13 (b)). The reason is that applications tend to execute in fewer clusters (the fastest ones), and therefore the importance for coordinated rescheduling among providers is reduced. As showed in Figure 7.14, the number of clusters per job is reduced in the heterogeneous environment. Most of the applications are scheduled to one or two clusters, whereas for the homogeneous environment similar number of applications access two, three, and four clusters.

Reducing the stretch factor has a direct impact on the user response time. Figures 7.15



(a) Homogeneous environment.

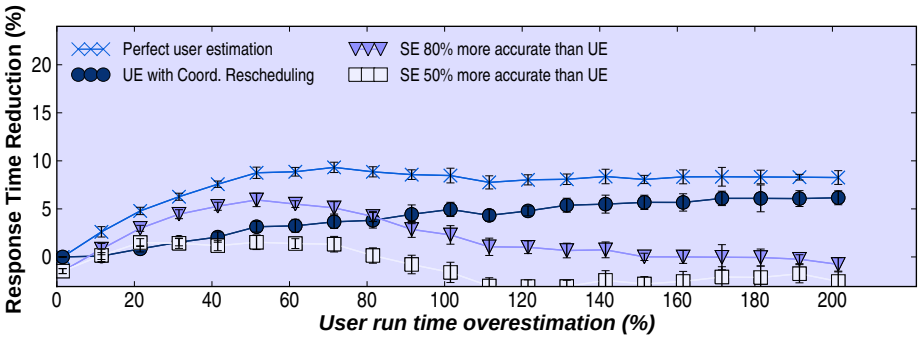


(b) Heterogeneous environment.

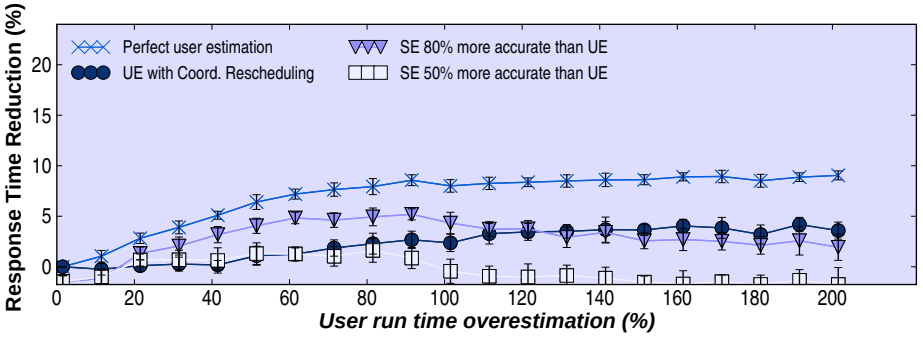
Figure 7.14: Number of clusters per job.

presents the response time reduction of coordinated rescheduling and system-generated predictions for homogeneous and heterogeneous environments in comparison to user run time estimates with uncoordinated rescheduling. We observe that the differences between the policies is higher for the homogeneous environment, since jobs are more distributed to multiple providers than in the heterogeneous environment. In addition, system-generated predictions have better improvements in the heterogeneous environment than in the homogeneous one. The reason is that incorrect load balancing in a heterogeneous environment causes more negative effects than in a homogeneous one. We also observe that system-generated prediction policies have a similar curve shape that perfect user estimation until a certain threshold (60% for homogeneous and 70% for heterogeneous environment). After this threshold the advantage of using system-generated predictions is reduced. This happens because there is a benefit limit in backfilling (as illustrated in Figure 7.12) and it becomes lower than the cost paid to obtain better estimations.

We have also analysed the slowdown (with 10 minutes bound), which is the response time divided by the application run time. Figure 7.16 presents the slowdown for homogeneous and heterogeneous environments. We observe that for this metric, coordinated rescheduling presents even better results than using perfect run time estimations. This happens because this metric highlights the improvements of smaller jobs in relation to big

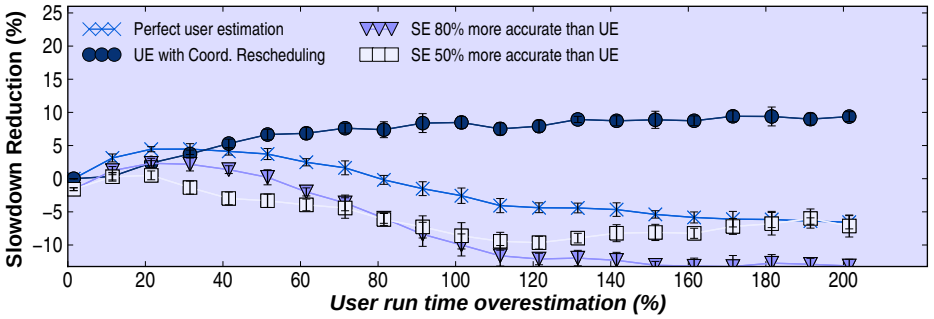


(a) Homogeneous environment.

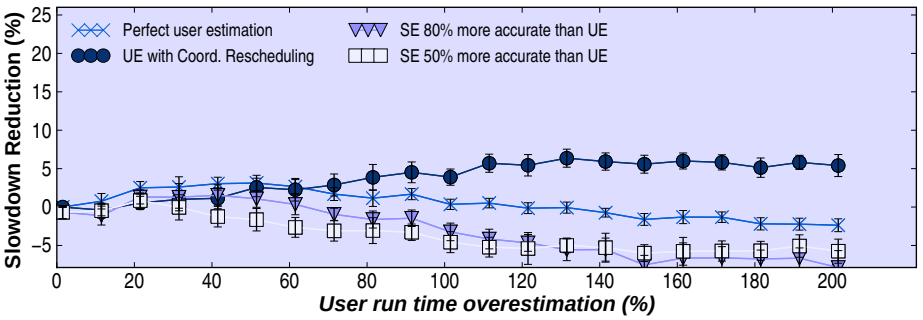


(b) Heterogeneous environment.

Figure 7.15: Global user response time reduction.



(a) Homogeneous environment.

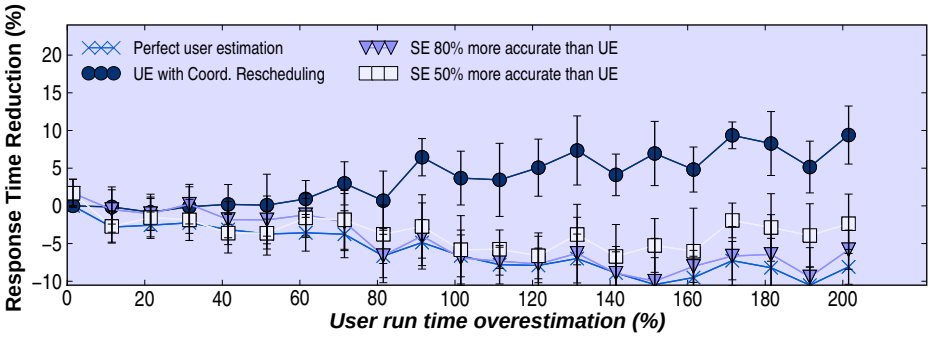


(b) Heterogeneous environment.

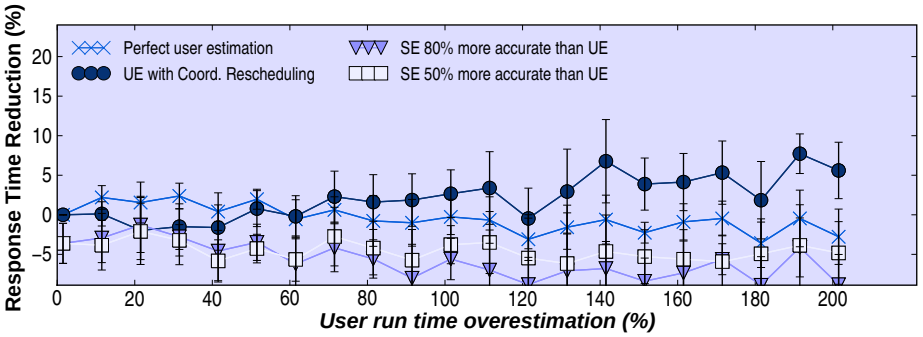
Figure 7.16: Global slowdown reduction.

ones. Smaller jobs have more chances of backfilling than the big ones. Similar happens for the heterogeneous environment.

We also analysed the user response time separately for multi- and single-cluster jobs. Figures 7.17 presents the results for single-cluster jobs. The increase of user overestimations actually reduces user response time for these jobs, which corroborates with previous studies on effects of run time estimates for job scheduling [120]. User response time for coordinated rescheduling produces an improvement of up to 5% in relation to uncoordinated rescheduling for these jobs. The main benefits of higher run time accuracy and coordinated rescheduling come from multi-cluster jobs, as illustrated in Figure 7.18.



(a) Homogeneous environment.

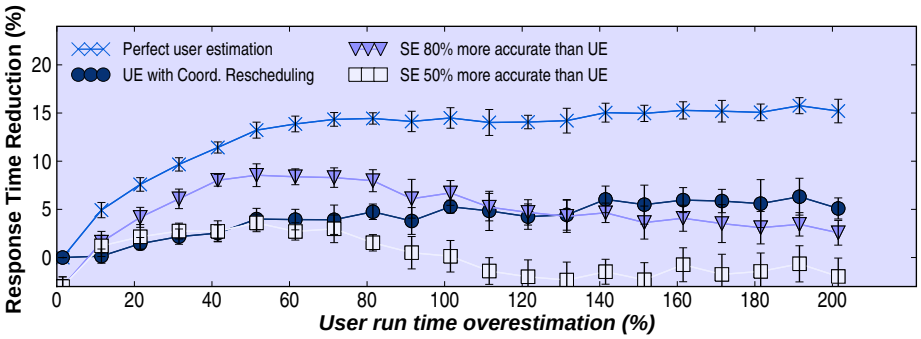


(b) Heterogeneous environment.

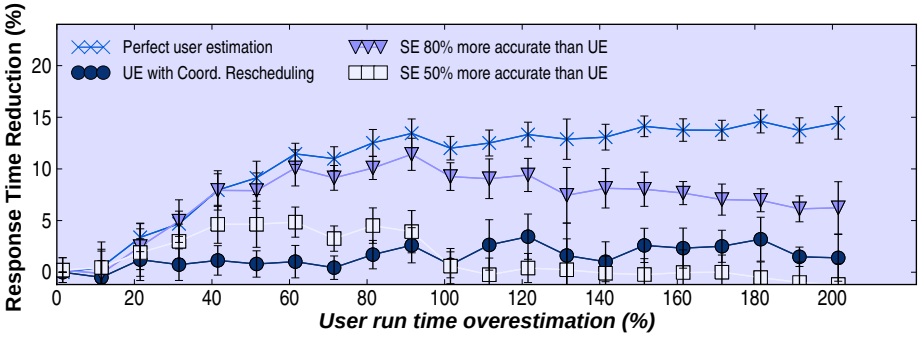
Figure 7.17: User response time reduction for single-cluster jobs.

We have calculated the system utilisation for user and system-generated estimations and uncoordinated/coordinated rescheduling algorithms. The results are similar with a difference of less than 1%. This difference may increase if we consider a competition scenario with providers offering different levels of completion time guarantees. In such a scenario, users tend to execute their applications on providers with more optimised completion time guarantees. Figure 7.19 illustrates the average system utilisation level of providers with different run time estimation approaches; the higher the accuracy of run time predictions the higher the chances of attracting more users.

We have also performed experiments in Grid'5000. Due to the complexity in gathering resources from several sites, usage policy restrictions of large-scale environments,

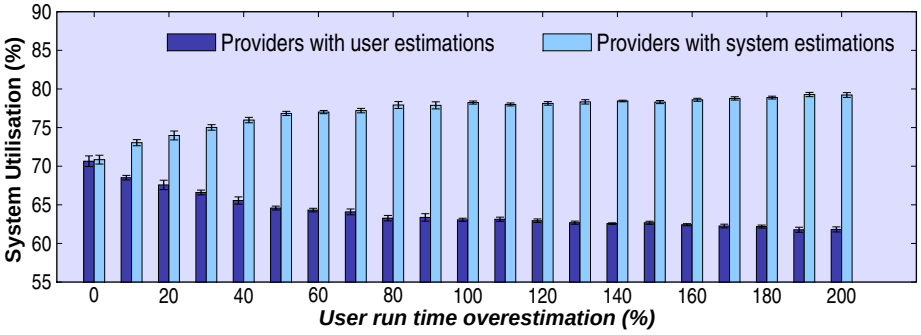


(a) Homogeneous environment.

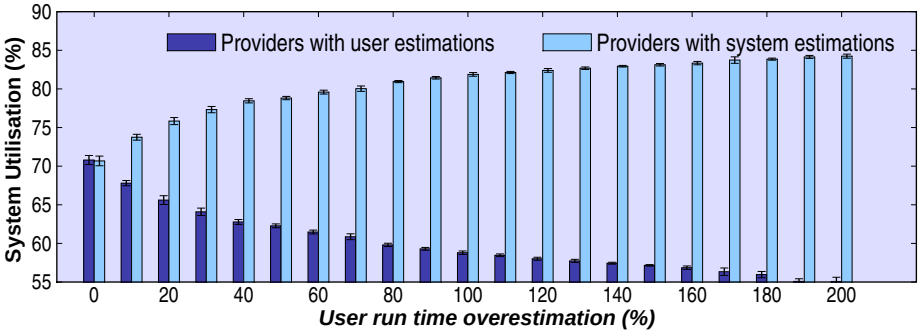


(b) Heterogeneous environment.

Figure 7.18: User response time reduction for multi-cluster jobs.



(a) SE = 0.5 UE



(b) SE = 0.2 UE

Figure 7.19: Impact of estimations on the system utilisation by attracting more users through more optimised completion time guarantees.

and execution time of real experiments, we have selected only three workloads used by the simulations having 50%, 100%, and 150% of run time overestimation parameters. The goal of these experiments is to compare the results of simulation and the execution in the real system. PaJFit has support for both simulations and real execution using sockets as a mechanism for the communication between the metascheduler, providers and users. Therefore, the implementation of the rescheduling algorithm is the same for both execution modes. The main difference lies on the network delay and the order in which messages are exchanged between the system components; in a real system the network delay is higher and the messages require more complex treatment in comparison to simulations. In spite of these differences, we observe in Table 7.4 that for these experiments, both simulations and executions in the real environment provided similar results, showing the practical benefits of coordinated rescheduling in a real environment. As we described in Section 7.2, the required modification in an existing scheduling architecture is minimal.

Table 7.4: Comparison of results from Grid'5000 and simulations.

Metric	Overestimation (%)	From simulation (%)	From real system (%)
SFactor uncoord	50	2.92 ± 0.08	3.05
SFactor uncoord	100	2.86 ± 0.07	2.81
SFactor uncoord	150	2.81 ± 0.04	2.69
SFactor coord	50	2.59 ± 0.05	2.65
SFactor coord	100	2.53 ± 0.07	2.42
SFactor coord	150	2.49 ± 0.04	2.56
Response time red.	50	3.14 ± 0.55	3.84
Response time red.	100	4.95 ± 0.74	6.94
Response time red.	150	5.68 ± 0.74	5.99
Slowdown red.	50	6.66 ± 0.79	6.74
Slowdown red.	100	8.48 ± 0.92	10.4
Slowdown red.	150	8.88 ± 1.29	10.07

7.6 Conclusion

This chapter presented a coordinated rescheduling algorithm for BoT applications executing across multiple providers and the impact of run time estimates for these applications. Due to inaccurate run time estimates, initial schedules have to be updated, and therefore, when each provider reschedules tasks of a BoT application independently, such tasks may have their completion time reduced locally, but not globally. Tasks of the same BoT can be spread over time due to rescheduling. The main idea of coordinated rescheduling for

BoT applications is to consider the completion time of the entire BoT. Therefore, small jobs can have more chances of backfilling without delaying the BoT applications.

Moreover, accurate run time estimates assist metaschedulers to better distribute the tasks of BoT applications on multiple sites. Although system generated predictions may consume time, the schedules produced by more accurate estimates pay off the profiling time since users have better response times than simply overestimating resource usages.

This chapter meets the three objectives proposed in Chapter 1 for BoT applications, which concludes the core chapters of the thesis. Following we discuss the main findings of the thesis and future research directions for resource co-allocation.

Chapter 8

Conclusions and Future Directions

Resource co-allocation in distributed computing systems is a complex problem mainly because resources are managed by autonomous providers. Distributed transactions, fault tolerance, inter-site network overhead, and schedule optimisation are the four major challenges we identified in this research field. From the literature review, we observed a lack of research on rescheduling applications accessing resources from multiple providers. Therefore, we set as the aim of this thesis to investigate the benefits for users and resource providers when rescheduling message passing and bag-of-tasks applications on multiple autonomous providers. To answer this question, we defined the following objectives:

- Understand the impact of inaccurate run time estimates in computing environments with applications co-allocating resources from multiple providers;
- Design, implement, and evaluate co-allocation policies with rescheduling support;
- Investigate technical difficulties to deploy the co-allocation policies in real environments.

This thesis contains an analysis of the impact of inaccurate run time estimates in three scenarios: a single provider scheduling advance reservations, multiple providers scheduling message passing applications, and multiple providers scheduling bag-of-tasks. It also proposes co-allocation policies with rescheduling support for message passing and bag-of-tasks application models, along with a description of technical difficulties to deploy these policies. The proposed policies consider four aspects: inaccurate run time estimations of user applications, completion time guarantees, coordinated rescheduling, and limited information access from resource providers.

As an important building block of resource co-allocation for message passing applications, we started the thesis by investigating flexible advance reservations. These advance

reservations have flexible start and completion time intervals, which can be explored by schedulers to increase system utilisation when unexpected events happen. One common event is the completion of executions before the expected time. We investigated the importance of rescheduling advance reservations for system utilisation using four scheduling heuristics under several workloads, reservation time intervals and inaccurate run time estimates. In addition, we studied cases when users accept an alternative offer from the resource provider on failure to schedule the initial request. Our main finding on this study is that system utilisation increases with the flexibility of request time intervals and the time users allow this flexibility while waiting for resource access. This benefit is mainly due to the ability of the scheduler to rearrange jobs in the scheduling queue, which reduces the fragmentation generated by advance reservations. This is particularly true when users overestimate application run time.

Based on flexible advance reservations for single resource provider settings, we extended the concept of flexible time intervals for applications requiring co-allocation of resources from multiple providers. We proposed a co-allocation model that relies on two operations to reschedule requests: start time shifting and process remapping. By using this model, metaschedulers can modify the start time of each job component and remap the number of processors they use in each provider. From our experiments, using workloads from real clusters, we showed 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, process remapping increases the chances of placing the tasks of multi-cluster jobs into a single cluster, thus eliminating the inter-cluster network overhead.

From the deployment point of view of the adaptive resource co-allocation for message passing applications, we observed that the use of start time shifting can be widely adopted by several parallel applications. This operation is not application dependent since it is only a shift on the start time of the user application. The process remap operation, on the other hand, is application dependent, which may limit its adoption for some applications. To better understand how to use this operation in practice, we developed an application-level scheduler for iterative parallel applications. We concluded that users can remap the processes with the cost of overestimating the execution time to avoid applications being aborted by the schedulers. To overcome this problem, metaschedulers can use performance predictions, and in particular for iterative applications, the cost to obtain predictions is negligible and requires no access to the user application source code.

Regarding resource co-allocation for BoT applications, we investigated how to distribute tasks of the same application on providers that are not willing to disclose private information to metaschedulers. We mainly focused on two types of information: total

computing power of a resource provider and its local load. To keep this information private, we introduced the concept of execution offers, in which resource providers advertise their interest in executing an entire BoT application or only part of it without revealing their load and total computing power. The main findings from this study are that offer-based scheduling produces less delay for jobs that cannot meet deadlines in comparison to scheduling based on load availability (i.e. free time slots); thus it is possible to keep providers' load private when scheduling multi-site BoT applications; and if providers publish their total computing power they can have more local jobs meeting deadlines.

As one of the key aspects of this thesis is the inaccurate run time estimates of user applications, we investigated the importance of accurate predictions when scheduling BoT applications on multiple providers and how tasks from the same application should be rescheduled. We observed that tasks of the same BoT can be spread over time due to inaccurate run time estimates and environment heterogeneity. To minimise the effect of having completion time variation of tasks from the same application, this thesis proposes a coordinated rescheduling algorithm, which reduces response time for both users accessing a single and multiple providers. We also observed that accurate run time estimates assist metaschedulers to better distribute the tasks of BoT applications on multiple sites. In order to obtain accurate run time estimates, users or the metascheduler can profile a sample of tasks from the same application. We concluded that the cost to obtain run time predictions pays off since users have better response times than simply overestimating resource usages.

8.1 Future Research Directions

Resource co-allocation is one of the main requirements to enable the execution of applications on multiple providers. Due to the demand for Quality-of-service, several researchers have been relying on advance reservations for resource co-allocation. Therefore, we believe that most of the future work on resource co-allocation will continue to follow this approach as well. In addition, we have seen more researchers working on negotiation mechanisms for co-allocation requests in order to better satisfy user demand and resource provider requirements [36, 44, 75, 107]. Negotiation is an important mechanism to avoid providers disclosing private information, such as load and resource capabilities, to the metascheduler.

Another research opportunity is the development of rescheduling policies for co-allocation requests. As users cannot predict their application run times, the scheduler has to reschedule them frequently. Other reasons for rescheduling applications are resource failures, dynamic resource demand, and optimisation of metrics such as system

utilisation, power consumption, and user response time. Rescheduling has also impact on management of contracts, also called Service Level Agreements, in utility computing environments. In these environments, users pay to access resources or services, and providers have to guarantee the delivery of these services with a pre-established Quality-of-Service level. For co-allocation users, several entities may participate in these contracts and hence managing issues such as violation becomes a complex task. Thus, for the coming years, especially due to the increasing number of utility computing centers around the world, researchers will be facing the challenge of developing and improving existing policies for managing contracts involving multiple entities

Virtualisation is another concept that will be highly explored to provide transparency to users when co-allocating multiple resources that are hosted in either a single or multiple administrative domains. Virtual clusters can be dynamically formed to deploy applications with various application requirements [29]. Moreover, with the consolidation of Cloud Computing, resource/service provisioning centers can avoid contract violations and increase system utilisation by co-allocation resources from multiple parties on demand.

In the following sections we describe in more detail some of the future directions identified during the development of this thesis.

8.1.1 Resource Co-allocation in Cloud Computing Environments

Cloud computing has emerged as an important platform to reduce costs and simplify access to IT resources. Cloud computing users vary from individuals such as home users, scientists, and educators, to small and medium-sized companies. One of the main challenges of Cloud computing is to provide users with the experience of accessing remote resources in the same way as they access local resources. This level of transparency can be achieved by offering to users services that are easily accessible and coupled with complex user demand, in particular from critical business applications. Another challenge is to deliver services that are robust; i.e. services that can handle change of plans from both user and provider sides, which is especially difficult when multiple participants are involved.

Transparency and robustness of services can be achieved through easy-to-use interfaces and interoperability among resource providers. One of the main projects that considers interoperability among cloud providers is RESERVOIR [100], which is a consortium with several institutes involved, including IBM, Telefonica, and Sun Microsystems. In the RESERVOIR model, users interact with service providers, which understand user business and are responsible for gathering resources to meet user requirements. By using this model, users have faster and more robust feedback, service providers have higher resource availability, and resource providers increase the chances of meeting user demand

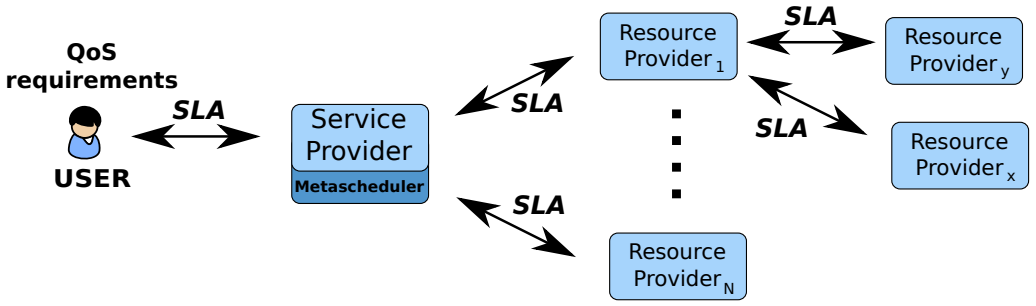


Figure 8.1: Multiple contracts to meet user QoS requirements.

at peek times by borrowing resources from other providers.

Gathering resources from multiple autonomous providers is challenging due to the uncertainty of both user demand and providers' resource availability. Moreover, once the initial resource selection is completed and contracts are established, changes of plans may arise from both user and provider sides. User changes may come from new project deadlines, or cut in costs for IT resources, whereas provider changes may come from unexpected increase demand of resources or high priority users that require immediate resource access. These changes have to be easily solved without causing impact on participants. Re-planning is simplified by using common and easy-to-use service interfaces, which assist outsourcing in case original participants cannot offer the quality-of-service defined in the contract between users and providers.

This thesis proposed adaptive resource co-allocation policies for message passing and BoT applications. The policies are adaptive since they support re-planning of application schedules. Applications require constant re-scheduling (or re-planning) due to inaccurate run time estimations in order to reduce user response time and increase resource utilisation. Further research in this context involves:

- Development of adaptive resource co-allocation policies for wider range of applications, including data-intensive and web applications;
- Management of contracts involving multiple participants. Note that chains of contracts, i.e. Service Level Agreements, may be required to meet user QoS requirements (Figure 8.1);
- Interoperable interfaces for communication between providers from different companies;
- When co-allocating resources from multiple cloud computing providers, the cost to access resources becomes an important issue [58, 132]. Metaschedulers have to consider prices from multiple resource providers, which impact on the utility of the user requesting resources.

As cloud computing technology evolves, cloud centers will provide different services that users will want to compose to meet their demand. Therefore, co-allocation policies, especially with re-planning features, will be fundamental to make cloud computing the new utility service for individual users and organisations.

8.1.2 Negotiation Protocols for Concurrent Co-allocation Requests

An increasing number of researchers are working on negotiation mechanisms for co-allocation requests in order to better satisfy user demand and resource provider requirements [36, 44, 75, 107]. In addition, negotiation is an important mechanism to avoid providers disclosing private information, such as load and resource capabilities, to the metascheduler. This thesis proposed the use of execution offers for deadline-constrained BoT applications, which is the initial step towards the development of negotiation protocols.

Chapter 2 described some of the existing work in the area of distributed transactions for resource co-allocation. Deadlocks and livelocks may arise when more than one client asks for resources at the same time from the same providers (Figure 8.2). One research direction in this scenario is to design negotiation protocols that reduce the number of messages required by all participants to achieve their goals. Resource providers and metaschedulers could use existing negotiation protocols until they find there is a resource contention from multiple requests. From that moment, metaschedulers and providers should generate offers that take into account the identified contention in order to come up with an agreement.

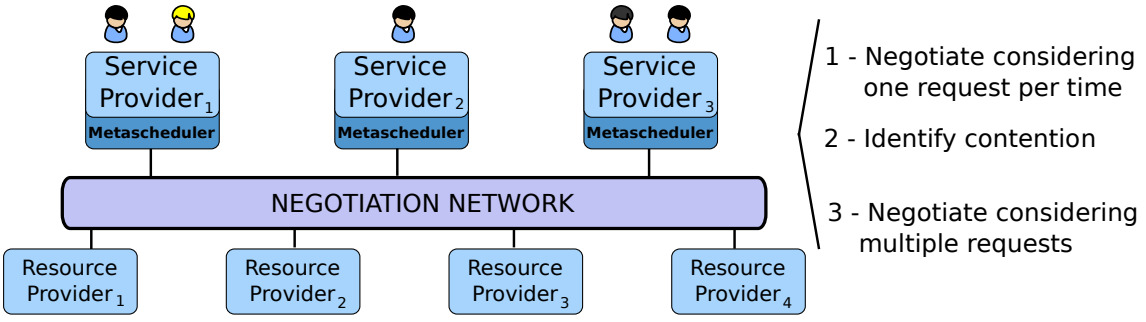


Figure 8.2: Multiple contracts to meet user QoS requirements.

8.1.3 Energy Consumption and Data-Transfer Constraints

The use of co-allocation for message passing applications often comes with a cost in inter-cluster communication. Studies in co-allocation showed that for inter-cluster communication overhead of up to 25% of execution time, co-allocation pays off (see Section 2.2.3 in

Chapter 2). However, this increase in execution time comes with energy-consumption of clusters and communication devices (Figure 8.3). A future direction is to evaluate energy-consumption [71] when executing applications over multiple providers. This research would require a detailed power consumption monitoring system of resources involved in the computation and communication. A possible outcome would be a new threshold for inter-cluster communication overhead that pays off the benefits of co-allocation.

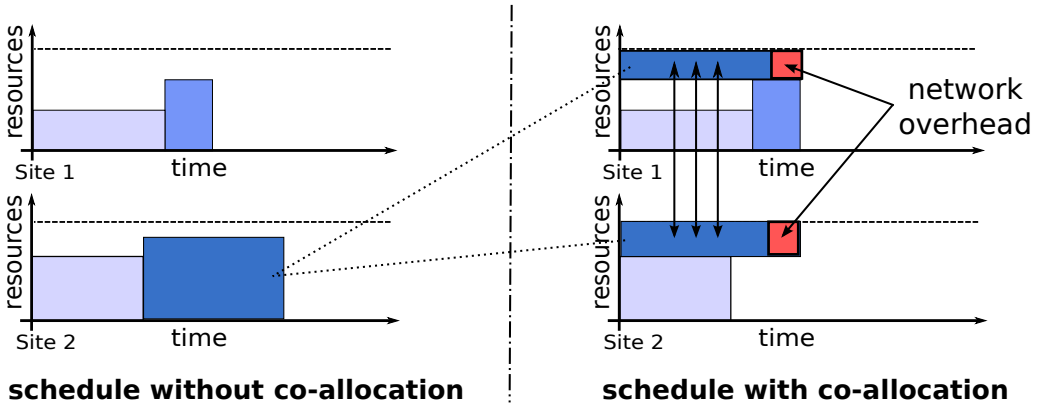


Figure 8.3: Network overhead due to inter-site latency.

Several data-intensive applications [123] require a considerable amount of computing power, which can be only achieved by co-allocating resources from multiple providers. There are two aspects to be investigated for these applications: co-allocation and rescheduling. Three research questions in this regard are:

- How to co-allocate resources considering that rescheduling may be required in future due to an unexpected event?
- How long should the metascheduler delay the data transfer to increase the chances of rescheduling options?
- When should resource providers notify the metascheduler about the interest in rescheduling tasks of a data-intensive application?

8.1.4 Other Research Directions

During the development of this thesis, we identified other research opportunities that we will not describe in detail here, but give an overview about them:

- **Workloads with deadlines.** Current work on scheduling, including this thesis, uses deadline generators based on a distribution function and/or on job sizes. It would be interesting to investigate more methods for deadline generation;

- **Run time rescheduling.** This thesis investigated rescheduling for jobs waiting for resources in scheduling queues. A possible extension is to considering the rescheduling policies for jobs already accessing resources.

- **Co-allocation and rescheduling using moldatiblity.** The use of moldability has been investigated for single-cluster applications [32, 114]. However, its use for multi-cluster applications requires further investigation.

- **Admission control with other criteria.** This thesis used expected completion time as the criterion for admission control. Other criteria, such as chances of meeting completion time proposals in case of failures or unexpected peek demand could be also investigated.

- **Different types of resources.** This thesis focused mainly on allocating processors to execute parallel applications. Further research involves the allocation of other types of resources, including network links, scientific devices, and software systems.

8.2 Final Remarks

Resource co-allocation is a challenging problem that has been increasingly important for several distributed applications. Heterogeneity and network overhead are still the main obstacles for deploying applications on multiple providers, especially if these applications require inter-process communication. Virtual machines will definitely play a major role to overcome heterogeneity issues from both software and hardware levels. The network problem can be minimised by smart scheduling decisions and application models that work with asynchronous communications.

User demand and resource availability vary over time, and unexpected events are a reality that needs to be addressed. This thesis explored rescheduling of applications over multiple autonomous providers. The trend shows there will be a shift towards the use of multiple Cloud Computing providers. Users have already started to gather local and external resources from single providers [37], and when Cloud providers start to offer inter-operable interfaces, users will have the option to deploy their applications also using services from multiple companies. Therefore, the adaptive co-allocation policies proposed in this thesis can have direct impact on the Cloud Computing settings in near future.

REFERENCES

[1] Abramson, D., Buyya, R., and Giddy, J. (2002). A computational economy for grid computing and its implementation in the Nimrod-G resource broker. *Future Generation Computer Systems.*, 18(8):1061–1074.

[2] Abramson, D., Giddy, J., and Kotler, L. (2000). High performance parametric modeling with Nimrod/G: Killer application for the global grid? In *Proceedings of the 14th International Parallel and Distributed Processing Symposium (IPDPS'00)*, Cancun, Mexico. IEEE Computer Society.

[3] Alhusaini, A. H., Prasanna, V. K., and Raghavendra, C. S. (2000). A framework for mapping with resource co-allocation in heterogeneous computing systems. In Raghavendra, C., editor, *Heterogeneous Computing Workshop (HCW'00)*, pages 273–286, Los Alamitos, California. IEEE Computer Society.

[4] Alhusaini, A. H., Raghavendra, C. S., and Prasanna, V. K. (2001). Run-time adaptation for grid environments. In Werner, B., editor, *International Parallel and Distributed Processing Symposium (IPDPS'01)*, pages 864–874, Los Alamitos, California. IEEE Computer Society.

[5] Altschul, S., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. (1990). Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410.

[6] Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al. (2009). Above the clouds: A Berkeley view of cloud computing. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28*.

[7] Aumage, O. and Mercier, G. (2003). MPICH/MADIII: a cluster of clusters enabled MPI implementation. In Titsworth, F. and Azada, D., editors, *International Symposium on Cluster Computing and the Grid (CCGrid'03)*, pages 26–, Los Alamitos, California. IEEE Computer Society.

[8] Auyoung, A., Grit, L., Wiener, J., and Wilkes, J. (2006). Service contracts and aggregate utility functions. In *Proceedings of the 15th International Symposium on High Performance Distributed Computing (HPDC'06)*, Paris, France. IEEE.

[9] Azougagh, D., Yu, J.-L., Kim, J.-S., and Maeng, S. R. (2005). Resource co-allocation: A complementary technique that enhances performance in grid computing environment. In Barolli, L., editor, *International Conference on Parallel and Distributed Systems (ICPADS'05)*, volume 1, pages 36–42, Los Alamitos, California. IEEE Computer Society.

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

[11] Bahi, J. M., Contassot-Vivier, S., and Couturier, R. (2006). Performance comparison of parallel programming environments for implementing AIAC algorithms. *The Journal of Supercomputing*, 35(3):227–244.

[12] Bal, H. E., Plaat, A., Bakker, M. G., Dozy, P., and Hofman, R. F. H. (1998). Optimizing parallel applications for wide-area clusters. In *Proceedings of the 12th International Parallel Processing Symposium / 9th Symposium on Parallel and Distributed Processing (IPPS/SPDP'98)*.

[13] Beaumont, O., Carter, L., Ferrante, J., Legrand, A., Marchal, L., and Robert, Y. (2008). Centralized versus distributed schedulers for bag-of-tasks applications. *IEEE Transactions on Parallel and Distributed Systems*, 19(5):698–709.

[14] Benoit, A., Marchal, L., Pineau, J.-F., Robert, Y., and Vivien, F. (2008). Offline and online master-worker scheduling of concurrent bags-of-tasks on heterogeneous platforms. In *Proceedings of the 22nd IEEE International Symposium on Parallel and Distributed Processing (IPDPS'00)*, Miami, USA. IEEE Computer Society.

[15] Berman, F., Casanova, H., Chien, A. A., Cooper, K. D., Dail, H., Dasgupta, A., Deng, W., Dongarra, J., Johnsson, L., Kennedy, K., Koelbel, C., Liu, B., Liu, X., Mandal, A., Marin, G., Mazina, M., Mellor-Crummey, J. M., Mendes, C. L., Olugbile, A., Patel, M., Reed, D. A., Shi, Z., Sievert, O., Xia, H., and YarKhan, A. (2005). New grid scheduling and rescheduling methods in the GrADS project. *International Journal of Parallel Programming*, 33(2-3):209–229.

[16] Berman, F., Wolski, R., Casanova, H., Cirne, W., Dail, H., Faerman, M., Figueira, S. M., Hayes, J., Obertelli, G., Schopf, J. M., Shao, G., Smallen, S., Spring, N. T., Su, A., and Zagorodnov, D. (2003). Adaptive computing on the grid using AppLeS. *IEEE Transactions on Parallel and Distributed Systems*, 14(4):369–382.

[17] Bernstein, P. A. and Goodman, N. (1981). Concurrency control in distributed database systems. *ACM Computing Surveys*, 13(2):185–221.

[18] Bolze, R., Cappello, F., Caron, E., Daydé, M., Desprez, F., Jeannot, E., Jégou, Y., Lanteri, S., Leduc, J., Melab, N., et al. (2006). Grid'5000: a large scale and highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, 20(4):481.

[19] Braun, T. D., Siegel, H. J., Beck, N., Bölöni, L., Maheswaran, M., Reuther, A. I., Robertson, J. P., Theys, M. D., Yao, B., Hensgen, D. A., and Freund, R. F. (2001). A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810–837.

[20] Bucur, A. I. D. and Epema, D. H. J. (2003a). The performance of processor coallocation in multicluster systems. In Titsworth, F. and Azada, D., editors, *International Symposium on Cluster Computing and the Grid (CCGrid'03)*, pages 302–309, Los Alamitos, California. IEEE Computer Society.

[21] Bucur, A. I. D. and Epema, D. H. J. (2003b). Priorities among multiple queues for processor co-allocation in multicluster system. In Bilof, R., editor, *Annual Simulation Symposium (ANSS'03)*, pages 15–27, Los Alamitos, California. IEEE Computer Society.

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

[23] Buyya, R. (1999). *High Performance Cluster Computing*. Prentice-Hall, Upper Saddle River, NJ.

[24] Buyya, R., Yeo, C., Venugopal, S., Broberg, J., and Brandic, I. (2009). Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616.

[25] Capit, N., Costa, G. D., Georgiou, Y., Huard, G., Martin, C., Mounié, G., Neyron, P., and Richard, O. (2005). A batch scheduler with high level components. In *International Symposium on Cluster Computing and the Grid (CCGrid'05)*, pages 776–783, Los Alamitos, California. IEEE Computer Society.

[26] Cappello, F., Caron, E., Daydé, M. J., Desprez, F., Jégou, Y., Primet, P. V.-B., Jeannot, E., Lanteri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Quétier, B., and Richard, O. (2005). Grid'5000: a large scale and highly reconfigurable grid experimental testbed. In *International Conference on Grid Computing (GRID'05)*, pages 99–106, Los Alamitos, California. IEEE.

[27] Carriero, N. and Gelernter, D. (1989). How to write parallel programs: A guide to the perplexed. *ACM Computing Surveys*, 21(3):323–357.

[28] Casanova, H., Legrand, A., Zagorodnov, D., and Berman, F. (2000). Heuristics for scheduling parameter sweep applications in grid environments. In *Proceedings of the Heterogeneous Computing Workshop (HCW'00)*, pages 349–363, Cancun, Mexico. IEEE Computer Society.

[29] Chase, J. S., Irwin, D. E., Grit, L. E., Moore, J. D., and Sprenkle, S. (2003). Dynamic virtual clusters in a grid site manager. In *International Symposium on High-Performance Distributed Computing (HPDC'03)*, pages 90–103, Los Alamitos, California. IEEE Computer Society.

[30] Chen, Y. T. and Lee, K. H. (2001). A flexible service model for advance reservation. *Computer Networks*, 37(3/4):251–262.

[31] Chiang, S.-H., Arpaci-Dusseau, A. C., and Vernon, M. K. (2002). 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 (JSSPP'02)*, volume 2537 of *Lecture Notes in Computer Science*, pages 103–127, Edinburgh, Scotland, UK. Springer.

[32] Cirne, W. and Berman, F. (2002). Using moldability to improve the performance of supercomputer jobs. *Journal of Parallel and Distributed Computing*, 62(10):1571–1601.

[33] Cirne, W., Paranhos, D., Costa, L., Santos-Neto, E., Brasileiro, F., Sauvé, J., Silva, F., Barros, C., and Silveira, C. (2003). Running bag-of-tasks applications on computational grids: The mygrid approach. In *Proceedings of the International Conference on Parallel Processing (ICPP'03)*, pages 407–416.

[34] Czajkowski, K., Foster, I., Karonis, N., Kesselman, C., Martin, S., Smith, W., and Tuecke, S. (1998). A resource management architecture for metacomputing systems. In Feitelson, D. G. and Rudolph, L., editors, *International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'98)*, volume 1459 of *Lecture Notes in Computer Science*, pages 62–82, Berlin. Springer.

[35] Czajkowski, K., Foster, I., and Kesselman, C. (1999). Resource co-allocation in computational grids. In *International Symposium on High Performance Distributed Computing (HPDC'99)*, pages 219–228, Los Alamitos, California. IEEE Computer Society.

[36] Czajkowski, K., Foster, I. T., Kesselman, C., Sander, V., and Tuecke, S. (2002). SNAP: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In Feitelson, D. G., Rudolph, L., and Schwiegelshohn, U., editors, *Proceedings of the 8th International Workshop Job Scheduling Strategies for Parallel Processing (JSSPP'02)*, Lecture Notes in Computer Science, pages 153–183, Berlin. Springer.

[37] de Assunção, M. D., di Costanzo, A., and Buyya, R. (2009). Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters. In Kranzlmüller, D., Bode, A., Hegering, H.-G., Casanova, H., and Gerndt, M., editors, *Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing (HPDC'09)*, pages 141–150, Garching, Germany.

[38] Deb, K. (2001). *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons.

[39] Deb, K., Thiele, L., Laumanns, M., and Zitzler, E. (2005). Scalable test problems for evolutionary multiobjective optimization. *Evolutonary Multiobjective Optmization*.

[40] Decker, J. and Schneider, J. (2007). Heuristic scheduling of grid workflows supporting co-allocation and advance reservation. In Schulze, B., Buyya, R., Navaux, P., Cirne, W., and Rebello, V., editors, *International Symposium on Cluster Computing and the Grid (CCGrid'07)*, pages 335–342, Los Alamitos, California. IEEE Computer Society.

[41] Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Blackburn, K., Lazzarini, A., Arbree, A., Cavanaugh, R., et al. (2003). Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing*, 1(1):25–39.

[42] Desell, T. J., Szymanski, B. K., and Varela, C. A. (2008). Asynchronous genetic search for scientific modeling on large-scale heterogeneous environments. In *Proceedings of the 17th Heterogeneity in Computing Workshop (HCW'08), in conjunction with 22nd IEEE International Symposium on Parallel and Distributed Processing (IPDPS'08)*.

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

[44] Elmroth, E. and Tordsson, J. (2009). A standards-based grid resource brokering service supporting advance reservations, coallocation and cross-grid interoperability. *Concurrency and Computation: Practice and Experience*, 25(18):2298 – 2335.

[45] Ernemann, C., Hamscher, V., Schwiegelshohn, U., Yahyapour, R., and Streit, A. (2002a). On advantages of grid computing for parallel job scheduling. In *International Symposium on Cluster Computing and the Grid (CCGrid'02)*, pages 39–, Los Alamitos, California. IEEE Computer Society.

[46] Ernemann, C., Hamscher, V., Streit, A., and Yahyapour, R. (2002b). Enhanced algorithms for multi-site scheduling. In Parashar, M., editor, *Proceedings of the 3rd International Workshop on Grid Computing (GRID'02)*, volume 2536 of *Lecture Notes in Computer Science*, pages 219–231, Berlin. Springer.

[47] Farooq, U., Majumdar, S., and Parsons, E. W. (2006). 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'06)*, pages 373–380, Columbus, USA. IEEE Computer Society.

[48] Feitelson, D. G., Rudolph, L., Schwiegelshohn, U., Sevcik, K. C., and Wong, P. (1997). Theory and practice in parallel job scheduling. In Feitelson, D. G. and Rudolph, L., editors, *Proceedings of the 3rd Workshop on Scheduling Strategies for Parallel Processing (JSSPP'97)*, volume 1291 of *Lecture Notes in Computer Science*, pages 1–34, Geneva, Switzerland. Springer.

[49] Ferrari, D., Gupta, A., and Ventre, G. (1997). Distributed advance reservation of real-time connections. *Multimedia Systems*, 5(3):187–198.

[50] Foster, I. and Kesselman, C. (1999). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufman, San Francisco, CA, EUA.

[51] Foster, I., Kesselman, C., Lee, C., Lindell, B., Nahrstedt, K., and Roy, A. (1999). A distributed resource management architecture that supports advance reservations and co-allocation. In *Proceedings of the International Workshop on Quality of Service (IWQoS'99)*, pages 27–36, Piscataway, New Jersey. IEEE Computer Society.

[52] Foster, I., Kesselman, C., and Tuecke, S. (2001). The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200.

[53] Gray, J. and Lamport, L. (2006). Consensus on transaction commit. *ACM Transactions on Database Systems*, 31(1):133–160.

[54] Gropp, W., Lusk, E., and Skjellum, A. (1999). *Using MPI: portable parallel programming with the message passing interface*. MIT press.

[55] Haji, M. H., Gourlay, I., Djemame, K., and Dew, P. M. (2005). A SNAP-based community resource broker using a three-phase commit protocol: A performance study. *The Computer Journal*, 48(3):333–346.

[56] He, L., Jarvis, S. A., Spooner, D. P., Chen, X., and Nudd, G. R. (2004). Dynamic scheduling of parallel jobs with QoS demands in multiclusters and grids. In *Proceedings of the International Conference on Grid Computing (GRID'04)*.

[57] Iosup, A., Sonmez, O. O., Anoep, S., and Epema, D. H. J. (2008). The performance of bags-of-tasks in large-scale distributed systems. In *Proceedings of the 17th International Symposium on High-Performance Distributed Computing (HPDC'08)*, pages 97–108, Boston, USA. ACM.

[58] Irwin, D. E., Grit, L. E., and Chase, J. S. (2004). Balancing risk and reward in a market-based task service. In *Proceedings of the 13th International Symposium on High-Performance Distributed Computing (HPDC'04)*, pages 160–169, Honolulu, USA. IEEE Computer Society.

[59] Islam, M., Balaji, P., Sabin, G., and Sadayappan, P. (2007). Analyzing and minimizing the impact of opportunity cost in QoS-aware job scheduling. In *Proceedings of the International Conference on Parallel Processing (ICPP'07)*, page 42, Xi-An, China. IEEE Computer Society.

[60] Islam, M., Balaji, P., Sadayappan, P., and Panda, D. K. (2003). QoPS: A QoS Based Scheme for Parallel Job Scheduling. In *Proceedings of the 9th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'03)*, volume 2862 of *Lecture Notes in Computer Science*, pages 252–268, Seattle, USA. Springer.

[61] Islam, M., Balaji, P., Sadayappan, P., and Panda, D. K. (2004). Towards provision of quality of service guarantees in job scheduling. In *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'04)*, pages 245–254, San Diego, USA. IEEE Computer Society.

[62] Jardine, J., Snell, Q., and Clement, M. J. (2001). Livelock avoidance for meta-schedulers. In Williams, A. D., editor, *International Symposium on High Performance Distributed Computing (HPDC'01)*, pages 141–146, Los Alamitos, California. IEEE Computer Society.

[63] Jarvis, S. A., Spooner, D. P., Keung, H. N. L. C., Cao, J., Saini, S., and Nudd, G. R. (2006). Performance prediction and its use in parallel and distributed computing systems. *Future Generation Computer Systems*, 22(7):745–754.

[64] Jones, W. M., III, W. B. L., and Shrivastava, N. (2006). The impact of information availability and workload characteristics on the performance of job co-allocation in multi-clusters. In *International Conference on Parallel and Distributed Systems (ICPADS'06)*, pages 123–134, Los Alamitos, California. IEEE Computer Society.

[65] Karonis, N. T., Toonen, B. R., and Foster, I. T. (2003). MPICH-G2: A Grid-enabled implementation of the Message Passing Interface. *Journal of Parallel and Distributed Computing*, 63(5):551–563.

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

[67] Kaushik, N. R., Figueira, S. M., and Chiappari, S. A. (2006). 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'06)*, pages 218–225, Monterey, USA. IEEE Computer Society.

[68] Kim, J.-K., Shivle, S., Siegel, H. J., Maciejewski, A. A., Braun, T. D., Schneider, M., Tideman, S., Chitta, R., Dilmaghani, R. B., and Joshi, R. (2007). Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment. *Journal of Parallel and Distributed Computing*, 67(2):154–169.

[69] Kirley, M. and Stewart, R. (2007). Multiobjective evolutionary algorithms on complex networks. In *Proceedings of 4th International Conference Evolutionary Multi-Criterion Optimization (EMO'07), Lecture Notes Computer Science 4403*, Matsushima, Japan.

[70] Kuo, D. and Mckeown, M. (2005). Advance reservation and co-allocation protocol for grid computing. In Stockinger, H., Buyya, R., and Perrott, R., editors, *International Conference on e-Science and Grid Technologies (e-Science'05)*, pages 164–171, Los Alamitos, California. IEEE Computer Society.

[71] Kurp, P. (2008). Green computing. *Communications of the ACM*, 51(10):11–13.

[72] Lee, C. B., Schwartzman, Y., Hardy, J., and Snavely, A. (2004). Are user runtime estimates inherently inaccurate? In *Proceedings of the 10th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'04)*, volume 3277 of *Lecture Notes in Computer Science*, pages 253–263, New York, USA. Springer.

[73] Lee, C. B. and Snavely, A. (2006). 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.

[74] Lee, Y. C. and Zomaya, A. Y. (2007). Practical scheduling of bag-of-tasks applications on grids with dynamic resilience. *IEEE Transactions on Computers*, 56(6):815–825.

[75] Li, J. and Yahyapour, R. (2006). Negotiation model supporting co-allocation for grid scheduling. In Gannon, D., Badia, R. M., and Buyya, R., editors, *International Conference on Grid Computing (GRID'06)*, pages 254–261, Los Alamitos, California. IEEE Computer Society.

[76] Li, Z. and Parashar, M. (2006). A decentralized computational infrastructure for grid-based parallel asynchronous iterative applications. *Journal of Grid Computing*, 4(4):355–372.

[77] Lublin, U. and Feitelson, D. G. (2003). The workload on parallel supercomputers: modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing*, 63(11):1105–1122.

[78] Maclaren, J., Keown, M. M., and Pickles, S. (2006). Co-allocation, fault tolerance and grid computing. In Cox, S. J., editor, *UK e-Science All Hands Meeting (AHM'06)*, pages 155–162. NeSC Press.

[79] Maghraoui, K. E., Desell, T. J., Szymanski, B. K., and Varela, C. A. (2009). Malleable iterative MPI applications. *Concurrency and Computation: Practice and Experience*, 21(3):393–413.

[80] Maheswaran, M., Ali, S., Siegel, H. J., Hensgen, D. A., and Freund, R. F. (1999). Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 59(2):107–131.

[81] Milojičić, D. S., Douglis, F., Paindaveine, Y., Wheeler, R., and Zhou, S. (2000). Process migration. *ACM Computing Surveys*, 32(3):241–299.

[82] Mohamed, H. H. and Epema, D. H. J. (2004). An evaluation of the close-to-files processor and data co-allocation policy in multiclusters. In *International Conference on Cluster Computing (CLUSTER'04)*, pages 287–298, Los Alamitos, California. IEEE Computer Society.

[83] Mohamed, H. H. and Epema, D. H. J. (2005). Experiences with the koala co-allocating scheduler in multiclusters. In *Proceedings of the International Symposium on Cluster Computing and the Grid (CCGrid'05)*, pages 784–791, Los Alamitos, California. IEEE Computer Society.

[84] Mohamed, H. H. and Epema, D. H. J. (2008). KOALA: a co-allocating grid scheduler. *Concurrency and Computation: Practice and Experience*, 20(16):1851–1876.

[85] Mu'alem, A. W. and Feitelson, D. G. (2001). 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.

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

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

[88] Netto, M. A. S. and Buyya, R. (2007). 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 (CCGrid'07)*, Los Alamitos, California. IEEE Computer Society.

[89] Netto, M. A. S. and Buyya, R. (2008). Rescheduling co-allocation requests based on flexible advance reservations and processor remapping. In *Proceedings of the International Conference on Grid Computing (GRID'08)*, pages 144–151, Los Alamitos, California. IEEE Computer Society.

[90] Netto, M. A. S. and Buyya, R. (2010). *Handbook of Research on P2P and Grid Systems for Service-Oriented Computing: Models, Methodologies and Applications. Edited by Nick Antonopoulos and Georgios Exarchakos and Maozhen Li and Antonio Liotta*, chapter Resource Co-allocation in Grid Computing Environments. IGI Global publisher.

[91] Nurmi, D., Brevik, J., and Wolski, R. (2007). Qbets: Queue bounds estimation from time series. In *Proceeding of the 13th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'07)*, volume 4942 of *Lecture Notes in Computer Science*, pages 76–101. Springer.

[92] Pande, V. S., Baker, I., Chapman, J., Elmer, S., Larson, S. M., Rhee, Y. M., Shirts, M. R., Snow, C. D., Sorin, E. J., and Zagrovic, B. (2003). Atomistic protein folding simulations on the submillisecond time scale using worldwide distributed computing. *Peter Kollman Memorial Issue, Biopolymers*, 68(1):91–109.

[93] Pant, A. and Jafri, H. (2004). Communicating efficiently on cluster based grids with mpich-vmi. In *Proceedings of the International Conference on Cluster Computing (CLUSTER'04)*, pages 23–33, Los Alamitos, California. IEEE Computer Society.

[94] Park, J. (2004). A deadlock and livelock free protocol for decentralized internet resource coallocation. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 34(1):123–131.

[95] Parkhill, D. (1966). *The challenge of the computer utility*. Addison-Wesley Educational Publishers Inc., US.

[96] Popovici, F. I. and Wilkes, J. (2005). Profitable services in an uncertain world. In *Proceedings of the ACM/IEEE Conference on High Performance Networking and Computing (SC'05)*, Seattle, USA. IEEE Computer Society.

[97] Röblitz, T. and Reinefeld, A. (2005). Co-reservation with the concept of virtual resources. In *Proceedings of the International Symposium on Cluster Computing and the Grid (CCGrid'05)*, pages 398–406, Los Alamitos, California. IEEE Computer Society.

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

[99] Röblitz, T., Schintke, F., and Wendler, J. (2004). Elastic grid reservations with user-defined optimization policies. In *Proceedings of the Workshop on Adaptive Grid Middleware (AGridM'04)*, Los Alamitos, California. IEEE Computer Society.

[100] Rochwerger, B., Breitgand, D., Levy, E., Galis, A., Nagin, K., Llorente, I. M., Montero, R., Wolfsthal, Y., Elmroth, E., Caceres, J., Ben-Yehuda, M., Emmerich, W., and Galan, F. (2009). The reservoir model and architecture for open federated cloud computing. *IBM Journal of Research and Development*, 53(4).

[101] Romanazzi, G. and Jimack, P. K. (2008). Parallel performance prediction for numerical codes in a multi-cluster environment. In *Proceedings of the 2008 International Multiconference on Comp. Science and Information Technology (IMCSIT'08)*, Wisla, Poland.

[102] Sadjadi, S. M., Shimizu, S., Figueroa, J., Rangaswami, R., Delgado, J., Duran, H., and Collazo-Mojica, X. J. (2008). A modeling approach for estimating execution time of long-running scientific applications. In *Proceedings of the 22nd IEEE International Symposium on Parallel and Distributed Processing (IPDPS'08)*.

[103] Sanjay, H. A. and Vadhiyar, S. S. (2008). Performance modeling of parallel applications for grid scheduling. *Journal of Parallel and Distributed Computing*, 68(8):1135–1145.

[104] Shmueli, E. and Feitelson, D. G. (2005). Backfilling with lookahead to optimize the packing of parallel jobs. *Journal of Parallel Distributed Computing*, 65(9):1090–1107.

[105] Siddiqui, M., Villazón, A., and Fahringer, T. (2006). Grid capacity planning with negotiation-based advance reservation for optimized QoS. In *Proceedings of the ACM/IEEE Conference on High Performance Networking and Computing (SC'06)*, Tampa, USA. ACM Press.

[106] Sievert, O. and Casanova, H. (2004). A simple MPI process swapping architecture for iterative applications. *International Journal of High Performance Computing Applications*, 18(3):341–352.

[107] Sim, K. M. (2007). Relaxed-criteria G-negotiation for grid resource co-allocation. *ACM SIGecom Exchanges*, 6(2):37–46.

[108] Sinaga, J. M. P., Mohamed, H. H., and Epema, D. H. J. (2004). A dynamic co-allocation service in multicluster systems. In Feitelson, D. G., Rudolph, L., and Schwiegelshohn, U., editors, *International Workshop Job Scheduling Strategies for Parallel Processing*, volume 3277 of *Lecture Notes in Computer Science*, pages 194–209, New York, USA. Springer.

[109] Singh, G., Kesselman, C., and Deelman, E. (2007). A provisioning model and its comparison with best-effort for performance-cost optimization in grids. In *Proceedings of the 16th International Symposium on High-Performance Distributed Computing (HPDC'07)*, pages 117–126, Monterey, USA. ACM.

[110] Smallen, S., Casanova, H., and Berman, F. (2001). Applying scheduling and tuning to on-line parallel tomography. In *Proceedings of the ACM/IEEE Conference on Supercomputing (SC'01)*, Denver, USA. ACM.

[111] Smith, W., Taylor, V. E., and Foster, I. T. (1999). Using run-time predictions to estimate queue wait times and improve scheduler performance. In *Proceeding of the International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'99)*, volume 1659 of *Lecture Notes in Computer Science*, pages 202–219. Springer.

[112] Snell, Q., Clement, M. J., Jackson, D. B., and Gregory, C. (2000). The performance impact of advance reservation meta-scheduling. In Feitelson, D. G. and Rudolph, L., editors, *Proceedings of the International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'00)*, volume 1911 of *Lecture Notes in Computer Science*, pages 137–153, Berlin. Springer.

[113] Sonmez, O., Mohamed, H., and Epema, D. (2006). Communication-aware job placement policies for the KOALA grid scheduler. In *Proceedings of the International Conference on e-Science and Grid Computing (e-Science'06)*, page 79, Los Alamitos, California. IEEE Computer Science.

[114] Srinivasan, S., Subramani, V., Kettimuthu, R., Holenarsipur, P., and Sadayappan, P. (2002). Effective selection of partition sizes for moldable scheduling of parallel jobs. In *Proceedings of the 9th International Conference on High Performance Computing (HiPC'02)*, volume 2552 of *Lecture Notes in Computer Science*, pages 174–183. Springer.

[115] Takefusa, A., Nakada, H., Kudoh, T., Tanaka, Y., and Sekiguchi, S. (2007). GridARS: an advance reservation-based grid co-allocation framework for distributed computing and network resources. In Frachtenberg, E. and Schwiegelshohn, U., editors, *Proceedings of the International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'07)*, Lecture Notes in Computer Science, Berlin. Springer.

[116] Takemiya, H., Tanaka, Y., Sekiguchi, S., Ogata, S., Kalia, R. K., Nakano, A., and Vashishta, P. (2006). Sustainable adaptive grid supercomputing: multiscale simulation of semiconductor processing across the pacific. In *Proceedings of the Conference on High Performance Networking and Computing (SC'06)*, page 106, New York, USA. ACM Press.

[117] Tanenbaum, A. S. and Steen, M. V. (2001). *Distributed Systems: Principles and Paradigms*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

[118] Tsafrir, D., Etsion, Y., and Feitelson, D. G. (2005). Modeling user runtime estimates. In *Proceedings of the 11th International Workshop Job Scheduling Strategies for Parallel Processing (JSSPP'05)*, volume 3834 of *Lecture Notes in Computer Science*, pages 1–35. Springer.

[119] Tsafrir, D., Etsion, Y., and Feitelson, D. G. (2007). Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Transactions on Parallel and Distributed Systems*, 18(6):789–803.

[120] Tsafrir, D. and Feitelson, D. G. (2006). The dynamics of backfilling: Solving the mystery of why increased inaccuracy may help. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC'06)*, San Jose,USA.

[121] Vazhkudai, S. (2003). Enabling the co-allocation of grid data transfers. In Werner, B., editor, *Proceedings of the International Workshop on Grid Computing (GRID'03)*, pages 44–51, Los Alamitos, California. IEEE Computer Society.

[122] Vecchiola, C., Kirley, M., and Buyya, R. (2009). Multi-objective problem solving with offspring on enterprise clouds. In *Proceedings of the 10th International Conference on High-Performance Computing in Asia-Pacific Region (HPC Asia'09)*.

[123] Venugopal, S., Buyya, R., and Ramamohanarao, K. (2006). A taxonomy of data grids for distributed data sharing, management, and processing. *ACM Computing Surveys*, 38(1).

[124] Vieira, G. E., Herrmann, J. W., and Lin, E. (2003). Rescheduling manufacturing systems: A framework of strategies, policies, and methods. *Journal of Scheduling*, 6(1):39–62.

[125] Viswanathan, S., Veeravalli, B., and Robertazzi, T. G. (2007). Resource-aware distributed scheduling strategies for large-scale computational cluster/grid systems. *IEEE Transactions on Parallel and Distributed Systems*, 18(10):1450–1461.

[126] Wilkinson, B. and Allen, M. (1998). *Parallel programming: techniques and applications using networked workstations and parallel computers*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA.

[127] Wu, Y.-L., Huang, W., Lau, S.-C., Wong, C. K., and Young, G. H. (2002). An effective quasi-human based heuristic for solving the rectangle packing problem. *European Journal of Operational Research*, 141(2):341–358.

[128] Xu, D., Nahrstedt, K., and Wichadakul, D. (2001). QoS and contention-aware multi-resource reservation. *Cluster Computing*, 4(2):95–107.

[129] Yang, C.-T., Yang, I.-H., Wang, S.-Y., Hsu, C.-H., and Li, K.-C. (2007). A recursively-adjusting co-allocation scheme with cyber-transformer in data grids. *Future Generation Computer Systems*.

[130] Yang, L. T., Ma, X., and Mueller, F. (2005). Cross-platform performance prediction of parallel applications using partial execution. In *Proceedings of the ACM/IEEE Conference on High Performance Networking and Computing (SC'05)*.

[131] Yeo, C. S. and Buyya, R. (2007a). Integrated risk analysis for a commercial computing service. In *Proceedings of the 21th International Parallel and Distributed Processing Symposium (IPDPS'07)*, pages 1–10, Long Beach, USA. IEEE.

[132] Yeo, C. S. and Buyya, R. (2007b). Pricing for utility-driven resource management and allocation in clusters. *International Journal of High Performance Computing Applications*, 21(4):405.

[133] Yoshimoto, K., Kovatch, P. A., and Andrews, P. (2005). Co-scheduling with user-settable reservations. In Feitelson, D. G., Frachtenberg, E., Rudolph, L., and Schwiegelshohn, U., editors, *Proceedings of the International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'05)*, volume 3834 of *Lecture Notes in Computer Science*, pages 146–156, Berlin. Springer.

[134] Yu, J. and Buyya, R. (2005). A taxonomy of workflow management systems for grid computing. *Journal of Grid Computing*, 3(3-4):171–200.

[135] Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., and da Fonseca, V. G. (2003). Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132.