Provisioning Techniques and Policies for Resource Sharing between Grids

by

Marcos Dias de Assunção

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

March 2009

Provisioning Techniques and Policies for Resource Sharing between Grids

Marcos Dias de Assunção

Supervisors: Assoc. Prof. Rajkumar Buyya and Dr. Srikumar Venugopal

---

Abstract

Grids enable sharing, selection and aggregation of computational resources, storage resources, and scientific instruments belonging to multiple institutions. Scientific collaborations routinely choose Grids to fulfil heavy computational requirements. Recent evidence, however shows that most Grids work in isolation, and with different utilisation levels.

This thesis explores mechanisms and technologies for enabling Grids to share resources. It surveys key concepts and systems used to enable resource sharing among physical organisations, and classifies existing systems considering their architectures, operational models, resource control techniques, support for virtual organisations, and support for inter-Grid operation.

We present an architecture for enabling resource sharing between Grids, based on the idea of peering arrangements between gateways that mediate access to the resources of interconnected Grids. The gateways rely on resource-availability information obtained from resource providers participating in the interconnected Grids; providers have their local users, yet agree to provide resources to the Grid. We thus investigate site-level provisioning techniques that enable providers to satisfy the requirements of their local users and offer resources to a gateway, whereby the gateway can provision resources to Grid applications and avoid contention. Simulation results show that for a Grid such as DAS-2, a very small part of the jobs scheduled face resource contention (*i.e.* less than 0.5%), if providers inform the gateway about the resource availability every 15 to 30 minutes and use a provisioning strategy based on conservative backfilling.

Moreover, this thesis describes a mechanism by which gateways can share resources under peak load conditions. The acceptance and redirection of requests is based on the marginal cost of allocation. The mechanism is effective in balancing the load across interconnected Grids and improves the response time of Grid applications.

This thesis also extends current provisioning techniques by presenting a model where organisations can use resources from commercial providers to augment the capacity of their clusters. This thesis proposes various strategies to utilise the resources from commercial providers, and quantifies the cost of achieving performance improvements. A strategy based on selective backfilling using resources from commercial providers yields the best ratio of slowdown improvement to money spent.

The proposed resource provisioning strategies and the load sharing mechanism are evaluated using discrete-event simulation. This thesis also realises the proposed architecture using virtual machines to accommodate user requests; resources allocated by one gateway from another are used to run virtual machines.

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 supervisors, Associate Professor Rajkumar Buyya and Dr. Srikumar Venugopal, for their advice and guidance throughout my candidature. While a PhD candidature is a great experience for working on stimulating topics and challenging problems, most importantly, it is a wonderful opportunity to meet and collaborate with extraordinary people. This research environment would not be possible without the determination of my supervisors who have helped create the best study conditions.

I am also immensely grateful to my parents. They have provided me with lessons on honesty and ethics that no university could ever rival, and their humbleness and patience have always amazed me. Thanks also to my brothers and my sister for their lifetime love, companionship, and support.

To Cynthia Fantoni, my heartfelt thanks for her patience, constructive comments, and for providing me with encouragement during rough times. She always reminded me that there is light at the end of the tunnel. If there is, I will soon figure it out.

I would also like to thank all past and current members of the GRIDS Laboratory, at the University of Melbourne. In particular, I want to thank Marco Netto, Alexandre di Costanzo, Krishna Nadiminti, Hussein Gibbins, Mukaddim Pathan, Sungjin Choi, Suraj Pandey, Chee Shin Yeo, Kyong Hoon Kim, Christian Vecchiola, Jia Yu, Saurabh Garg, William Voorsluys, Mohsen Amini, Rajiv Ranjan, and Mustafizur Rahman for their help and comments on my work. I also extend my gratitude to the staff and other friends from the CSSE Department for their help and support.

I would like to express gratitude to Professor Torsten Eymann, Werner Streitberger, Vanessa Teague, Fernando Koch, Carlos Becker Westphall, Cecilia Xia, and Rodrigo Calheiros. It has been a pleasure to exchange ideas and work with you. I am grateful to Professor Carlos Varela for his valuable comments on a preliminary version of this thesis.

A special thanks to Anne Wight, Una Wright, Daniela Silva, Carolyne Njihia, and Taísia Autran for the friendship and help during the difficult times. I also thank all my friends and everybody that contributed to this research in any form.

I also express gratitude to National ICT Australia (NICTA), the Australian Research Council (ARC) and Australian Department of Innovation, Industry, Science and Research (DIISR). Without their support, I would not have been able to carry out this research. I also thank the University of Melbourne for providing me with an IPRS scholarship, an office, and computer facilities during the course of my PhD.

Marcos Dias de Assunção
Melbourne, Australia
March 2009.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

Advances in distributed computing have enabled the creation of national and international Grids such as the TeraGrid [36, 55], Open Science Grid [144], Enabling Grids for E-sciencE (EGEE) [65], APACGrid in Australia [134], K*Grid in Korea [139], NAREGI in Japan [124], Garuda in India [145], E-Science Grid in the UK [96], OurGrid in Brazil [8], Grid'5000 [21] and AuverGrid [106] in France, and DAS in the Netherlands [43]. Composed of multiple resource providers, these Grids enable collaborative work and sharing of computational resources, storage resources, and scientific instruments among groups of individuals and organisations. These collaborations have been known as Virtual Organisations (VOs) [82].

For over a decade various technologies have enabled applications to be deployed on these Grids, including Grid middleware such as Globus [79], Legion [37], UNICORE [3], and gLite [57]; schedulers such as Application Level Schedulers (AppLeS) [18]; and resource brokers including Gridbus Resource Broker [186], Nimrod/G [30], Condor-G [84], and GridWay [98].

Although these Grids have contributed to various sciences and disciplines, evidence shows that they mostly work in isolation and with different utilisation levels [101]. Current Grids are like "islands" with little or no resource sharing between them; a situation starkly contrasted with the original vision of Grid computing, which imagined a single global infrastructure providing users with computing power on demand [80]. Efforts to address this issue include providing interoperability among different Grids and Grid middleware [89], and creating trust federations between Grids to grant users in one Grid easy access to another.

Despite these interoperability efforts, enabling Grids to share their resources remains a virtually untouched challenge. Boghosian *et al.* [20] highlight the lack of mechanisms to enable the allocation or provisioning of resources from multiple Grids to applications. They also describe three sample applications whose performance would significantly improve if resources from multiple Grids were utilised. Studying the utilisation logs of the DAS-2 [43] in the Netherlands and Grid'5000 [21] in France, Iosup *et al.* [101] showed that there was a significant load imbalance between these Grids. Moreover, Orgerie *et al.* [136] studied the power consumption of the Grid'5000 computing sites and showed that the efficient management of resources can result in significant electricity savings. In light of this evidence, it would seem that interconnecting and enabling resource sharing between Grids offers great opportunities and benefits. However, enabling resource shar-

ing between Grids is a complex task. Each Grid's autonomy for capacity planning and provisioning of resources to user communities exacerbates contention for resources and dynamic demand within participant Grids.

This thesis provides an architecture for interconnecting Grids, and explores mechanisms and techniques to allow sharing of resources between Grids; focusing on computing resources. The remaining part of this chapter elaborates the need for interconnecting Grids, and describes the research problem, the objectives of this thesis, its contributions, and its organisation.

1.1 Motivations for Interconnecting Grids

The ultimate goal of Grid computing is to enable the creation of an infrastructure that allows scientists and practitioners to cope with the scale and complexity of both current and next-generation scientific challenges [55, 64, 83]. To date, various national programs have initiated e-Science projects to enable resource-sharing and collaboration among scientists.

Such Grids, however, generally follow restricted organisational models, wherein a VO is created for a specific collaboration and all interactions and resource-sharing are limited to within the VO. Further, an infrastructure is often set up to realise the technical requirements of a particular user community, which can differ considerably from the requirements of another community. Thus, these factors have generally resulted in dispersed Grid initiatives and the creation of disparate Grids with little or no resource sharing between them. The bottom layer in Figure 1.1 illustrates this scenario of dispersed Grids around the world. Resource sharing between Grids is needed to provide users with a larger infrastructure upon which they can deploy applications that adapt, by growing and shrinking in terms of resource consumption as depicted by the top layer in Figure 1.1.

Grids could enable these applications just as execution environments are enabled in other large-scale testbeds. Large-scale testbeds such as PlanetLab [141] present an interesting model for providing "slices" of resources to users, so they can deploy their applications. In turn, these applications can execute in isolation from other applications running on the same physical infrastructure. PlanetLab achieves this by allocating resources in a centralised manner and performing resource control using the concept of virtual resources or virtual machines. Grid computing, however, generally carries out resource control on a job basis, wherein users encapsulate their applications as jobs that are routed to the resources where they are executed. Moreover, Grids have autonomy to implement and enforce their own resource usage policies.

In this thesis, we argue that an architecture and mechanisms based on the idea of peering arrangements would enable resource provisioning and sharing between Grids. Internet Service Providers (ISPs) establish peering agreements to allow traffic into one another's networks [12, 123, 132]. The middle layer of Figure 1.1 illustrates the idea of resource provisioning between Grids through peering arrangements. However, as described earlier, some of the issues involved in providing resources from multiple Grids to applications arise from different Grids, and usually parts of a Grid, operating under different administrative domains. Moreover, in contrast to other large-scale testbeds, Grids are composed of autonomous resource providers, which are part of a Grid but also need to respect the resource demands of their local users.

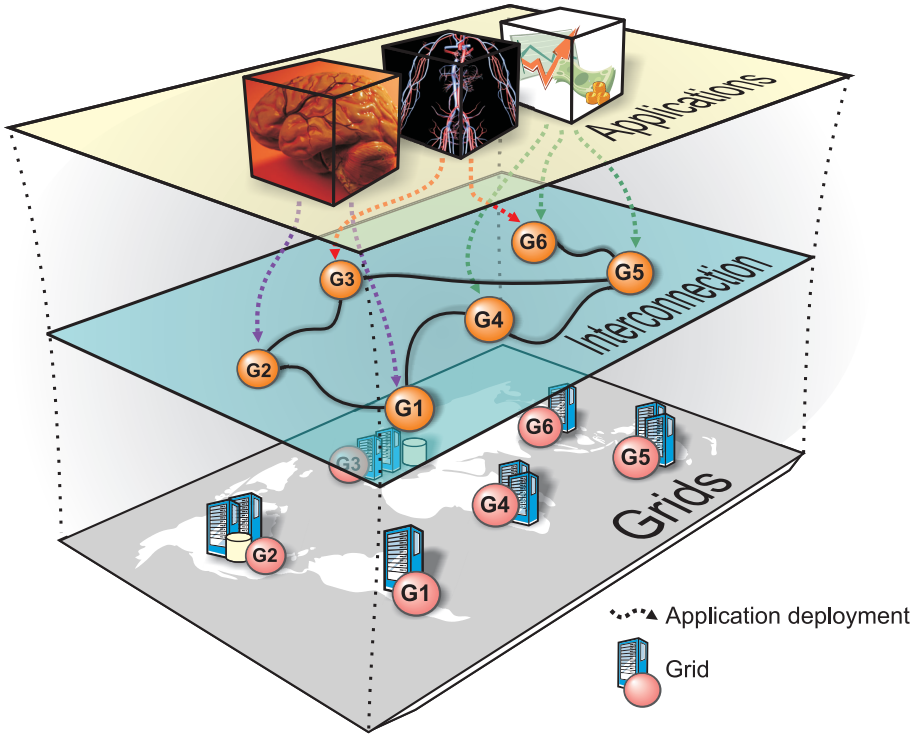


Figure 1.1: Abstract view of the interconnection of Grids.

1.2 Research Problem and Objectives

This thesis tackles the challenge of how Grids can share resources, thus increasing their overall capacity and providing benefits to their users, without disrespecting the demands of the involved providers' local users.

Towards that end, this thesis investigates mechanisms for resource sharing between Grids to enable deploying applications across Grids, recognising that Grid users generally contend for resources[1] with the provider's local users. Resource sharing between Grids introduces another level of contention: between users local to the interconnected Grids and users external to those Grids.

Mechanisms for resource sharing between Grids should provide techniques to minimise the impact of these contentions. To achieve that, this thesis investigates strategies that providers can use to allocate resources to Grids, while providing guarantees over access to resources further shared by the interconnected Grids.

1.2.1 Objectives

Based on the challenge described above, we delineated the following objectives:

1. Provide an architectural model that supports the interconnection of Grids and resource sharing between interconnected Grids.

[1]This thesis uses the term contention to represent a situation where two users try to use a particular resource over the same period.

2. Investigate resource provisioning techniques to deal with the resource contention that can arise within the interconnected Grids.

3. Investigate resource sharing mechanisms between Grids that benefit the interconnected Grids, such as improve performance of Grid applications.

1.3 Methodology

To evaluate the mechanisms proposed in this thesis, we perform discrete-event simulations. Simulations provide a controllable environment for repeatable experiments. Moreover, as the research community uses Grids as production environments to run their applications or as testbeds, it would be difficult to *actually* vary the computing resources' configuration to experiment with different provisioning mechanisms, even if system administrators could be induced to relinquish such privileges.

We realised the proposed architecture by designing a system prototype that uses virtual machines to accommodate user requests [15]. This thesis describes the system that enables the allocation of virtual machines from multiple computing sites, or Grids. The methodology used to investigate each proposed mechanism is described in the respective chapters.

1.4 Contributions

Based on the objectives defined above, this thesis makes the following contributions:

1. It introduces a provisioning architecture that enables resource sharing between Grids, and deploying applications across Grids. The proposed architecture uses peering arrangements between gateways that mediate access to the resources of interconnected Grids; offering a framework for resource provisioning within Grids.

2. It investigates several site-level provisioning strategies for allocating resources to intra-Grid applications. Providers have their local users, yet agree to provide resources to the Grid. Site-level provisioning aims to minimise contentions between providers' local users and Grid users.

   Experimental results show that for a Grid similar to DAS-2 [43], a strategy based on conservative backfilling should suffice to provide resources to the Grid. Based on multiple resource partitions and load estimates, the proposed strategy shows improved job slowdown in scenarios where the providers inform the gateway about the availability at long intervals (*i.e.* a few hours). Thus, the multiple resource partitions are a better option if long intervals are considered to gather the resource availability required for provisioning decisions.

3. It presents a mechanism for resource sharing between Grids. According to the proposed mechanism, Grids can share resources under peak load conditions. Grids have pre-defined contracts that specify the terms of the interconnection, such as the price paid for allocating resources from one another. The acceptance and redirection of requests takes into account the marginal cost of resource allocation. This

load-sharing mechanism relies on resource-availability information provided by the site-level provisioning techniques. Specifically, we select three provisioning strategies, namely the single-partition with conservative backfilling [128], the multiple-partition strategy with conservative backfilling and the multiple-partition strategy with load estimates.

Simulation results show that the proposed mechanism effectively balances the load among the interconnected Grids. Results also demonstrate that interconnected Grids benefit one another, as the Average Weighted Response Time (AWRT)[2] improves considerably for both Grid users' requests and requests from the providers' local users. Even stand-alone Grids showed improvements in AWRT for both Grid and local jobs using a site-level provisioning strategy based on multiple resource partitions and conservative backfilling. In some scenarios, the time for Grid resource exchange negotiations impacts the AWRT of Grids with the lightest load. These Grids face small increases in job response time, which are introduced by the negotiation mechanism.

4. Availability of virtual machine technologies has increased in recent years, offering benefits such as performance isolation, environment isolation, and server consolidation. The proposed architecture has been realised using virtual machine systems to accommodate user requests. Resources from one Grid, allocated by another Grid, are used to run virtual machines, and the system allows deploying virtual machines on commercial resource providers, who use virtualisation solutions to manage their infrastructure.

5. The maturity of virtual machine and network technologies has led to the emergence of commercial infrastructure providers, who offer virtually unlimited resources to end-users on a pay-as-you-go basis. This model is generally termed as "Cloud Computing" [9, 193] as the resources are on a "Cloud" whose physical infrastructure is unknown to the users. The emergence of these commercial infrastructure providers, their economies of scale, and the increasing costs of operating a Grid infrastructure may contribute to Grids comprising commercial and non-commercial infrastructures [48].

   This thesis investigates a scenario where the capacity of an organisation's cluster is extended by leasing Grid resources from a commercial provider. It proposes and evaluates various scheduling strategies that consider the use of resources from commercial providers, in order to understand how these strategies achieve a balance between performance and usage cost, and how much they improve the requests' response times. We evaluate the cost of performance improvements, given the money spent for using commercial resources. Simulation results show that different strategies can yield different ratios of response time improvement to money spent. A strategy based on selective backfilling [172] yields the best ratio of slowdown improvement to money spent. A deadline-aware strategy based on conservative backfilling [128] is the preferable option when resources from the commercial provider are used to minimise rejected requests.

[2]The weight is given by the job's resource consumption. This metric is explained in Chapter 5.

1.5 Thesis Organisation

The core chapters of this thesis derive from various research papers published during the course of the PhD candidature. The interrelationship among the chapters and the fields to which they are related are shown in Figure 1.2, and the remaining part of this thesis is organised as follows:

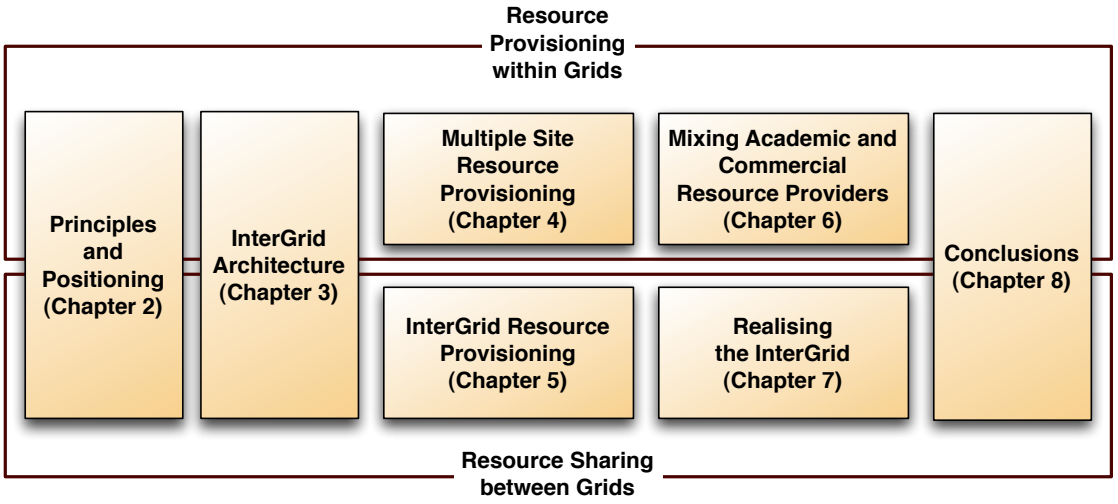


Figure 1.2: Organisation of this thesis.

- Chapter 2 presents background and the literature review on systems and mechanisms used by Grids to enable resource sharing among participating organisations. It therefore positions the thesis in regards to existing work. Chapter 2 derives from:
  - **M. D. de Assunção** and R. Buyya, Architectural Elements of Resource Sharing Networks, The Handbook of Research on Scalable Computing Technologies, IGI Global, 2009 (in print).
- The proposed architecture used for resource sharing between Grids is described in Chapter 3, using the concept of peering arrangements between Grids. Chapter 3 is partially derived from:
  - **M. D. de Assunção**, R. Buyya and S. Venugopal, InterGrid: A Case for Internetworking Islands of Grids, *Concurrency and Computation: Practice and Experience (CCPE)*, Vol. 20, Issue 8, pp. 997-1024, Wiley Press, New York, USA, June 2008.
- Chapter 4 describes site-level provisioning strategies and accompanying availability information used to provide resources to Grid applications. Chapter 4 derives from:
  - **M. D. de Assunção**, W. Streitberger, T. Eymann, and R. Buyya, Enabling the Simulation of Service-Oriented Computing and Provisioning Policies for Autonomic Utility Grids, Proceedings of the *4th International Workshop on Grid Economics and Business Models (GECON 2007)*, LNCS, Vol. 4685, Springer-Verlag, Berlin, Germany), Aug. 28, 2007, Rennes, France.

  - **M. D. de Assunção** and R. Buyya, Performance Analysis of Multiple Site Resource Provisioning: Effects of the Precision of Availability Information, Proceedings of the *15th International Conference on High Performance Computing (HiPC 2008)*, pp. 157-168, Bangalore, India, 17-20 Dec. 2008.

- A mechanism for resource sharing between Grids is proposed in Chapter 5. This mechanism is based on requests' marginal cost of allocation and uses the site-level provisioning techniques proposed in Chapter 4. Chapter 5 is derived from:

  - **M. D. de Assunção** and R. Buyya, A Cost-Aware Resource Exchange Mechanism for Load Management across Grids, Proceedings of the *14th International Conference on Parallel and Distributed Systems (ICPADS 2008)*, pp. 213-220, Melbourne, Australia, 8-10 Dec. 2008.
  - **M. D. de Assunção** and R. Buyya, Performance Analysis of Allocation Policies for InterGrid Resource Provisioning, *Information and Software Technology*, Elsevier, Vol. 51, Issue 1, pp. 42-55, Jan. 2009.

- Chapter 6 describes and evaluates scheduling strategies that enable the use of commercial providers to extend the capacity of academic or non-commercial clusters using the architecture described in Chapter 3. Chapter 6 derives from:

  - **M. D. de Assunção**, A. Di Costanzo, and R. Buyya, Evaluating the Cost-Benefit of Using Cloud Computing to Extend the Capacity of Clusters, Proceedings of the *International Symposium on High Performance Distributed Computing (HPDC 2009)*, Munich, Germany, 11-13 Jun. 2009.

- Chapter 7 describes how the architecture proposed in Chapter 3 has been realised. We detail the system implementation and how it uses virtual machine systems to accomplish the resource provisioning goals. Both the simulation models and the real system use a data structure to store the resource availability information. Details on the data structure used to carry out request admission control and to store the resource availability information are given in Appendix A.

- We present general considerations, conclusions and future directions in Chapter 8.

Chapter 2

Principles and Positioning

This chapter discusses approaches for enabling resource allocation across Grids and other large-scale testbeds. Focusing on allocation of computational resources, the chapter critiques existing systems according to their architectures, operational models, support for the life cycle of Virtual Organisations (VOs), and techniques for resource control. We also present key background information to facilitate a better understanding of the topics addressed in the remaining chapters, and position the thesis in regards to related work.

2.1 Grid Computing

Grids enable sharing, selection and aggregation of computational resources, storage resources, and scientific instruments belonging to multiple institutions [80]. Several scientific collaborations have used Grid infrastructure for their heavy computational requirements. A layered view of the Grid architecture is depicted in Figure 2.1.

It is worth noting that although Grids can comprise various types of resources, this thesis focuses on computational resources, hence a *Grid is viewed as a collection of clusters* running local resource management systems.

From bottom to top, the key layers and components of the depicted architecture are:

1. **Grid Fabric/Distributed Resources:** this layer represents resources distributed in the Grid: computers, clusters of computers, databases, sensors, and scientific instruments. As described in greater detail later, clusters can run a variety of *local resource management systems* such as Condor [117], the Portable Batch System (PBS) [188], and Sun Grid Engine (SGE) [25].

2. **Core Grid Middleware:** this layer offers core services such as remote process management, resource directories and discovery, security, and resource reservation. The Grid middleware also aims to provide standard interfaces to utilise underlying Grid resources [37, 79].

3. **Grid Resource Management Systems:** this layer contains *high-level or user Grid middleware*, including programming models, development environments, resource brokers, and meta-schedulers for managing resources and scheduling application

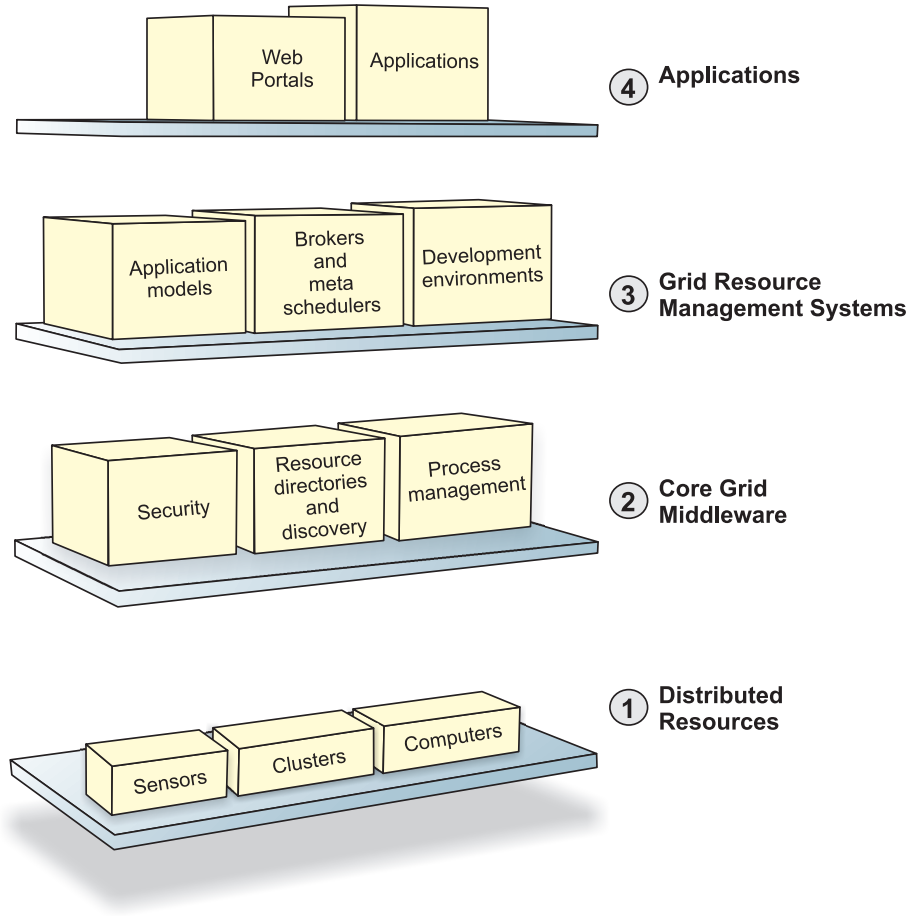


Figure 2.1: Layered view of Grid architecture.

jobs for execution on Grid resources. The meta-schedulers utilise the local resource management systems via the interfaces provided by the *Core Grid middleware*.

4. **Applications:** comprises applications,[1] and Web portals that allow users to execute and monitor their experiments.

2.2 Job Scheduling in Clusters

User applications executed on clusters of computers or supercomputers are typically encapsulated as batch jobs that require one or more system processors. Deciding where and when to run the users' jobs is termed as *scheduling*, and the entity responsible for scheduling jobs is commonly termed the *batch scheduler*, or simply *cluster scheduler*. Examples of cluster schedulers include PBS [188], SGE [25], the MAUI scheduler [104], and OAR [32].

This thesis uses the term Local Resource Management System (LRMS) to refer to a cluster scheduler. The rest of this section describes job backfilling schemes, multiple

[1]These applications are typically developed using the Grid programming models provided by the high-level Grid middleware

resource partitions, and advance reservations.

2.2.1 Job Backfilling

Job scheduling in a cluster can be viewed in terms of a 2D chart with time along the *x* axis and number of processors along the *y* axis (Figure 2.2). Each job is a rectangle whose length is the job's runtime and height is the number of processors required. This thesis uses this simple representation to illustrate the proposed provisioning schemes. One way of scheduling jobs that arrive in the system is to use a First Come First Served (FCFS) policy, however this approach can lead to low resource utilisation.

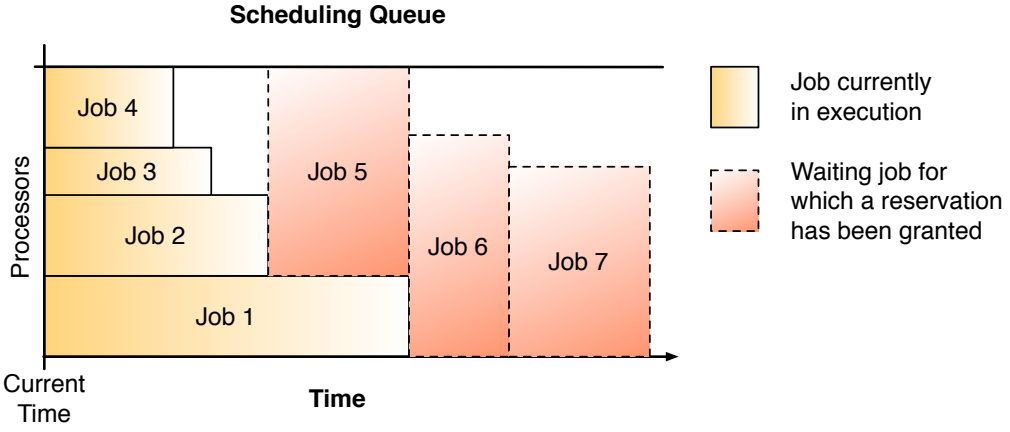


Figure 2.2: Graphical representation of a scheduling queue.

Current LRMSs generally use optimisations to the FCFS policy, such as job backfilling [116, 128]. Jobs are queued for execution in the order they arrive. Backfilling involves slotting in later jobs (*i.e.* further back in the queue) to use available resources without delaying other waiting jobs. This technique improves job response time, maximises resource utilisation, and reduces scheduling queue fragmentation. Extensive literature exists on job backfilling [114, 116, 128, 162, 172, 180, 182]. The next sections describe relevant job backfilling techniques.

Conservative Backfilling

A common term used for a scheduled job under backfilling is that of a reservation. A job is granted a reservation when its start time is defined (*i.e.* a place in the 2D chart has been found). Under conservative backfilling, a job can be brought forward and execute earlier if it does not delay any other job in the waiting queue [128].

Aggressive Backfilling

Aggressive backfilling, also known as EASY backfilling, brings forward a job and starts execution if it does not delay only the first job in the waiting queue – termed *pivot job*. That is, only the job at the head of the waiting queue is granted a reservation [116]. Under aggressive backfilling, the schedule contains the expected completion of running jobs and the start time of the pivot job only. Some schedulers, such as Maui [104], allow the

system administrator to configure the maximum number of pivots, which in turn enables the scheduler to maintain the start and expected completion times of up to the maximum number of pivot jobs [104]. For example, if the maximum number of pivots is set to 5, and there are 5 jobs waiting in the queue, a 6^{th} job that just arrived is used to backfill only if it does not delay any of the 5 pivot jobs. If the maximum number of pivots is very large, the scheduler becomes conservative backfilling.

Selective Backfilling

Selective backfilling grants reservations to jobs that have waited *long enough* in the queue [172]. A threshold parameter computed as the result of a response time related metric, such as job slowdown [74],[2] generally defines how long a job should wait in the queue.

2.2.2 Multiple Resource Partitions

Work on multiple resource partitions and priority scheduling has shown to reduce the job slowdown compared to scheduling based on aggressive backfilling [114]. Under this scenario, the scheduler divides the resources available into partitions and assigns jobs to these partitions according to criteria specified by the system administrator. A partition can borrow resources from another when the latter is not using them, and the scheduler allows such borrowing. If aggressive backfilling is used, for example, each partition performs the scheduling and has its own pivot job. One partition *A* can borrow resources from another partition *B* if doing so will not delay the pivot job of *B*.

2.2.3 Advance Reservations

With the traditional FCFS policy and backfilling strategies, the start time of a job depends on the cluster's workload and is not under the user's control. However, users in general appreciate guarantees of resource availability to ensure their jobs meet deadlines; severe weather forecasting is an example of deadline-constrained application. Advance reservation mechanisms can provide guarantees about the application start and completion times as they allow users to request resources at specific times [169, 170]. A provisioning method based on reservations offers the guarantees required for deadline-constrained applications [167]. LRMSs such as LSF, PBS Pro and Maui can support advance reservation requests [120].

However, Smith *et al.* [169] and Margo *et al.* [121] showed that the waiting time of normal workload increases when reservations are allowed and the increase depends on how reservations are supported. The use of advance reservations can also lead to a scenario of low resource utilisation if users do not use the resources they reserve. A study by Snell *et al.* [170] on the impact of advance reservations meta-scheduling produced similar findings; the resource utilisation decreases as the number of reservations for jobs from the meta-scheduler increases.

Some techniques can minimise the impact of advance reservations on normal workload. These techniques include:

[2] Slowdown is the ratio of the response time on a loaded system to the response time on a dedicated system.

- limiting the number of reservations allowed in the system [121];
- increasing the laxity of advance reservation requests, where laxity is the difference between a request's deadline and the time the request would finish executing if it started at its start time [71];
- enabling flexible reservation requests that are elastic in properties such as start-time, duration, and number of processors required [62, 130, 153, 154, 174]; and
- overbooking resources [175, 183].

Guarantees are also essential to jobs that require co-allocation of resources from multiple clusters. Several solutions for co-allocating Grid resources have been proposed in the literature. The Globus Architecture for Reservation and Allocation (GARA) [81] extended a previously proposed co-allocation architecture [41] by introducing agents responsible for discovering and reserving a set of resources that can satisfy the user's requirements. A co-reservation request returns a handle that is then passed to a co-allocation agent in charge of allocating the reserved resources. Anand *et al.* [5] proposed a multi-stage co-allocation strategy based on resource hierarchies. The strategy can separately handle small co-allocation requests at lower levels of the hierarchy, thus not all resources need to be available at the time a job is executed. In addition, meta-schedulers and brokers supporting co-allocation of resources have been designed [62, 125, 189]. Although co-allocation of resources may be required by some Grid applications and can bring benefits, we do not aim to address co-allocation issues in this thesis.

2.3 Job Scheduling in Grids

Several of the organisational models used within existing Grids are based on the idea of VOs. The VO scenario is characterised by resource providers offering different shares of resources to different VOs via an agreement or contract; these shares are further aggregated and allocated to users and groups within each VO on a needs basis. The problem of managing resources within VOs is further complicated by the fact that resource control is commonly performed on a job basis. Providers generally contribute clusters of computers managed by LRMSs, such as those described in Section 2.2. Grid Resource Management Systems (GRMSs) schedule Grid users' applications and allocate the resources contributed by providers. A GRMS comprises components such as:

- Meta-schedulers, which communicate with LRMSs to place jobs at the provider sites;
- VO-schedulers that allocate resources considering how providers and users are organised in virtual organisations [52]; and
- Resource brokers, which represent users or organisations, and schedule and manage job execution on their behalf.

These components interact with providers' LRMSs either directly or via interfaces provided by the Grid middleware. The Grid schedulers can communicate with one another in various ways, such as via sharing agreements, hierarchical scheduling, and Peer-to-Peer (P2P) networks. We discuss GRMSs in Section 2.4 and classify existing systems according to their support for the life cycle of VOs, their resource control techniques, and the mechanisms for inter-operation with other systems.

2.4 Grid Resource Management Systems

Previous work has classified systems according to their architectures, operational models and scheduling techniques [29, 35, 101]. We extend these classifications by including a new operational model. Moreover, systems with similar architecture can still differ in terms of the mechanisms employed for resource sharing, the self-interest of the system's participants, and the communication model.

A Grid system can use decentralised scheduling wherein schedulers communicate their decisions with one another in a co-operative manner, thus guaranteeing the maximum global utility of the system. Alternatively, a scheduler/broker may represent a particular user community within the Grid, can have contracts with other schedulers in order to use the resources they control, and allocate resources that maximise its own utility, or achieved profit. In this section we classify the arrangements between schedulers. Furthermore, systems can also differ with regard to their resource control techniques and support for different stages of the VO life cycle; therefore this section also classifies resource control techniques. Section 2.5 discusses the systems' support for virtual organisations. The attributes of GRMSs and the taxonomy are summarised in Figure 2.3.

2.4.1 Architecture and Operational Models of GRMSs

Grid systems can have schedulers and brokers organised in various manners. Classification by architectural model is summarised in Table 2.1. Iosup *et al.* [101] and Buyya *et al.* [29] considered a multiple cluster scenario and classified the architectures possibly used as GRMSs in the following categories:

- **Independent clusters:** each cluster has its LRMS and there is no meta-scheduler component, *i.e.* users submit their jobs to the clusters of the organisations to which they belong or on which they have accounts. We extend this category by including single-user Grid resource brokers; in which the user sends their jobs to a broker, who submits them on their behalf to clusters the user can access.

- **Centralised meta-scheduler:** users forward jobs to a centralised entity, which sends the jobs to the clusters at which they execute. The centralised component is responsible for determining which resources to allocate to the job and, in some cases, for migrating jobs if the load conditions change.

- **Hierarchical meta-scheduler:** schedulers are organised in a hierarchy, with jobs arriving either at the root of the hierarchy or at the LRMSs.

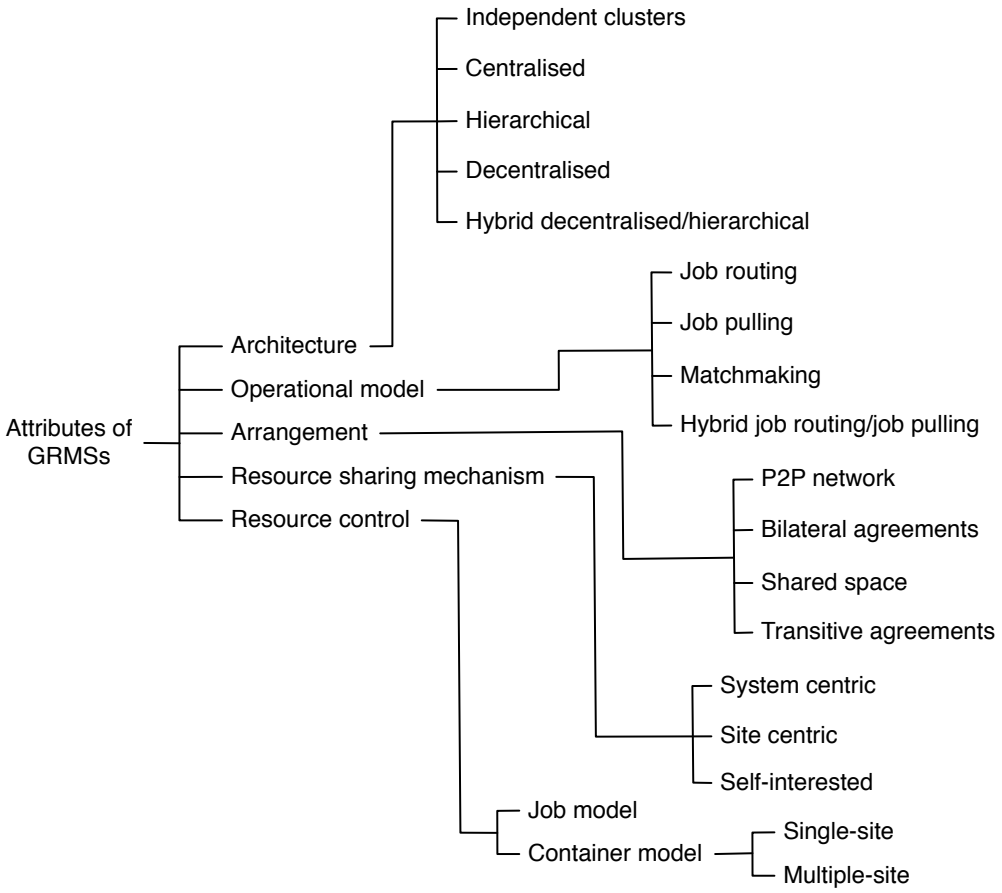


Figure 2.3: Taxonomy on Grid resource management systems.

- **Decentralised meta-scheduler:** clusters can share with one another jobs that arrive at their LRMSs. Links can be defined either in a static manner (*i.e.* by the system administrator at the system start-up phase) or dynamically (*i.e.* peers are selected dynamically at runtime).[3]

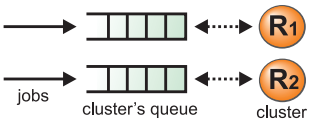
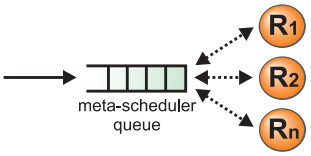
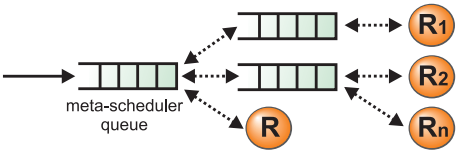
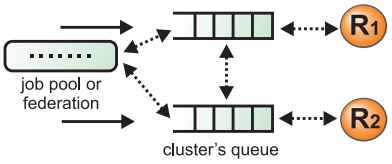
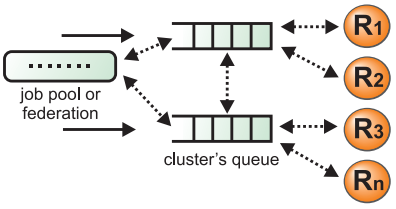
- **Hybrid decentralised/hierarchical meta-scheduler:** each Grid site is managed by a hierarchical meta-scheduler. Additionally, the root meta-schedulers can share the load with one another.

This comprehensive classification captures the main forms through which schedulers and brokers can be organised in Grids and large-scale testbeds. However, some categories can be extended. For example, site schedulers can be organised in several decentralised ways and use various mechanisms for resource sharing, such as a mesh network in which contracts are established between schedulers [86, 102] or a P2P network with a bartering-inspired economic mechanism for resource sharing [6].

Systems are also heterogeneous regarding their operational models, or the mechanism that ensures jobs entering the system arrive at the resource in which they run. We use and extend the classification proposed by Iosup *et al.* as follows:

[3]Grit [91] discusses the types of contracts that schedulers (or brokers) can establish with one another.

Table 2.1: Architectural models of Grid resource management systems.

Model	Architecture	Example Systems
Independent clusters		Portable Batch System (PBS), Sun Grid Engine (SGE)
Centralised meta-scheduler		EGEE Workload Management Service (WMS), KOALA, PlanetLab
Hierarchical meta-scheduler		Computing Center Software (CCS)
Decentralised		OurGrid, Askalon, Shirako
Hybrid of decentralised and hierarchical		Delegated Matchmaking

- **Job routing:** schedulers route jobs from the arrival point to the resources where they run using a push operation (scheduler-initiated routing).
- **Job pulling:** resources pull jobs from a higher-level scheduler (resource-initiated routing).
- **Matchmaking:** the resource manager connects jobs and resources, acting as a broker to match requests from users and availability from resources.
- **Job routing/pulling hybrid:** systems use a job pool to (or from) which jobs are pushed (or pulled) by site busy (or unoccupied) LRMSs [90].

2.4.2 Arrangements between Brokers

In decentralised or semi-decentralised architectures, brokers and schedulers can establish various types of communication arrangements. It is important to distinguish the way

links between sites are established and their communication pattern from the mechanism used for negotiating the resource shares. We classify existing systems according to the communication model as follows:

- **P2P network:** the sites of the Grid are peers in a P2P network, and they use the network to locate sites where the jobs can run [7, 27].
- **Bilateral sharing agreements:** sites establish bilateral agreements through which a site can locate another suitable site to run a given job. The redirection or acceptance of jobs occurs only between sites that have a sharing agreement [67].
- **Shared spaces:** sites co-ordinate resource sharing via shared spaces such as federation directories and tuple spaces [90, 149].
- **Transitive agreements:** similar to bilateral agreements, however, a site can utilise resources from another site with which it has no direct agreement [86, 102].

2.4.3 Resource Sharing Mechanisms

Although existing work can present similar communication and organisational models for schedulers, the resource sharing mechanisms can be different. The schedulers or brokers can use mechanisms for resource sharing from the following categories:

- **System centric:** the mechanism aims to maximise the overall utility of the participants. Such mechanisms aim, for example, to balance the load between sites [101] and prevent free riding [6].
- **Site centric:** brokers and schedulers strive to maximise the utility of the participants within the site they represent without the explicit goal of maximising overall system utility [27, 148].
- **Self-interested:** schedulers/brokers act with the goal of maximising their own utility, generally given by profit, yet satisfying the requirements of their users. They also do not take into account the utility of the whole system [102].

2.4.4 Resource Control Techniques

Recently, organisations have deployed resource managers that allow the partitioning of physical resources and the allocation of raw resources which users can customise with the operating system and software they prefer. Virtualisation technologies such as Xen [15, 137] and VMWare[4] make this partitioning possible.

The emergence of virtualisation technologies has resulted in the creation of testbeds where different communities can access multiple-site slices (*i.e.* multiple-site containers) and customised virtual clusters [38, 77, 108, 141]. Slices run concurrently, isolated from one another. This type of resource control is termed here as a *container model*. Most of the existing Grid middleware employ a *job model* in which jobs are routed until they reach the sites' LRMSs for execution. Both models can co-exist; thus an existing Grid technology

[4] http://www.vmware.com/

can be deployed in a workspace enabled by container-based resource management [127, 146]. We classify systems in the following categories:

- **Job model:** this is the model currently utilised by most Grid systems. Jobs are directed or pulled across the network until they arrive at the nodes where they run.
- **Container-based:** systems in this category can manage a cluster of computers within a site by means of virtualisation technologies [38, 108]. Resources are bound to virtual clusters or workspaces according to a user's demand. They commonly provide an interface through which one can allocate a set of nodes, generally virtual machines, and configure them with the operating system and software of choice.
  - **Single-site:** these container-based resource managers allow the user to create a customised virtual cluster using shares of the physical machines available at the site. These resource managers are termed here as single-site because they usually manage the resources of one administrative site [38, 76]. However, with proper extensions they can enable container-based resource control at multiple sites [127].
  - **Multiple-site:** utilising the features of single-site container-based resource managers to create networks of virtual machines on which an application or existing Grid middleware can be deployed [146]. These networks of virtual machines are termed here as *multiple-site containers* because they can comprise resources bound to workspaces at multiple administrative sites [102, 146, 156, 164]. These networks of virtual machines are also referred to as virtual Grids [97] or slices [141].

Some systems such as Shirako [102] and VioCluster [157] provide container-based resource control. Shirako also offers resource control at the job level by providing a component that is aware of the resources leased [146]. This component recommends which site can execute a given job.

2.5 Virtual Organisations

The idea of user communities or VOs underlies several of the organisational models adopted by Grid systems and guides many efforts for fair resource allocation within Grids. However, existing systems are heterogeneous in terms of the VO awareness of their scheduling and resource allocation mechanisms. One may advocate that several systems, not explicitly designed to support VOs, are suitable for resource management within a VO, however we restrict ourselves to classify systems according to the:

- VO awareness of their resource allocation and scheduling mechanisms; and
- Provision of tools for handling different issues related to the VO life cycle.

The life cycle of a VO comprises four distinct phases namely *creation*, *operation*, *maintenance*, and *dissolution*. During the creation phase, an organisation looks for collaborators and then selects a list of potential partners to start the VO. The operation phase

involves resource management, task distribution, and usage policy enforcement [51, 192]. The maintenance phase deals with the adaptation of the VO, such as allocation of additional resources according to its users' demands, while VO dissolution involves legal and economic issues such as determining the success or failure of the VO, intellectual property, and revocation of access and usage privileges. Our criteria for classifying VOs are presented in Figure 2.4.

For the VO awareness of scheduling mechanisms, we can classify the systems as:

- **Multiple VOs:** those mechanisms that perform scheduling and allocation considering the various VOs existing within a Grid.
- **Single VO:** those mechanisms suitable for job scheduling within a VO.

Furthermore, the Grid computing community has used the idea of VO in slightly different ways. For example, in the Open Science Grid (OSG) [144], VOs are recursive and may overlap.

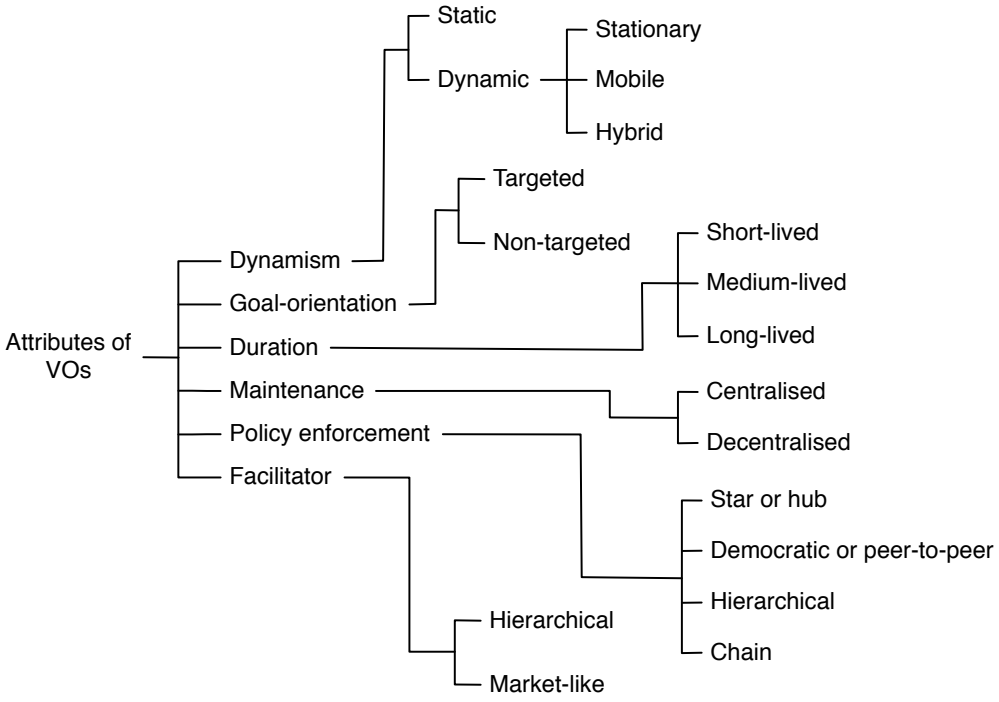


Figure 2.4: Taxonomy of Grid facilitated virtual organisations.

With regard to dynamism, we classify VOs as:

- **Static:** although Grid computing is considered an enabler for dynamic VOs, it has been used to create more static and long-term collaborations such as APAC [134], EGEE [65], and the UK e-Science Centre [96]. A static VO has a pre-defined number of participants and its structure does not change over time.
- **Dynamic:** a dynamic VO presents a number of participants that changes constantly as the VO evolves [195]. New participants can join, whereas existing participants may leave. A dynamic VO can be:

  - **Stationary:** composed of highly specialised resources including supercomputers, clusters of computers, personal computers, and data resources. The components of the VO are not mobile.
  - **Mobile:** composed of mobile resources such as Personal Digital Assistants (PDAs), mobile phones, and personal computers, and the VO is highly responsive and adapts to different contexts [195]. Mobile VOs are found in disaster handling and crisis management situations. A VO can be hybrid, having both stationary and mobile components.

Considering goal-orientation, we divide VOs into two categories:

- **Targeted:** an alliance or collaboration created to explore a market opportunity or achieve a common research goal. A VO for e-Science collaboration is an example of a targeted VO as the participants have a common goal [96].
- **Non-targeted:** characterised by the absence of a common goal; it generally comprises participants who pursue different goals, yet benefit from the VO by pooling resources. This VO is highly dynamic because participants can leave when they achieve their goals.

In respect to the duration of VOs, they are typically of the following types:

- **Short-lived:** lasts for minutes or hours.
- **Medium-lived:** lasts for several days or weeks. For example, a scientist who needs to carry out experiments that take several days to finish could start a medium-lived VO. The experiments may require input data and the scenario may become simpler if the VO model is used; the VO is not required as soon as the experiments have been carried out.
- **Long-lived:** lasts for several months or years. For example, a long-lived VO could be formed to explore a market opportunity (goal-oriented) or pool resources to achieve disparate objectives (non-targeted).

As discussed earlier, the formation and maintenance of a VO present several challenges. Previous work has tackled these challenges in different ways, which in turn have created different formation and maintenance approaches. We classify the formation and membership, or maintenance, as:

- **Centralised:** a trusted third party, such as OSG [144] and the Enabling Grids for E-SciencE [65], controls the formation and membership of a centralised VO. OSG provides an open market where providers and users can advertise their needs and intentions; a provider or user may form a VO for a given purpose. EGEE provides a hierarchical infrastructure to enable the formation of VOs.
- **Decentralised:** no third party is responsible for enabling or controlling the formation and maintenance. This kind of VO can become complex; they require the negotiation of Service Level Agreements (SLAs) among multiple participants. In addition, the monitoring of SLAs and members' commitment to the VO are difficult to control. The VO also needs to self-adapt when participants leave or new participants join.

Regarding the enforcement of policies, VOs can follow different approaches, such as hub or democratic. This is also termed as topology. Katzy *et al.* [107] classify VOs in terms of topology, identifying the following types: chain, star or hub, and peer-to-peer. Sairamesh *et al.* [159] identify business models for VOs; which are analogous to topologies. However, by discussing the business models for VOs, the authors are concerned with a larger set of problems, including enforcement of policies, management, trust and security, and financial aspects. We classify the enforcement and monitoring of policies as **star** or **hub**, **democratic** or **peer-to-peer**, **hierarchical**, and **chain**.

Some projects such as OSG [144] and EGEE [65] aim to establish consortiums or clusters of organisations, which in turn allow the creation of dynamic VOs. Although not very related to the core issues of VOs, they aim to address an important problem: the establishment of trust between organisations and the means for them to look for and find potential partners. These consortiums are classified as **hierarchical** and **market-like**. A market-like structure is any infrastructure that offers a market place that organisations can join, and show interest in either starting a new collaboration or accepting to participate in an ongoing collaboration. These infrastructures may make use of economic models such as auctions, bartering, and bilateral negotiations.

2.6 An Investigation of Existing Work

This section investigates relevant work in greater detail. First, it describes a list of systems with decentralised architectures. Second, it presents systems of hierarchical structure, resource brokers, and meta-scheduling frameworks. Third, it discusses work on interoperation of Grids and large-scale testbeds. Finally, this section presents relevant work focusing on VO issues.

2.6.1 Distributed Architecture Based Systems

Condor Flocking: The flocking mechanism used by Condor [67] provides a software approach to interconnect pools of Condor resources [117]. The mechanism requires manual configuration of sharing agreements between Condor pools, with each pool and workstation owner maintaining full control about when external jobs can use their resources.

A layered design for Condor's flocking mechanism enables Condor's Central Manager (CM) [117] and other Condor machines to remain unmodified and operate transparently from the flock.

Gateway Machines (GWs) are the basis of the flocking mechanism. GWs act as resource brokers between pools, with at least one GW in each Condor pool. Every GW has a configuration file describing the subset of connections that GW maintains with other GWs. Periodically a GW queries the status of its pool from the CM, building a list of idle resources. The GW then sends this list to the other GWs to which it is connected. The GW receiving this list periodically chooses a machine from the list, and advertises itself with the characteristics of that machine to the CM. The flocking protocol, a modified version of the normal Condor protocol [117], allows the GWs to create shadow processes. Hence, a submission machine is under the impression of contacting the execution machine directly.

Self-Organising Flock of Condors: The original flocking scheme of Condor has the drawback that GWs need to know a priori all pools with which they can share resources [67]. This static information poses limitations regarding the number of resources available and resource discovery. Butt *et al.* [27] introduced a self-organising resource discovery mechanism for Condor, which allows pools to dynamically discover one another and resources available. The P2P network used by the flocking mechanism relies on Pastry [155] and takes into account the network proximity, which may save bandwidth in network communications.

Shirako: Shirako [102] is a system for on-demand leasing of shared networked resources across clusters. Shirako utilises a leasing abstraction in which *site authorities* representing provider sites offer their resources to be provisioned by *brokers* to *guest applications*. Shirako brokers are responsible for co-ordinating resource allocation across provider sites. The provisioning determines the proportion of each resource that each guest application receives, when and where. Site authorities define the amount of resource that providers give to service requests, and to which brokers; providers offer resources to brokers by issuing resource tickets. When a broker approves a request, it issues a ticket that is redeemable for a lease at a site authority. The ticket specifies the type of resource, the number of resource units granted and the interval for which the ticket is valid.

A *service manager* is a component that represents the guest application and uses Shirako's lease API to request resources from the broker. The service manager determines when and how to redeem existing tickets, extend existing leases, or acquire new leases to meet changing demand. The system allows guest applications to renew or extend their leases. The broker and site authorities match accumulated pending requests with resources under the authorities' control. The broker prioritises requests and selects resource types and quantities to serve them. Site authorities use Cluster on Demand [38] to configure the resources allocated at the remote sites.

Shirako's leasing abstraction is a useful basis for co-ordinating resource sharing in systems that create environments consisting of virtual machines [1, 108, 158, 164]. Ramakrishnan *et al.* [146] used Shirako's leasing core to provide a hosting model where Grid deployments run on multiple-site containers isolated from one another. An Application Manager (AM), which is the entry point of jobs from a VO or Grid, interacts with a Grid Resource Oversight Co-ordinator (GROC) to obtain a recommendation of a site to submit jobs.

VioCluster: VioCluster is a system that enables dynamic machine trading across clusters of computers [157]. VioCluster introduces the idea of *virtual domain*. Initially comprising its physical domain of origin (*i.e.* a cluster of computers), a virtual domain can grow the number of its computing resources, thus dynamically allocating resources from other physical domains according to the demands of its user applications.

VioCluster presents two important system components: the creation of dynamic virtual domains and the mechanism used to negotiate resource sharing. VioCluster uses machine and network virtualisation technology to move machines between domains. Each virtual domain has a broker that interacts with other domains. The broker has a borrowing and a lending policy; the borrowing policy determines the circumstances under which the broker attempts to obtain more machines, while the lending policy governs when the broker lets another virtual domain make use of the machines within its physical domain.

Virtualisation technology[5] simplifies the transfer of machines between domains. A resource borrowed by a virtual domain *A* from a physical domain *B* is utilised to run a virtual machine, which matches the configuration of the machines in domain *A*. Network virtualisation enables establishing virtual network links to connect the new virtual machine to the nodes of domain *A*. For the presented prototype, PBS [188] manages the nodes of the virtual domain. PBS is aware of the computers' heterogeneity and does not schedule jobs on a mixture of virtual and physical machines. The size of the work queue in PBS indicates the demand within a domain.

OurGrid: OurGrid [7] is a resource sharing system organised as a P2P network of sites that share resources equitably, forming a Grid to which they all have access. Aiming to ease the assembly of Grids, OurGrid provides connected sites with access to the Grid resources with the minimal guarantees needed. OurGrid supports the execution of Bag-of-Tasks (BoT) applications; parallel applications composed of a set of independent tasks that do not communicate with one another during their execution. In contrast to other Grid infrastructures, the system does not require offline negotiations if a resource owner wants to offer their resources to the Grid.

The three participants in OurGrid's resource sharing protocol are *clients*, *consumers*, and *providers*. A client requires access to the Grid resources to run their applications. The consumer receives requests for resources from clients, proceeds to find the resources able to serve the request, and then executes the tasks on the resources. The provider manages the resources shared in the community and makes them available to consumers.

OurGrid uses a resource exchange mechanism termed *network of favours*. A participant *A* is doing a favour for participant *B* when *A* allows *B* to use A's resources. According to the network of favours, every participant does favours for other participants expecting the favours to be reciprocated. In conflicting situations, participants prioritise those who have done them favours in the past. The more favours participants do, the more rewards they expect. The participants account locally for their favours, and cannot profit from them other than expecting other participants to do favours for them in return. Detailed experiments have demonstrated the scalability of the network of favours [6], showing that the larger the network becomes, the more fair the mechanism performs.

Delegated Matchmaking: Iosup *et al.* [101] introduced a matchmaking protocol in which a computing site binds resources from remote sites to its local environment. Further they created a network of sites over the local cluster schedulers to manage the resources of the interconnected Grids. Sites are organised according to administrative and political agreements, thus establishing parent-child links. Then, they created a hierarchy of sites with the Grid clusters at the bottom. Supplementing the hierarchical links, they also created sibling links between sites that are at the same hierarchical level and operate under the same parent site. The proposed delegated matchmaking mechanism enables the delegation of requests for resources up and down the hierarchy, achieving a decentralised network.

Their architecture is different from work that considers job routing. The main aim of the matchmaking mechanism is to delegate ownership of resources to the user who requested them via this network of sites, and add the resources transparently to the user's local site. When a site cannot satisfy a request locally, the matchmaking mechanism adds

[5] VioCluster relies on User Mode Linux (UML) for machine virtualisation: http://user-mode-linux.sourceforge.net/

remote resources to the user's site. This approach aims to simplify security issues as the mechanism adds the resources to the local pool of trusted resources.

Grid Federation: Ranjan *et al.* [147] proposed a system that federates clusters of computers via a shared directory. Grid Federation Agents (GFAs), representing the federated clusters, post quotes about idle resources (*i.e.* a claim stating that a given resource is available) and, upon the arrival of a job, query the directory to find a suitable resource to execute the job. The directory is a shared-space implemented as a Distributed Hash Table (DHT) P2P network that matches quotes and user requests [149].

Ranjan *et al.* [148] also proposed an SLA-driven co-ordination mechanism for Grid superscheduling. GFAs negotiate SLAs and redirect requests using a Contract-Net protocol. A GFA is an LRMS and uses a greedy algorithm to schedule resource requests. GFAs engage into bilateral negotiations for each request they receive, without considering network locality.

Askalon: Siddiqui *et al.* [165] introduced a capacity planning architecture with a three-layer negotiation protocol for advance reservation on Grid resources. Allocators reserve individual nodes and co-allocators reserve multiple nodes for a single Grid application. A co-allocator receives requests from users and generates alternative offers the user can utilise to run their application. A co-allocation request can comprise a set of allocation requests, each corresponding to an activity of the Grid application. A workflow with a list of activities is an example of a Grid application requiring co-allocation of resources [198]. Co-allocators aim to agree on Grid resource sharing. Their proposed co-ordination mechanism produces contention-free schedules, either by eliminating conflicting offers or lowering some allocators' objective levels.

GRUBER/DI-GRUBER: Dumitrescu *et al.* [54] highlighted that challenging usage policies can arise in VOs comprising participants and resources from different physical organisations. Participants want to delegate access to their resources to a VO, yet maintain such resources under the control of local usage policies. Dumitrescu *et al.* sought to address the following issues:

- How to enforce usage policies at the resource and VO levels.
- What mechanisms a VO uses to ensure policy enforcement.
- How to carry out distribution of policies to enforcement points.
- How to make policies available to VO job and data planners.

Dumitrescu *et al.* proposed a policy management model in which participants can specify the maximum percentage of resources delegated to a VO, and a VO can specify the maximum percentage of resource usage it wishes to delegate to a given user group. Based on this model, they proposed a Grid resource broker termed GRUBER [52]. GRUBER architecture is composed of four components, namely:

- **Engine:** which implements several algorithms to detect available resources.
- **Site monitoring:** responsible for collecting data on the status of Grid resources.

- **Site selectors:** consist of tools that communicate with the engine and provide information about which sites can execute jobs.

- **Queue manager:** which resides on the submitting host and decides how many jobs to execute and when.

Users willing to execute jobs, do so by sending them to submitting hosts. The integration of existing external schedulers with GRUBER is made in the submitting hosts. The external scheduler utilises GRUBER either as the queue manager that controls the start time of jobs and enforces VO policies, or as a site recommender when the queue manager is not available.

Dumitrescu *et al.* [53] also introduced a distributed version of GRUBER termed DI-GRUBER. DI-GRUBER works with multiple decision points, which gather information to steer resource allocations defined by Usage Service Level Agreements (USLAs). These points make decisions on a per-job basis to comply with resource allocations to VO groups. Authors advocate that four to five decision points would be enough to handle job scheduling for a Grid 10 times larger than Grid3 [78] at the time the experiments were carried out [53].

Other important work: Medusa is a stream processing system that allows the migration of stream processing operators from overloaded to under-utilised resources. Balazinska *et al.* [14] proposed a load balancing mechanism for Medusa. Offloading occurs on the basis of the marginal cost of requests, which equates to the increase or decrease in the cost curve resulting from the acceptance or removal of a request.

NWIRE [161] links various resources to a meta-computing system, or meta-system, which also enables the scheduling in these environments. A meta-system comprises interconnected *MetaDomains*. A *MetaManager* manages a MetaDomain, or a set of *ResourceManagers*. A ResourceManager interfaces with the scheduler at the cluster level; the MetaManager permanently collects information about all of its resources, handles all requests inside its MetaDomain and works as a resource broker to other MetaDomains. In this way, requests received by a MetaManager can be submitted either by users within its MetaDomain or by other MetaManagers. Each MetaManager contains a scheduler that maps requests for resources to a specific resource in its MetaDomain.

Grimme *et al.* [90] presented a mechanism for resource providers collaboration. Jobs are interchanged via a central job pool, with every LRMS adding jobs it cannot start immediately to the pool. After scheduling local jobs, a LRMS can schedule jobs from the central pool if resources are available.

Dixon *et al.* [50] provided a tit-for-tat mechanism for resource allocation in large-scale distributed infrastructures based on local non-transferable currency. Each domain locally maintains the currency about the past provision of resources to other domains; as credit. This creates pair-wise relationships between administrative domains, which resemble OurGrid's network of favours [7]. As the information about exchanged resources decays with time, recent behaviour carries greater importance. Simulation results show that, for an infrastructure like PlanetLab, the proposed mechanism is fairer than its current free-for-all approach.

Graupner *et al.* [88] introduced a resource control architecture for federated utility data centres. The architecture groups physical resources in virtual servers to the services with

which they are mapped. The meta-system is the upper layer, implemented as an overlay network whose nodes contain descriptive data about the two layers below. Allocations change according to service demand, which requires control algorithms to be reactive and deliver quality solutions. The control layer performs allocation of services to virtual server environments.

2.6.2 Hierarchical Systems, Brokers, and Meta-Scheduling

This section describes systems that are organised in a hierarchical manner. In addition, this section describes work on Grid resource brokering and frameworks suitable to build meta-schedulers.

Computing Centre Software (CCS): CCS [23] manages geographically distributed High Performance Computers (HPC). It consists of three components: the CCS, which is a vendor-independent LRMSs for local HPC systems; the Resource and Service Description (RSD), used by the CCS to specify and map hardware and software components of computing environments; and the Service Coordination Layer (SCL), which co-ordinates the use of resources across computing sites.

The CCS controls the mapping and scheduling of interactive and parallel jobs on massive parallel systems. It uses the island concept, where each island has components for user interface, authorisation and accounting, scheduling of user requests, access to the physical parallel system, system control, and management of the island. At the meta-computing level, the Centre Resource Manager (CRM) sits over the CCS islands, and exposes their scheduling and brokering features. When a user submits an application, the CRM maps the user request to the static and dynamic information regarding available resources. Once the CRM finds resources, it requests the allocation of all required resources at all the islands involved. If not all resources are available, the CRM either re-schedules the request or rejects it. Centre Information Server (CIS) is a passive component that contains information about resources and their statuses, and is analogous to Globus Meta-computing Directory Service (MDS) [79]. The CRM uses the CIS to obtain information about available resources.

The Service Co-ordination Layer (SCL) sits one level above the LRMSs to co-ordinate the use of resources across the island network. Organised as a network of co-operating servers, where each server represents one computing centre, the centres determine which resources are available to others and retain full autonomy over them.

EGEE Workload Management System (WMS): EGEE WMS has a semi-centralised architecture [185]. One or more schedulers can be installed in the Grid infrastructure, each providing scheduling functionality for a group of VOs. The EGEE WMS components are:

- User Interface (UI), where the user dispatches the jobs;
- Resource Broker (RB), which uses Condor-G [84];
- Computing Element (CE), the cluster front-end;
- Worker Nodes (WNs), the cluster nodes;
- Storage Element (SE), to store job files; and

- Logging and Bookkeeping service (LB), which registers job events.

Condor-G: Condor-G [84] employs software from Globus [79] and Condor [84] to allow users to utilise resources spanning multiple domains as if they all belong to one personal domain. Although existing work considers Condor-G as a resource broker itself [187], it can also provide a framework to build meta-schedulers.

The *GlideIn* mechanism of Condor-G can start a daemon process on a remote resource using standard Condor mechanisms to advertise the resource availability to a Condor collector process, which the *Scheduler* queries to learn about available resources. Condor-G uses Condor mechanisms to match locally queued jobs to the resources advertised by these daemons and to execute them on those resources. Condor-G submits an initial GlideIn executable (a portable shell script), which in turn uses Grid Security Infrastructure (GSI)-authenticated GridFTP to retrieve the Condor executables from a central repository. By submitting GlideIn executables to all remote resources capable of serving a job, Condor-G can guarantee optimal queuing times to user applications.

Gridbus Broker: Gridbus Grid resource broker [187] is a user-centric broker that provides scheduling algorithms for both computing- and data-intensive applications. In Gridbus, each user has their own broker, which represents them by selecting resources that minimise service constraints such as execution deadline and provided budget, submitting jobs to remote resources, and copying input and output files. Gridbus interacts with various Grid middleware [47, 187].

GridWay: GridWay [98] is a resource broker that provides a framework for executing jobs in a 'submit and forget' fashion. The framework performs job submission and execution monitoring, with job execution adapting itself to dynamic resource conditions and application demands to improve performance. GridWay enables adaptation through application migration following performance degradation, sophisticated resource discovery, requirements change, or remote resource failure.

The framework is modular with the following modules set on a per-job basis: resource selector, performance degradation evaluator, *prolog*, wrapper, and *epilog*. As the names of the first two modules or steps are intuitive, we describe only the last three here. During prolog, the component responsible for job submission (*i.e.* submission manager) submits the prolog executable, which configures the remote system and transfers executable and input files. In the case of an execution restart, the prolog also transfers restart files. GridWay submits the wrapper executable after prolog; enclosing the actual job to obtain its exit code. The epilog script transfers the output files and cleans the remote resource.

GridWay also enables the deployment of virtual machines in a Globus Grid [126]. GridWay performs the scheduling and selection of suitable resources, and provides a virtual workspace for each Grid job. A pre-wrapper phase performs advanced job configuration routines, whereas the wrapper script starts a virtual machine and triggers the application job on it.

GridWay framework has also been used to enable hierarchical meta-scheduling structures [99]. Each target group of resources or Grid is handled as another resource, in a recursive manner. This approach enables multiple-level hierarchies of meta-schedulers.

KOALA: Mohamed and Epema [125] proposed KOALA; a Grid scheduler that supports resource co-allocation. KOALA Grid scheduler interacts with LRMSs to execute jobs.

The work proposed an alternative to advance reservation at LRMSs, when reservation features are not available, allowing the allocation of processors from multiple sites simultaneously.

SNAP-Based Community Resource Broker: The Service Negotiation and Acquisition Protocol (SNAP)-based community resource broker uses a three-phase commit protocol. SNAP is proposed to counter traditional advance reservation facilities not coping with the fact that information availability may change between the moment at which resource availability is queried and the time when the reservation of resources is actually performed [92]. The three phases of the SNAP are:

1. Resource availability is queried and probers are deployed, which inform the broker in case the resource status changes;
2. Resources are selected and reserved; and
3. The job is deployed on the reserved resources.

Platform Community Scheduler Framework (CSF): CSF [143] provides a toolset to create Grid meta-schedulers or community schedulers. The resulting meta-scheduler enables users to define protocols to interact with resource managers in a system independent manner. A component termed Resource Manager (RM) Adapter works as the interface with LRMSs. CSF supports the GRAM protocol [79] to access the services of LRMSs that do not support the RM Adapter interface.

Platform's Load Sharing Facility (LSF) and MultiCluster products use CSF to provide a framework for meta-scheduling. The Grid Gateway interface integrates LSF and CSF. A scheduling plug-in for Platform LSF scheduler decides which LSF jobs to forward to the meta-scheduler, based on information obtained from a Grid Gateway information service. When the plug-in forwards a job to the meta-scheduler, the submission and monitoring tools dispatch the job and query its status information through the Grid Gateway. The Grid Gateway uses the job submission, monitoring, and reservation services from the CSF. Platform MultiCluster also allows clusters using LSF to exchange jobs.

Other important work: Kertész *et al.* [109] introduced a meta-brokering system in which the meta-broker, invoked through a Web portal, submits jobs, monitors job status, and copies output files using brokers from different Grid middleware, such as NorduGrid Broker and EGEE WMS.

Kim and Buyya [110] tackled the problem of fair-share resource allocation in hierarchical VOs, providing a model for hierarchical VO environments based on a resource sharing policy, and proposing a heuristic solution.

2.6.3 Inter-Operation of Grids and Large-Scale Testbeds

This section discusses relevant work on inter-operation of Grids and large-scale testbeds.

PlanetLab: PlanetLab [141] is a large-scale testbed that enables the creation of slices, or distributed environments based on virtualisation technology. A slice is a set of virtual machines, each running on a unique node. The service running on the slice manages the set of virtual machines. Each individual virtual machine contains no information about

other virtual machines in the set. Each service deployed on PlanetLab runs on a slice of PlanetLab's global pool of resources. Multiple slices can run concurrently and each slice isolates services from other slices.

The principals in PlanetLab are:

- **Owner:** an organisation that hosts (owns) one or more PlanetLab nodes.
- **User:** a researcher who deploys a service on a set of PlanetLab nodes.
- **PlanetLab Consortium (PLC):** a centralised trusted intermediary that manages nodes for a group of owners, and creates slices on those nodes on behalf of a group of users.

When PLC acts as a Slice Authority (SA), it maintains the state of the list of system-wide slices for which it is responsible. The SA provides an interface through which users register themselves, create slices, bind users to slices, and request the instantiation of a slice on a set of nodes. PLC, acting as a Management Authority (MA), maintains a server to install and update the software running on the nodes it manages, and monitor these nodes for correct behaviour, taking appropriate action when detecting anomalies and failures. The MA maintains a database of registered nodes; each node being affiliated with an organisation (owner) and located at a site of the organisation. MA provides an interface for node owners to register nodes with the PLC, and users and slices authorities to obtain information about the set of nodes managed by the MA.

PlanetLab's architecture has evolved to enable decentralised control or federations of PlanetLabs [141]. The recent architecture design splits the PLC into two components namely the MA and SA, which allow PLC-like entities to evolve these components independently. In this way, autonomous organisations can federate and define peering relationships with one another. One of goals of PlanetLab Europe[6] and OneLab2 [85] is for peering relationships with other infrastructures. A resource owner can choose a MA to which they want to provide resources. MAs, in turn, can blacklist particular SAs. A SA can trust only certain MAs to provide the virtual machines required by its users. This enables various types of agreements between SAs and MAs.

It is also important to mention that Ricci *et al.* [151] discussed issues related to the design of a general resource allocation interface that is sufficiently wide for allocators in a large variety of current and future testbeds, including PlanetLab federations. They have described an allocator as a component that receives the users' abstract description for the required resources and the resource status from a resource discoverer, and produces allocations performed by a deployment service. An allocator's goal is to allow users to specify characteristics of their slice in high-level terms and find resources to match these requirements. Describing their experience in designing PlanetLab and Emulab,[7] Ricci *et al.* advocated several important issues, including:

- In future infrastructures, several allocators may co-exist and it might be difficult for them to operate without interfering with one another;

[6] http://www.planet-lab.eu/
[7] http://www.emulab.net/

- With the proportional-share philosophy of PlanetLab, where multiple management services can co-exist, allocators do not have guarantees over any resources; and
- Co-ordination between allocators can be required.

Grid Interoperability Now – Community Group (GIN-CG): GIN-CG has been working on providing interoperability between Grids by developing components and adapters that enable secure and standard job submissions, data transfers, and information queries [152]. These efforts provide the basis for load management across Grids by facilitating standard job submission and request redirection. They also enable secure access to resources and data across Grids. Although GIN-CG's efforts are essential, its members have also highlighted the need for common resource allocation and brokering across Grids.[8]

Interoperable Brokering Service: Elmroth and Tordsson [60] proposed a standards-based Grid resource brokering service for providing interoperability among different Grid middlewares. The brokering service architecture is distributed, user-oriented and comprises a job submission client, and some middleware-specific interfaces. Currently, interfaces for the NorduGrid [56] Advanced Resource Connector (ARC) middleware [58] and Globus are available.

The broker aims to reduce the Total Time to Delivery (TTD) – the time between job submission and staging out of output files – for each job submission. Benchmarks are used to estimate job execution times; at submission, the user must specify one or more benchmarks with performance characteristics similar to those of the job along with an execution time estimate. Avoiding the overhead of executing the benchmarks at each job submission, the benchmarks are run once for each Grid resource and published in an index server. The broker uses the benchmarks to make execution time estimates for the candidate Grid resources assuming a linear scaling [61].

In addition, the brokering service supports advance reservations, which are used to provide job waiting time guarantees and co-allocate Grid resources [62]. The proposed co-allocation algorithm strives to find the earliest common start time for all jobs within a given job start window. The earliest start time is obtained by creating a set of reservations within a period smaller than the job start window.

A proxy mechanism enables the proposed brokering service to achieve job submission interoperability between Grid middlewares by allowing clients to express their jobs in the native job description language of their middleware [62].

Other important work: Boghosian *et al.* [20] performed experiments using resources from more than one Grid for three projects: Nektar, SPICE and Vortonics. The applications in these three projects require numerous computing resources, achievable only through interconnected Grids. Although they used resources from multiple Grids during the experiments, they also emphasised that several human interactions and negotiations are required to use federated resources. They highlighted that even if interoperability at the middleware level existed, it would not guarantee that the federated Grids are utilised for large-scale distributed applications due to important additional requirements, such as compatible and consistent usage policies, automated advanced reservations, and co-scheduling.

[8]The personal communication among GIN-CG members is online at:
http://www.ogf.org/pipermail/gin-ops/2007-July/000142.html

Caromel *et al.* [34] proposed the use of a P2P network to acquire resources dynamically from Grid'5000 and desktop machines, to run computationally intensive applications. The P2P network and Grid'5000 communicate via Secure Shell (SSH) tunnels.[9] Moreover, the allocation of nodes for the P2P network uses the ProActive [33] deployment framework; deploying Java Virtual Machines on the allocated nodes.

In addition to GIN-CG's efforts, Wang *et al.* [190] introduced another Grid middleware interoperability approach, describing a gateway approach to achieve interoperability between gLite [57], the middleware used in EGEE, and CNGrid GOS, the Chinese National Grid middleware.[10] Their work focused on job management interoperability, also describing interoperability between the different protocols used for data management and resource information. The proposed interoperability approach encapsulates gLite as a site job manager of GOS, whereas gLite submits jobs to GOS resources in a different manner; an extended job manager is instantiated for each job submitted to a GOS resource. The extended job manager sends the whole batch job to execute in the CNGrid.

2.6.4 Virtual Organisations

This section investigates how projects address different challenges in the VO life cycle. Two main categories of projects have been identified: the facilitators for VOs, which provide means for building clusters of organisations, hence enabling collaboration and formation of VOs; and enablers for VOs, which provide middleware and tools to help in the formation, management, maintenance, and dissolution of VOs. This classification is not strict as a project can belong to both categories, providing software for enabling VOs and working as a consortium, which organisations can join and then start collaborations that are more dynamic.

The investigation is divided into three parts: middleware and software infrastructure for enabling VOs, consortiums and charters that facilitate the formation of VOs, and other relevant work that addresses issues related to a VO's life cycle.

Enabling Technology

Enabling a VO means to provide the required software tools to help in the different phases of the VO life cycle. Several projects do not address all the phases due to the complex challenges in the life cycle.

CONOISE Project: CONOISE [140] uses a marketplace based on combinatorial auctions to form VOs [131]. Combinatorial auctions allow a high degree of flexibility, whereby VO initiators can specify a broad range of requirements. A combinatorial auction allows the selling of multiple units of a single item or multiple items simultaneously. However, combinatorial auctions lack on means for bid representation and efficient clearing algorithms to determine prices, quantities, and winners. As demonstrated by Dang [42], clearing combinatorial auctions is an NP-Complete problem. Thus, CONOISE provides polynomial and sub-optimal auction clearing algorithms for combinatorial auctions.

In VOs enabled by CONOISE, agents are the stakeholders. Demonstrating VO formation, a user may request a service of an agent, who in turn verifies whether it is able to

[9] http://en.wikipedia.org/wiki/Secure_Shell
[10] http://www.cngrid.org

provide the service at the specified time. If the agent cannot provide the service, it looks for Service Providers (SPs) offering the service. The Requesting Agent (RA) then starts a combinatorial auction and sends call for bids to SPs. Once the RA receives the bids, it determines the best set of partners and starts the formation of a VO. Once the VO is formed, RA becomes the VO manager.

An agent that receives a call for bids can: (a) decide not to bid for the auction; (b) bid considering her resources; (c) bid using resources from an existing collaboration; or (d) identify the need to start a new VO to provide the extra resources required. CONOISE uses cumulative scheduling based on a Constraint Satisfaction Program (CSP) to model an agent's decision process.

CONOISE also focuses on the operation and maintenance phases of VOs. After the formation of a VO, CONOISE uses principles of coalition formation to distribute tasks among agents. Dang [42] presented and evaluated an algorithm for coalition structure generation. Although not very focused on authorisation issues, CONOISE provides reputation and policing mechanisms to ensure minimum quality of service, thus dealing with issues regarding VO trust and reputation.

TrustCoM Project: TrustCoM [49] addresses issues related to the establishment of trust throughout the VO life cycle. Its members envision that the establishment of Service Oriented Architectures (SOAs) and the dynamic open electronic marketplaces will allow dynamic alliances and VOs among enterprises to respond quickly to market opportunities. Hence, it is important to establish trust not only at a resource level, but also at a business process level. TrustCoM aims to provide a framework for trust, security, and contract management to enable on-demand and self-managed dynamic VOs [49, 179].

The framework extends VO membership services [179] by providing means to:

- Identify potential VO partners through reputation management;
- Manage users according to the roles defined in the business process models that VO partners perform;
- Define and manage SLA obligations on security and privacy; and
- Enable the enforcement of policies based on SLAs and contracts.

From a corporate perspective, Sairamesh *et al.* [159] provide examples of business models for the enforcement of security policies and the VO management. As the goal is to enable dynamic VOs, TrustCoM focuses on security requirements for the establishment of VOs composed of enterprises. TrustCoM has performed studies and market analyses to identify the main issues and requirements to build a secure environment in which VOs form and operate.

Facilitators or Breeding Environments

In order to address the issue of trust between organisations, projects have created federations and consortiums which physical organisations or Grids can join to start VOs based on common interests. This section describes the main projects in this field and explains some of the technologies they use.

Open Science Grid (OSG): OSG [144] is a facilitator for VOs. The project aims to form a cluster, or consortium, of organisations and recommends they follow a policy that states how collaboration occurs and how to form a VO. To join the consortium and consequently form a VO, it is necessary to have a minimum infrastructure and preferably use the Grid middleware suggested by OSG. In addition, OSG provides tools to monitor existing VOs. OSG facilitates the formation of VOs by providing an open-market-like infrastructure, where the consortium members can advertise their resources and goals and establish VOs. VOs are recursive and hierarchical; thus VOs can comprise sub-VOs.[11]

The basic infrastructure required to form a VO includes a VO Membership Service (VOMS) [2] and operational support, primarily to provide technical support services for requests from member sites. As OSG intends to federate across heterogeneous Grid environments, the resources of the member sites and users are organised in VOs under the contracts resulting from negotiations among the sites, which have to follow the consortium's policies. Such contracts are defined at the middleware layer, where negotiation can be automated. However, thus far, there is no easily responsive means to form a VO, hence the formation requires complex multilateral agreements among the involved sites.

OSG middleware uses VOMS to support authorisation services for VO members, hence assisting in the maintenance and operation phases. Additionally, for the sake of scalability and ease of administration, Grid User Management System (GUMS) facilitates the mapping of Grid credentials to site-specific credentials. GUMS and VOMS provide means to facilitate the authorisation in the operation and maintenance phases. GridCat provides maps and statistics on running jobs and storage capacity of the member sites; information that can guide schedulers and job submission brokers, and in turn facilitate the operation phase. Additionally, MONitoring Agents using a Large Integrated Services Architecture (MONALISA) [115] monitor computational nodes, applications, and network performance of the VOs within the consortium.

Enabling Grids for E-sciencE (EGEE): Similarly to OSG, EGEE [65] federates resource centres to provide a global infrastructure for researchers. EGEE's resource centres are hierarchically organised: an Operations Manager Centre (OMC) located at the European Organisation for Nuclear Research (CERN), Regional Operations Centres (ROCs) located in different countries, Core Infrastructures Centres (CICs), and Resource Centres (RCs) responsible for providing resources to the Grid.

A ROC carries out activities such as supporting deployment and operations, negotiating SLAs within its region, and organising Certification Authorities (CAs). CICs are in charge of providing VO services such as maintaining VO servers and registration; VO specific services such as databases, resource brokers and user interfaces; and other activities such as accounting and resource usage. The OMC interfaces with international Grid efforts, and is responsible for activities such as approving connection with new RCs, promoting cross-trust among CAs, and enabling co-operation and agreements with user communities, VOs and existing national and regional infrastructures.

As well as installing the LCG-2/gLite Grid middleware [57], a formal request is needed to join EGEE, with further assessment by special committees. A VO is formed if the application is suitable for EGEE. Accounting is based on the use of resources by members of the VO.

[11] More details about the VOs enabled by OSG are available at the consortium's blueprint document [135].

Other Important Work

Resource allocation in a VO depends on several conditions and rules. The VO can contain physical organisations under different, sometimes conflicting, resource usage policies. Participating organisations provide their resources to the VO through contracts and agree to enforce VO level policies, which state who can access the resources within the VO. Different models can be adopted for negotiation and contract enforcement. One model relies on a trusted VO manager, where resource providers supply resources to the VO according to contracts established with the VO manager, who assigns resource quotas to VO groups and users based on a commonly agreed VO-level policy. Alternatively, a VO can follow a democratic or P2P sharing approach, in which "you give what you can and get what others can offer" or "you get what you give" [192].

Elmroth and Gardfjäll [59] presented an approach that uses a scheduling framework to enforce policies to enable Grid-wide fair-share scheduling. The policies are hierarchical in the sense that they can be subdivided recursively to form a tree of shares, yet they are enforced in a flat and decentralised manner. In the proposed framework, resources have local policies that reference the VO-level policies, which split the available resources to given VOs. While a centralised scheduler is not required, the proposed framework and algorithm depend on locally caching global usage information.

2.7 Requirements, Analysis, and Positioning

This section first presents the requirements for resource sharing among computational Grids, before classifying the work described earlier, and providing an analysis of the examined systems and the requirements they meet. After that, this section positions the thesis in regards to existing work and the requirements for resource sharing.

2.7.1 Requirements for Resource Sharing between Grids

To achieve resource sharing between Grids, a solution should meet the following requirements:

1. **Interoperability:** efforts on interoperability between Grid middlewares provide the basis for load management and resource sharing across Grids by facilitating job submission or request redirection.

2. **Interface with existing GRMSs:** existing Grids have resource management systems in place, which are the result of several years of software engineering and development. A solution that does not consider current LRMSs and GRMSs may discourage system administrators utilising it.

3. **Provide a general infrastructure and dynamic collaborations:** the interconnection of Grids must provide research communities with a general infrastructure for deploying applications, which enables more dynamic and prompt collaborations among interconnected Grids. Analogously to large-scale testbeds, a research community should be able to request a "slice" of the infrastructure on demand.

4. **Decentralisation:** a solution for resource sharing between Grids should not rely on centralised architectures, as is not always clear who is the responsible for maintaining centralised components in a multiple-Grid scenario. In addition, a centralised approach is vulnerable to a single point of failure.

5. **Respect administrative separation:** this requirement relates to decentralisation. Some of the existing large-scale testbeds are consortiums where a centralised entity performs the resource allocation. As Grid computing aims to allow providers to retain ultimate control over the resources they offer to the Grid, Grids should retain ultimate control over the resources they provide to other (interconnected) Grids, while respecting the needs of their internal Grid users.

6. **Deal with resource contention:** ideally, for a Grid *A* to provide resources to another Grid *B*, Grid *A* should have minimum guarantees over the resources providers are offering before agreeing to accept requests from Grid *B*. These guarantees are difficult to obtain because there is contention for resources between providers' local users and Grid users. Resource sharing between Grids introduces another level of resource contention: between users of the local Grid and users from other Grids. A solution for resource sharing between Grids should deal with these contentions.

7. **Provide incentives:** resource sharing between Grids must provide incentives to the interconnected Grids; otherwise discouraged to share resources. Grids can have incentives in terms of:

    - Improved application performance;
    - Increased geographic reach-ability; and
    - Reduced infrastructure costs.

2.7.2 Analysis of Existing Work

This section classifies the existing work investigated earlier. The tables presented consider a subset of the investigated systems. Then, the section presents an analysis of existing work and a discussion on what requirements they meet.

Table 2.2 classifies existing work according to their architectures and operational models. Gridbus Broker [187], GridWay [98], and the SNAP-based community resource broker [92] are resource brokers that act on behalf of users to submit jobs to Grid resources to which they have access. They follow the operational model based on job routing. GridWay enables various architectural models such as hierarchical and decentralised [99]. GridWay provides means to deploy virtual machines, in which the deployment takes place on a job basis [126]. DI-GRUBER [54], VioCluster [157], Condor flocking [67], and Platform's CSF [143] have distributed-scheduler architectures in which brokers or meta-schedulers have bilateral sharing agreements (Table 2.3). OurGrid [7] and Self-organising flock of Condors [27] utilise P2P networks of brokers or schedulers, whereas Grid federation [147] uses a P2P network to build a shared space utilised by providers and users to post resource claims and requests respectively (Table 2.3). VioCluster and Shirako enable the creation of virtualised environments on which the user can deploy job routing

Table 2.2: GRMSs according to their architectures and operational models.

System	Architecture	Operational Model
SGE and PBS	Independent clusters	Job routing
Condor-G	Independent clusters*	Job routing
Gridbus Broker	Resource broker	Job routing
GridWay	Resource broker**	Job routing***
SNAP-Based Community Resource Broker	Resource broker	Job routing
EGEE WMS	Centralised	Job routing
KOALA	Centralised	Job routing
PlanetLab	Centralised	N/A+
Computing Center Software (CCS)	Hierarchical	Job routing
GRUBER/DI-GRUBER	Distributed/static	Job routing
VioCluster	Distributed/static	N/A+
Condor flocking	Distributed/static	Matchmaking
Community Scheduler Framework (CSF)	Distributed/static	Job routing
OurGrid	Distributed/dynamic	Job routing
Self-organising flock of Condors	Distributed/dynamic	Matchmaking
Grid federation	Distributed/dynamic	Job routing
Askalon	Distributed/dynamic	Job routing
SHARP/Shirako	Distributed/dynamic	N/A+
Delegated Matchmaking	Hybrid	Matchmaking

* Condor-G provides software suitable for building meta-schedulers.

** GridWay is an organisation-centric broker that enables various architectural models such as hierarchical and decentralised.

*** GridWay also manages the deployment of virtual machines.

+ PlanetLab, VioCluster and Shirako use resource control at the containment level, but they also enable the creation of virtual execution environments on which systems based on job routing can be deployed.

or job pulling based systems. These last two systems control resources at the level of containment or virtual machines.

SGE, PBS, and Condor are included here for the sake of completeness, but in reality, they are LRMSs. The solutions based on resource brokers [98, 109, 187]: allow for the scheduling of jobs across organisations with heterogeneous Grid middleware; can work with existing GRMSs; and provide interoperability at the job submission level. However, there is still a need to preserve administrative separation. On one hand, user-centric brokers provide a decentralised architecture; on the other hand, they can exacerbate the problem of resource contention in interconnected Grids. In addition, existing resource brokers are unaware of agreements established between Grid infrastructures, which define the amount of resource one Grid can allocate from another. Job submission performed ignoring such agreements makes it difficult to enforce Grid-level resource usage policies.

Some systems based on federations of clusters [27, 67, 102, 143, 147, 157] present decentralised architectures and have attributes that are essential as GRMSs. However, during the conception of these systems, the need for supporting interconnection of Grids following the idea of administrative separation and dealing with resource contention in interconnections of Grids was not identified.

As described earlier, Shirako [102, 146] uses an architecture composed of a network of brokers that have information about the resources offered by providers. These brokers can exchange resources, which service managers use to deploy virtual machines. In this way, Shirako is able to provide a general infrastructure based on the abstraction of leases. Users utilise the resources they lease to deploy different types of applications. However, Shirako does not deal with two aspects: the mechanisms and policies that define how brokers exchange resources; and how resource providers estimate the utilisation of resources by their local users before agreeing to give resources to a broker, which means that it does not deal with the contention between providers' local users and Grid users.

Table 2.3 summarises the communication models and sharing mechanisms that systems based on distributed schedulers utilise. Shirako uses transitive agreements in which brokers can exchange claims of resources issued by site authorities that represent the resource providers. It allows brokers to delegate access to resources multiple times.

Table 2.3: Classification of GRMSs according to their sharing arrangements.

System	Communication Pattern	Sharing Mechanism
GRUBER/DI-GRUBER	Bilateral agreements	System centric
VioCluster	Bilateral agreements	Site centric
Condor flocking	Bilateral agreements	Site centric
OurGrid	P2P network	System centric
Self-organising flock of Condors	P2P network	Site centric
Grid federation	Shared space	Site centric
Askalon	Bilateral agreements	Site centric
SHARP/Shirako	Transitive agreements	Self-interest
Delegated MatchMaking	Bilateral agreements	Site centric

The sharing mechanisms utilised by these systems based on distributed-schedulers can derive the incentives required by participating clusters. Bilateral agreements and a P2P network, for example, can define the conditions under which the clusters exchange resources. As described earlier, however, interconnection of Grids was not envisaged during the design of these systems. Thus, resolving situations of dispute for resources between Grids was not crucial.

Table 2.4 summarises the resource control techniques employed by the investigated systems. As noted earlier, VioCluster, Shirako and PlanetLab use containment based resource control, whereas the remaining systems utilise the job model. Control at the containment level (*i.e.* using virtual machines) makes more general execution environments possible because they allow for the customisation of the resources allocated. However, as described earlier, virtual machines and batch jobs can co-exist; the former can produce

Table 2.4: GRMSs according to their support for VOs and resource control.

System	Support for VOs	Resource Control
EGEE WMS	Multiple VO	Job model
KOALA	Single VO	Job model
GRUBER/DI-GRUBER	Multiple VO	Job model
VioCluster	Single VO	Container model/multiple site*
Condor flocking	Single VO	Job model
OurGrid	Single VO	Job model
Self-organising flock of Condors	Single VO	Job model
Grid federation	Single VO	Job model
Askalon	Single VO	Job model
PlanetLab	Multiple VO**	Container model/multiple site
SHARP/Shirako	Multiple VO***	Container model/multiple site+
Delegated MatchMaking	Single VO	Job model

* VioCluster supports containment at both single site and multiple site levels.

** The slices provided by PlanetLab are analogous to VOs here.

*** Shirako enables the creation of multiple-site containers suitable for hosting multiple VOs [146], even though it does not handle issues on job scheduling among multiple VOs.

\+ Shirako supports containment at both (i) single site level through Cluster on Demand [38] and (ii) multiple-site level. Shirako also explores resource control at job level by providing recommendations about the site on which jobs should execute [146].

greater levels of performance isolation, whereas the latter poses fewer overheads.

The support of various systems for the VO life cycle phases is depicted in Table 2.5. We select a subset of the systems investigated in this thesis, particularly the work that focuses on VO related issues such as their formation and operation. As shown in Table 2.4, EGEE WMS [185] and DI-GRUBER take into account the scheduling of jobs according to the VOs to which users belong and the shares contributed by resource providers. The other systems enable the formation of a single VO wherein the control of jobs can take place on a user basis.

DI-GRUBER and gLite schedule jobs by considering the resource shares of multiple VOs. EGEE and OSG also work as facilitators of VOs by providing consortiums to which organisations can join and start VOs (Table 2.6). However, the process requires the establishment of contracts between the consortium and the physical resource providers. Shirako enables the creation of multiple-provider virtualised environments on which multiple VOs can be hosted [146].

The systems' characteristics and the VOs they enable are summarised in Table 2.6. Conoise [140] and Akogrimo [195] allow the formation of dynamic VOs in which a user utilising a mobile device can start the VO. Resource providers in Shirako can offer their resources in return for economic compensation. This means that the resource providers could not have a common target in solving a particular resource challenge, making the

Table 2.5: Support to the phases of the VO's lifecycle by the projects analysed.

Project Name	Support for the phases of the VO life cycle				Support for short term collaborations
	Creation	Operation	Maintenance	Dissolution	
OSG*	Partial	Partial	Not available	Not available	Not available
EGEE/gLite*	Partial	Available	Not available	Not available	Not available
CONOISE	Available	Available	Available	Not available	Available
TrustCoM	Partial**	Partial**	Not available	Not available	Not available
DI-GRUBER	Not available	Available	Partial***	Not available	Not available
Akogrimo+	Partial	Partial	Partial	Partial	Partial
Shirako	Not available	Available	Available	Not available	Not available

* OSG and EGEE work as consortiums enabling trust among organisations and facilitating the formation of VOs. They also provide tools for monitoring the status of resources and job submissions. EGEE's WMS performs job scheduling taking into account multiple VOs.

** Mainly related to security issues.

*** DI-GRUPER's policy decision points allow for the re-adjustment of the VOs according to the current resource shares offered by providers and the status of the Grid.

+ Akogrimo aims to enable collaboration between doctors upon the patient's request or in case of a health emergency.

VOs non-targeted. A virtual environment adapts by leasing additional resources or terminating existing leases according to the demands of the VO that the environment is hosting [146].

Each solution focuses on specific aspects of the VO life cycle. Some projects aim to facilitate the formation of VOs through the establishment of consortiums. In a general manner, physical organisations join the consortiums to be part of a VO. We argue that such an approach has contributed to the isolation of Grids, thus, we propose that Grids should provide a general infrastructure upon which communities can create VOs on demand. Drawing an analogy to PlanetLab, VOs should be "slices" of the interconnected Grids. This scenario still requires the techniques proposed for the formation, management and termination of VOs described earlier.

Inter-operation Efforts

As noted earlier, there have been initiatives for interconnecting Grids and federating large-scale testbeds [85, 142, 152], and other attempts to make different Grid middleware interoperable [190]. These efforts provide the basis for load management across Grids by facilitating standard job submission and request redirection.

Inter-operability between different Grid middlewares is important, however, previous work has shown that mechanisms to enable resource sharing are equally essential to interconnect Grids [152]. While standard interfaces and adaptors for job submission and resource discovery are also important, there is a need for mechanisms that use these interfaces. This thesis attempts to build on inter-operability efforts and investigate mechanisms to enable the provisioning of resources from multiple Grids to applications.

Table 2.6: Mapping of the systems against the VO taxonomies.

System	Dynamism	Goal Orientation	Duration	Control	Policy Enforcement	Facilitators
Conoise*	Dynamic/Hybrid	Targeted	Medium-lived	Decentralised	Democratic	N/A
TrustCoM**	Static	Targeted	Long-lived	N/A	N/A	N/A
GRUBER/DI-GRUBER	Static	Targeted	Long-lived	Decentralised	Decentralised***	N/A
gLite/EGEE	Static	Targeted	Long-lived	Centralised	Centralised	Centralised+
Open Science Grid	Static	Targeted	Long-lived	Hierarchical	Centralised	Market-like
Akogrimo*	Dynamic/Hybrid	Targeted	Short or Medium-lived	Decentralised	Democratic	N/A
Shirako	Dynamic	Non-Targeted	Medium-lived	Decentralised	Democratic	N/A

* Conoise and Akogrimo allow a client using a mobile device to start a VO; the VO can comprise fixed and mobile resources.

** TrustCoM deals with security issues and does not provide tools for the management and policy enforcement in VOs.

*** DI-GRUBER uses a network of decision points to guide submitting hosts and schedulers about which resources can execute jobs.

+ EGEE Workload Management System is aware of the VOs and schedules jobs accordingly to the VOs in the system.

2.7.3 Positioning of this Thesis

In order to meet the requirements enumerated in Section 2.7.1, this thesis proposes an architecture and provisioning strategies to enable the inter-operation of Grids. The architecture discussed in Chapter 3 is inspired by the manner in which Internet Service Providers (ISPs) establish peering arrangements in the Internet [123]. ISPs compete for users, but have incentives to interconnect and allow traffic into one another's networks. These benefits can include increased network coverage, reduction of traffic in other expensive links, improved quality of service for their users, and the likely increase in profits [12, 123].

An architecture for Grid computing based on peering arrangements can provide the incentives required to interconnect Grids. In addition, it can offer the interfaces with GRMSs currently utilised by the interconnected Grids. The proposed architecture introduces the idea of InterGrid Gateways that mediate access to the resources of the interconnected Grids. InterGrid Gateways also work as entry points to the Grids. In addition, the system implementation of the proposed architecture considers resource control at the containment level where resources allocated run virtual machines. That minimises some security concerns that are common in resource sharing between organisations and gives the possibility of building more general execution environments that users can customise according to their needs.

However, the requirements also demand mechanisms for resource sharing between Grids. This thesis presents a mechanism for resource sharing between Grids that uses resource availability information obtained from typical LRMSs. The mechanism redirects requests based on their marginal cost of allocation, and meets the requirements in terms of working with traditional LRMSs and minimising the contention for resources between users of a Grid and users from other Grids.

This thesis also considers that the resources allocated from the interconnected Grids are utilised by the users to deploy their applications. An InterGrid Gateway routes requests to a provider able to offer the required resources. However, for negotiation between InterGrid Gateways we do not consider the routing and migration of the jobs with all required input files. Once the InterGrid Gateway grants the request with resources, the user copies the input files to the target site.

The resource sharing mechanism derives from Medusa [14], but differs in terms of the: negotiation protocol for exchanging resources between Grids, resource selection and request redirection policies, and the heterogeneity of resources within Grids, given by the number of processors. The resource selection policies take into account the availability of resources within multiple clusters that in turn employ optimisations such as backfilling.

The scenario for applying the proposed resource sharing mechanism differs from the load-sharing scenario described in other work. For example, the scenario considered in this thesis is different from that by Wang and Morris [191] because the resources considered here have multiple processors. In addition, resources are heterogeneous in the number of processors, making the local scheduling sub-problem different. Resource management across Grids introduces a third sub-problem: the load sharing between Grids. Previous work has also introduced load balancing for DHT-based networks. Surana *et al.* [178] introduced a mechanism where nodes of the P2P system run virtual servers responsible for ranges of objects in the address space. They inform directories about the load in the virtual servers, whereas the directories periodically compute reassignments

and trigger migrations of virtual servers to achieve load balance. This thesis, in contrast, does not perform migration of virtual servers, but focuses on the redirection and assignment of resources to requests.

The resource sharing mechanism investigated in this thesis is different from those proposed for federation of clusters. It considers a higher-level hierarchy of the system entities wherein a Grid is a collection of resource providers and needs to provision resources to applications within the Grid. This is in itself a federation problem, but additionally a Grid provides resources to other Grids aiming not to sacrifice the performance of its users. In contrast to the Grid Federation Agents proposed by Ranjan *et al.* [148], as an example, InterGrid Gateways do not engage into negotiations if they can handle requests locally without increasing the resource utilisation cost above its threshold.

The proposed architecture is a hybrid of hierarchical and distributed. Providers within a Grid are organised in a hierarchical manner similarly to existing Grids. The gateways representing the Grids are organised in a decentralised way through a network of peering arrangements. Our architectural model shares aspects with Iosup *et al.*'s work [101], in the sense that InterGrid Gateways work as site recommenders matching requests to resources available. However, it is different with respect to the resource sharing mechanisms, the consideration of providers' local users, and the compensation of resource providers for the resources acquired by an InterGrid Gateway.

Market and Economics-inspired Mechanisms

The use of mechanisms inspired in economic principles comes from observing of how economies allocate resources. However, the advantages and performance of market-based mechanisms over other resource management approaches are generally overstated. Nakai and Van Der Wijngaart [129] made a critical analysis of the General Equilibrium (GE) theory and the applicability of markets to global scheduling in Grids. Considering that GE theory is a description of a rather special type of economy and does not consider factors such as externalities, they found that it fails to deal with mechanisms that make real economies work. The perfect competition that drives an economy under GE does not reflect that among organisations in real markets where concentration of market power can lead to imperfect competition and market failure. Moreover, economists tend to agree that markets can malfunction and fail, which is a situation that can arise in artificial markets such as those created to manage computational resources.

It is also argued that one well-designed market-based resource allocation mechanism provides incentives for participation by ensuring that all the actors in the system maximise their utility and do not have incentives to deviate from the designed protocol [44]. In reality, designing such strategy-proof mechanisms is difficult, and resulting mechanisms can allow participants with privileged information to disrupt or game the system in their benefit [122].

Furthermore, in addition to the dispute surrounding the self-organisation properties of markets, where they can acquire a functional structure or equilibrium without specific interference of a central planner [17], evidence has shown that in real markets, if ever reached, this equilibrium may lead to the impoverishment of large parts of the population [111]. Therefore, mechanisms designed for artificial markets may display similar malfunctioning and failure behaviours, thus undermining the allocation of the computer

system's resources.

In spite of these issues, market-based models for resource allocation can bring benefits to existing Grid infrastructures. As pointed out by Shneidman *et al.* [163], many computer systems have reached a point where the goal is not always to maximise utilisation; instead, when demand exceeds supply and not all needs can be met, a policy for making resource allocation decisions is required. Hence, market-based approaches are a good choice to carry out policy-directed resource allocation. In addition, economics-inspired mechanisms can allow resources to be quantified in terms of a common currency and exchanged by closed distributed systems belonging to different organisations. For example, OurGrid [7], described earlier in this thesis, uses a common currency (*i.e.* favours) that sites can use to account for the resources borrowed from one another.

It is also important to notice, as argued by Shneidman *et al.* [163], that when speaking about real world markets, economists are hardly given the opportunity to deploy a market or a whole economy, having to address issues of already existing economies. Previous work has proposed a number of approaches that use economic models to address resource usage and incentives in a Grid [10, 24, 28, 69, 70, 113]. It seems natural to consider mechanisms based on economic principles for the system presented in this thesis because it comprises multiple Grids, established by different communities that are heterogeneous in terms of goals, funding, priorities and quality of service requirements; economics-inspired mechanisms can encourage participants to use resources in a wise manner. Therefore, this thesis uses an economics-inspired mechanism, and evaluates system performance metrics such as response time of both Grid and providers' local jobs.

Resource Provisioning within Grids

As discussed earlier, two levels of contention for resources can arise in a scenario with interconnected Grids. The first level, also termed intra-Grid, refers to the dispute between the resource providers' local users and users of the Grid. The second level, termed inter-Grid, relates to the resource contention between Grid users and users from other Grids. The proposed architecture and resource sharing mechanism aim to meet the requirements of respecting administrative separation and dealing with resource contention at the inter-Grid level. To minimise the resource contention at the inter-Grid level, Grids must employ provisioning techniques to handle contention at the intra-Grid level. Addressing these two levels of resource contention is important in order to achieve resource sharing between Grids, as discussed earlier in Section 2.7 as Requirement 6.

However, as described beforehand, LRMSs commonly use optimisations such as job backfilling. While these optimisations maximise resource utilisation, they make it difficult to predict the resource availability over a given period as the jobs' start and completion times depend on the workloads. Existing work on mechanisms for sharing resources abstract the availability information obtained from LRMSs. For example, AuYoung *et al.* [11] have considered a scenario wherein service providers establish contracts with resource providers. They have modelled the resource availability as ON/OFF intervals, which correspond to off-peak and peak periods respectively. However, such models may not represent the LRMSs' load because it depends on the jobs' characteristics [118].

Therefore, this thesis investigates mechanisms that gather information about the resource availability from typical LRMSs. We evaluate the precision of this information

and its impact to check whether typical LRMSs can offer the information required for resource sharing between Grids. Furthermore, the thesis proposes incremental changes to scheduling mechanisms used by LRMSs to enable providers to provision resources to a Grid and respect their local users. The thesis introduces strategies based on multiple resource partitions, although extends existing multiple-partition approaches. It proposes a new multiple-resource-partition policy based on load forecasts for resource provisioning.

These techniques have similarities with previous work. Singh *et al.* [166, 168] presented a provisioning model where Grid sites provide information about the periods over which sets of resources would be available. They propose sites provide their resources to the Grid in return for payments, hence presenting a cost structure consisting of fixed and variable costs over the resources provided. The evaluation of the provisioning model considers the scheduling of workflow applications. The main goal is to find a subset of the aggregated resource availability, termed resource plan, to minimise both the allocation cost and the application makespan. They utilise a Multi-Objective Genetic Algorithm (MOGA) approach to approximate the group of resource plans that correspond to the *Pareto-optimal* set. The experiments consider one cluster and one broker at a time. The work in this thesis is different as it investigates multiple approaches to obtain availability information and the reliability of this information in multiple-site environments. In addition, it provides additional provisioning strategies based on multiple resource partitions.

In the scope of some of the systems discussed earlier, mechanisms for flexible advance reservations and generation of alternative time slots for advance reservation requests have been proposed [154, 196]. Such approaches are useful for resource provisioning, and the scenario described in this thesis can incorporate them to improve resource utilisation and generate alternative offers should resource contention occur. However, we aim to reduce the interaction between resource providers and InterGrid Gateways by allowing the providers to inform the gateways about their spare capacity. This thesis therefore focuses on how to obtain availability information from LRMSs and how reliable this information is under different conditions.

Other approaches consider resource provider sites that use virtualisation technologies to provide resources. For example, Padala *et al.* [137] apply control theory to address the provision of resources to multi-tier applications in a data centre. Garbacki and Naik [87] consider a scenario where customised services are deployed on virtual machines, which in turn are placed into physical hosts. The provisioning scenario in Grids is, however, different as it uses traditional queue-based LRMSs.

In regards to the use of virtualisation technology, this thesis has similarities with systems such as Shirako [102], VioCluster [157] and Virtuoso [164], but extends existing work. It extends Shirako's lease model by investigating techniques that allow resource providers to supply gateways with the resource availability information described earlier as a requirement for provisioning resources. Providing this availability information is similar to issuing resource tickets in Shirako, therefore this thesis also provides techniques that define how to create these resource tickets.

2.8 Conclusion

This chapter investigated and classified systems that enable resource sharing among organisations and that can provide means for inter-operating Grids. The chapter also described classifications on Virtual Organisations (VOs) with a focus on Grid computing practices. In addition, it discussed background and related work on job backfilling techniques, advance reservations, and scheduling using multiple resource partitions, which are essential for a better understanding of the topics discussed in subsequent chapters.

The chapter presented the requirements for resource sharing between Grids and positioned this thesis with regards to existing work. As discussed in this chapter, the interconnection of Grids presents itself with two instances of contention for resources. In the first instance, resources from providers are allocated to users within a Grid. Such provisioning should be performed without visible impact on the applications of users local to these resource providers. In the second instance, spare resources are shared between Grids, which in turn should not impact users local to these interconnected Grids.

In the next chapters we describe an architecture that enables resource sharing between Grids and related provisioning strategies, aiming to provide resources to Grid applications without impacting the performance of providers' local users.

Chapter 3

The InterGrid Architecture

As noted in Chapter 1, over the years several institutions and nations have created Grids to share resources. These Grids are generally tailored to the requirements of the scientific applications for which they have been created. In this chapter, we describe an architecture to enable resource sharing across Grids based on the concept of peering arrangements between Grids. This architecture provides the basis to create execution environments using resources from multiple Grids. This chapter also describes the challenges involved in provisioning resources from interconnected Grids to applications; topics explored in subsequent chapters.

3.1 Introduction

The resource sharing in current Grids follows organisational models based on the idea of Virtual Organisations (VOs) [82]; a VO is created for a specific collaboration and the resource sharing is limited to within the VO. In addition, the resource control and enforcement of resource allocation policies in Grid-enabled VOs follow a job abstraction, where users encapsulate their applications as jobs that are routed by schedulers to the resources where they finally run. The schedulers strive to enforce VO or Grid-wide resource allocation policies.

Grid computing does not follow principles of peering relationships or a structure of networks of networks, which are generally present in other networked systems such as the Internet [12, 123] and the World Wide Web [19]. This chapter presents an architecture for resource sharing between Grids inspired by the peering agreements established between Internet Service Providers (ISPs) in the Internet, whereby ISPs agree to allow traffic into one another's networks. The architecture, termed as InterGrid, relies on intermediate parties called InterGrid Gateways that mediate access to the resources of participating Grids. The chapter describes the underlying concept of Internet peering arrangements before introducing the architecture and requirements for interconnecting Grids.

3.1.1 The Internet

The Internet began in 1969 as a small Defense Advanced Research Projects Agency (DARPA) project that linked a few sites in the USA; growing to the millions of hosts and

networks that comprise its current intricate topology [112]. Today's Internet comprises numerous ISPs; companies that offer their customers access to the Internet. Figure 3.1 illustrates the interconnection between ISPs, with hosts connecting to ISPs through access networks. In dial-up or broadband services, the local Public Switched Telephone Network (PSTN) loop is usually utilised to give users access to the Internet. These local ISPs then connect to regional ISPs, which in turn, connect to national and international ISPs, commonly termed as National Service Providers (NSPs) or Tier-1 ISPs. Tier-1 ISPs represent the highest level of the Internet hierarchy and they are connected to one another either directly or through Internet Exchanges (IXs). In this way, ISPs can provide services such as Internet access, backbone, content, application, and Web hosting. This structure has allowed the Internet topology to grow quickly and without the endorsement of a central authority [112].

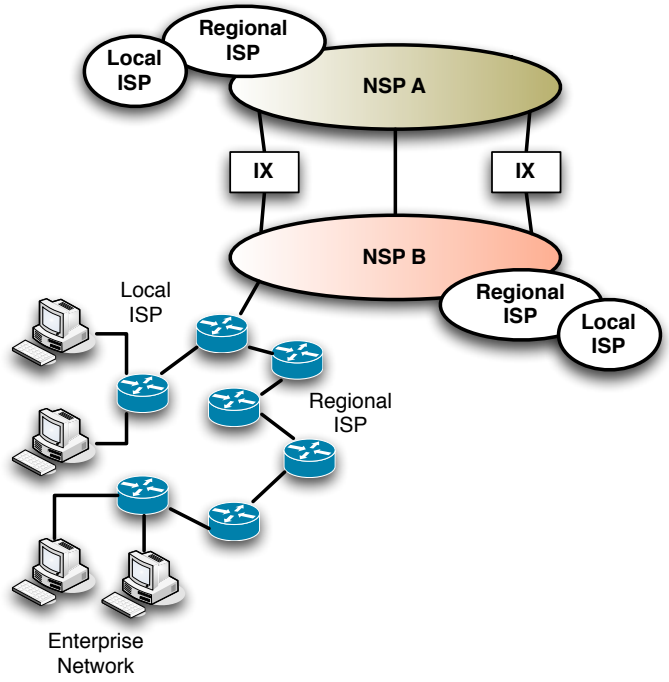


Figure 3.1: Interconnection of Internet service providers [112].

Currently, the Internet presents a structure composed of a vast number of physical network connections established as a result of commercial contracts such as peering agreements [123, 132]; legal contracts that specify the details of how ISPs exchange traffic. The reasons for peering involve social, economical, and technological factors, although ISPs need to consider their policies, economical advantages, and conflicts before establishing agreements [12]. There are various types of agreements, such as private, via IXs, or in a relationship between customer and provider. These agreements can specify the amount and proportion of traffic ISPs exchange, and the settlements since the Internet traffic exchanged can be asymmetric. Tier-1 providers – ISPs that have access to the global Internet – commonly establish non-charging contracts with other Tier-1 providers, but charge smaller ISPs under a peering arrangement.

Another important concept is that of an Autonomous System (AS) [94], or network under a single administrative domain with policies for diverting traffic, sometimes avoiding

peering ASs. Border Gateway Protocol (BGP) allows ASs to advertise preferred routes and enforce these traffic related policies. An AS can have policies incorporating shortest or most cost-effective paths [72, 73]. Such policy-based routing or peering could also be applicable to interconnected Grids, where a Grid can favour a peering Grid more than others. In summary, the Internet has a set of standard protocols that allow traffic to flow from one ISP to other ISPs. However, when ISPs interconnect their networks, they consider incentives that include the increase in network coverage, profit, and quality of service improvements.

Some important aspects to note about the structure of the Internet include:

- The Internet is a global network enabled by a common set of standard protocols that allow interoperability among networks with different technologies, physical network interconnections, and various peering arrangements.
- Although ISPs compete with one another, peering allows interconnected ISPs to provide global connectivity, reduce traffic across an expensive boundary, and improve the efficiency for their customers [12, 132]. In addition, its business model benefits end-users and compensates service providers [100].
- Routing protocols divert traffic under unfavourable conditions according to routing policies based on internal interests.

3.2 Peering Arrangements Between Grids

The Internet and Grids have a number of similarities. For example, both aim to provide infrastructure and standard protocols that users can utilise to build and run their applications, and both have user applications that would benefit from increased network coverage and greater resources. The cost reductions that ISPs derive from peering arrangements, through factors such as decreased traffic across expensive links, are similar to the benefits that Grids can derive from resource sharing and better resource management.

Considering that a Grid infrastructure is analogous to an ISP – both can be viewed as networks composed of smaller networks – the agreements that Grids would establish and the policies they would need to enforce to guarantee interoperation of Grids would be similar to those already studied and understood in the Internet community [12, 13, 22, 86, 100, 123, 132, 194]. The game of peering, which comprises the strategies that ISPs follow when their interconnect their networks, the viability of interconnecting, and the accounting mechanisms required by these peering agreements, can be sources of inspiration for deriving mechanisms for Grid interoperation.

We therefore argue that concepts applicable to ISP peering arrangements can be applied to Grid computing, and the rest of this chapter presents an architecture for connecting Grids inspired by the manner in which ISPs establish peering arrangements.

3.3 Research Challenges

In recent years, the distributed computing realm has shifted significantly. Grid computing has matured and commercial providers have emerged, offering resources to users in a

pay-as-you-go manner [150, 193]. Similar to the Internet, resource providers are profit-motivated; viewing each other as competitors or sources of revenue. Users, on the other hand, are interested in using services at low price with good performance.

Furthermore, resource providers and consumers in different Grids can have different, sometimes conflicting, interests. Hence, economy based models are relevant to interconnections of Grids because resource allocation can be achieved through the economic behaviour of involved parties. Markets can provide incentives for providers to offer their resources to Grids, and the settlements necessary between Grids. In this sense, some research challenges described here have a certain focus on economic based policies and mechanisms for interlinking Grids.

3.3.1 Co-ordination Mechanisms

Current resource allocation and scheduling mechanisms utilised by Grids are not co-ordinated [20]. Different domains have their own resource brokers, schedulers, objectives and requirements, and in many cases these entities do not exchange information about their allocation decisions. Such divergent approaches can lead to scenarios with bad schedules and inefficient resource allocation. As Ricci *et al.* [151] argue, for multiple schedulers to co-exist in a large-scale testbed, they should co-ordinate their resource allocation decisions. Peering arrangements can achieve co-ordination between Grids, further enabling the execution of applications spanning multiple Grids. Additionally, co-ordination may imply co-allocation of resources; for example, if resources from multiple Grids are required to deploy a specific application, these resources may be needed at a very specific period in the future.

3.3.2 Peering Arrangements

Within the Internet standard protocols ensure interoperability and enable a host to send packets to any other Internet-connected host. Although ISPs interconnect, they place varying costs on routes and consider various criteria to carry out inter-ISP routing [72, 73, 123, 194]. Asymmetry of the Internet traffic between ISPs can result in agreements with settlements. Principles similar to the Internet's policy-based routing apply to interconnections of Grids in activities involving offloading and redirecting resource requests from one Grid to another. However, there are important differences between packet routing and redirection of Grid requests. While Internet routing has to consider only data packets, Grid interconnection involves managing resource allocation requests that could involve numerous attributes depending on the demands of user applications, thus adding greater complexity.

Moreover, a Grid infrastructure has its policies regarding how resources are allocated to users of that Grid and to peering Grids. For example, Grid *A* can provide a best-effort service to a peering Grid *B*, in which Grid *A* tries its best to provide the resources when Grid *B* needs to offload resource requests. Grid *A* expects some compensation from Grid *B* for the resources provided. Grid *A* can have a different contract with Grid *C* in which Grid *A* stipulates that it will provide a maximum 100 computing resources for no more than 60 hours per month. Grid *A* may expect an equal compensation in terms of computing resources from Grid *C*. These agreements and the settlements must be

considered when a Grid requests additional resources from peering Grids. It is therefore very important to investigate policies that define how much resource two interconnected Grids can exchange, and how these policies are enforced. Furthermore, it is essential to evaluate the impact of these peering agreements on the performance of Grid applications.

3.3.3 Integration of Accounting Systems

Operating systems and LRMSs normally have accounting systems to track resource usage and assist the enforcement of utilisation policies [95, 105]. Accounting in clusters commonly assumes a homogeneous environment where the sharing of usage records among different systems is hardly required. Heterogeneous systems, such as Grids, present additional requirements, which include a standard format for exchanging basic accounting and usage data [119].

Previous work [16, 160] has proposed Grid accounting systems where users are linked to accounts on a bank and are assigned credits, or other form of currency, that correspond to their allowed resource usage. Resource utilisation is mapped to credits, which are deducted from the amount in the user's account upon resource usage. Other systems have mechanisms that allow sites to maintain usage information locally in the form of a common currency within the Grid (*e.g.* favours) [7].

The interconnection of Grids might require the integration of accounting systems, where Grids would need to agree upon the measurements for resource usage. The scenario would further need the definition of a form of currency or credit common to all Grids or a platform to promote the use of resources across Grids through exchange rates, thus allowing each Grid to have its own local currency [24].

3.3.4 Pricing Schemes

The use of economic models in Grids for regulating the allocation of resources, goods, and the use of services comes from observing the role of economic institutions in the real world [24, 70, 113, 197]. However, the use of economic approaches requires providers to price the provision of Grid resources, and users to express their resource demands in terms of currency units; both of which have proven difficult. Therefore, if an economic approach is used for sharing resources between Grids, studies are necessary in areas such as resource pricing, modelling consumer's utility, resource provider's marginal cost of allocation, and the benefits of providing resources.

3.3.5 Deployment of Applications across Grids

A multiple-Grid scenario requires application models capable of adapting to the environment's dynamism. Applications can run on execution environments that span multiple Grids and grow and shrink in terms of resource allocations. However, the flexibility of allocations is dependent on factors such as cost, time and overhead for changing allocations, and underlying peering arrangements.

A software system is needed that allows the creation of execution environments for various applications on top of the physical infrastructure provided by the participating Grids. Peering arrangements established between the participating Grids can enable the

allocation of resources from multiple Grids to fulfil the requirements of the execution environments. However, this scenario identifies various requirements for an architecture, mechanisms and peering strategies that allow:

1. Grids to interconnect and provide resources to one another as required;
2. Resource provision and sharing between Grids;
3. Application models that cope with the dynamics of interconnected Grids; and
4. Execution environments that span multiple Grids.

This chapter presents an architecture that aims to meet these requirements. The rest of the thesis focuses on the second requirement in the list above, by providing mechanisms for resource provisioning within Grids, and sharing between Grids. The requirements of mechanisms for resource sharing between Grids have been described in Section 2.7.

3.3.6 Problem Description and Propositions

We notice that the Internet concept of a network of networks is missing in Grid computing. In addition, the Internet aims at simplicity and providing a common set of protocols; while several of the existing Grid middleware have complex architectures. Benefits from peering, such as reducing traffic, increasing revenues or using services, are reasons adopted by ISPs for interconnecting their networks. The challenge is to identify architectural changes needed to enable peering arrangements between Grids in a way that they can employ different mechanisms and policies for inter-Grid resource sharing. Furthermore, these mechanisms should bring benefits to users of participating Grids, perceived in terms of improvements in the applications' performance.

Based on these characteristics, we define the following goals or propositions:

- Provide an architecture for interconnecting Grids that resembles the peering arrangements between ISPs. The architecture is described in the next section.
- With peering arrangements, define '*pluggable*'[1] mechanisms for resource provisioning and sharing to enable the deployment of applications across Grids. These mechanisms are investigated in subsequent chapters.

3.4 InterGrid Architecture

This thesis proposes an architecture termed as the *InterGrid* to enable the deployment of applications across multiple Grids. The InterGrid is inspired by the peering agreements between ISPs. Figure 3.2 shows the main components for the InterGrid. The architecture is a hierarchy with InterGrid Gateways (IGGs) on top co-ordinating resource sharing

[1] We use the term pluggable to represent a mechanism or protocol used as a plug-in. That is, it should be possible to change the resource allocation scheme of the management system by simply plugging in different software that implements the new scheme.

between different Grids, followed by the Grid Resource Management Systems (GRMSs) taking care of resource allocation using resource shares assigned by resource providers.

The InterGrid can enable peering of Grids and resource sharing. An application can demand resources from different Grids and perform resource management of its own. However, we argue that applications can have performance and environment isolation provided by Distributed Virtual Environments (DVEs) that are deployed on top of the InterGrid infrastructure [1, 156, 164]. DVEs provide users with a transparent network that behaves like a dedicated computing and data infrastructure, requiring little or no changes in existing Grid middleware and services. DVEs can span multiple Grids, and grow or shrink in terms of resource consumption according to the demands of the applications they encapsulate [158, 176, 177].

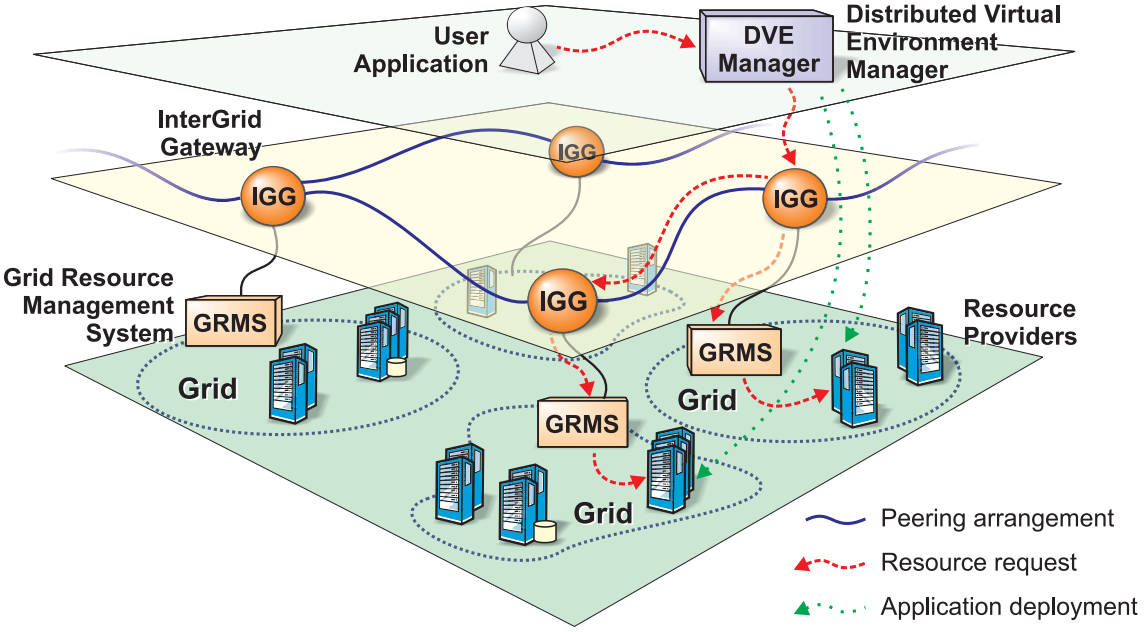


Figure 3.2: Architecture for interconnecting Grids.

3.4.1 Architectural Components

The proposed architecture integrates the resource allocation, represented by the InterGrid, with the application deployment, performed through DVEs. This section presents details about the components of this architecture. Note that the names of some components are different from previous work in which the architecture was introduced [46]; the changes aim to guarantee consistency throughout this thesis.[2]

Resource Providers (RPs)

RPs contribute a share of computational resources, storage resources, networks, application services or other types of resources to a Grid. The allocated share is based on

[2]In previous work [46], a Grid Resource Management System (GRMS) was termed as an IntraGrid Resource Manager (IRM); and a User Application (UA) was called a Client Application (CL).

provisioning policies according to the providers' perception of utility. The owner of a cluster could allocate a share of the nodes in return for regular payments. An RP may have local users whose resource demands need to be satisfied, yet it delegates provisioning rights over spare resources to an InterGrid Gateway (IGG). This delegation is performed by providing information about the resources available as *free time slots*.[3] The resources provided can be physical or virtual resources such as virtual machines. The delegation of resources to an IGG can use a secure protocol such as Secure Highly Available Resource Peering (SHARP) [86].

Grid Resource Management System (GRMS)

A GRMS represents the resource manager employed by the Grid middleware. The GRMS is the point of contact for acquiring resources, and manages the different shares of resources offered by the RPs to the Grid. The GRMS provides the IGG with access to the resources managed through plug-ins developed to interface with the Grid middleware, enabling the different kinds of resource allocation policies (determined inside a Grid and managed by the GRMSs) and peering arrangements (determined outside a Grid and managed by the IGG) to be reconciled.

InterGrid Gateway (IGG)

A Grid has pre-defined peering arrangements with other Grids, managed by IGGs, and through which they co-ordinate the use of resources from the InterGrid.[4] An IGG is aware of agreements with other IGGs; acting as a Grid selector by selecting a suitable Grid able to provide the required resources, and replying to requests from other IGGs, considering its policies. IGGs with pluggable policies enable resource allocation across multiple Grids. An IGG is chosen and maintained by a Grid administrator based on internal criteria. As described later in Section 3.5, the IGG also interacts with other entities including Grid Information Services (GISs), resource discovery networks, accounting systems, and GRMSs within interconnected Grids. A GIS is located within a Grid and provides details about the available resources, with accounting systems providing information on shares consumed by peering Grids.

Key IGG functionality consists of:

- **Resource selection:** selects resources from peering Grid infrastructures according to agreements between Grids, or based on resource selection policies defined by the peering Grids. The IGG selects, negotiates with, acquires, and ranks resources from peering Grids.

- **Resource allocation:** exposes the resources from a Grid to other peering Grids based on the resource sharing policies. In addition, the resource allocation component accepts or refuses requests from peering Grids based on these policies, accounting information, and monitoring services.

[3] We present some techniques for obtaining information about free resources in Chapter 4.

[4] The assumption of pre-established arrangements is reasonable as current Grids need to reserve the network links and set up the resources required. An example includes the interconnection of Grid'5000 with DAS-3 and NAREGI. More information is available at http://www.grid5000.fr

- **Peering policies:** underlie resource selection and allocation among Grid infrastructures. These policies can be inspired by economic principles.

Distributed Virtual Environment Manager (DVE Manager)

We suggest that a user application (indicated as UA in Figure 3.2) can acquire resources from the InterGrid by requesting the DVE Manager for the instantiation of a DVE. The DVE Manager, on the user's behalf, interacts with the IGG in order to acquire resources. The DVE Manager also monitors the resources that join or leave the DVE, deploys the services required by the user, and drives the adaptation of resource allocations based on the user application's demands. DVEs can accommodate existing Grid and virtualisation technologies by enabling dynamic overlay networks on top of the InterGrid, whose topologies and allocations may change over time. The DVE Manager can deploy existing Grid technology on the resources acquired for a DVE. The Grid middleware can provide the information required by the DVE Manager to adapt a DVE's topology or change its allocations. For example, the size of a scheduler's queue can provide the information required to detect the demand for additional resources.

User Application (UA)

A user application utilises resources from the interconnected Grids. Two types of users can exist in the InterGrid: those who deploy applications on resources from multiple Grids (*i.e.* deployers) and users who utilise these applications (*i.e.* final users). Although typically one and the same, they need not be.

A user application can acquire a slice of the InterGrid. In this way, a user application can be a set of *application services* for which there are final users. In this case, the user interested in deploying the application specifies the resources necessary, the required application services, and possibly the Grid middleware that must be deployed on the DVE. The final users can utilise the application once it is deployed.

3.4.2 Resource Allocation Model

This section illustrates how a user obtains resources from the InterGrid for an application. While the application can implement its own resource management mechanism or utilise existing Grid resource brokers, we envisage that the application can request a DVE Manager to acquire resources from the InterGrid, create a DVE, deploy the required services, and manage the resources in the DVE. The resource allocation across Grids uses the abstraction of containers or virtual machines. The workflow for resource allocation in the proposed architecture is shown in Figure 3.3, and can be described as follows:

1. Periodically, an RP advertises resources as free-time slots in the registry provided by the IGG. The advertisement is made through a delegation of the provisioning rights over a set of resource shares.

2. A user shows interest in obtaining a number of resources to deploy an application. The client contacts the DVE manager, to which it provides a description of the required resources and the services to deploy.

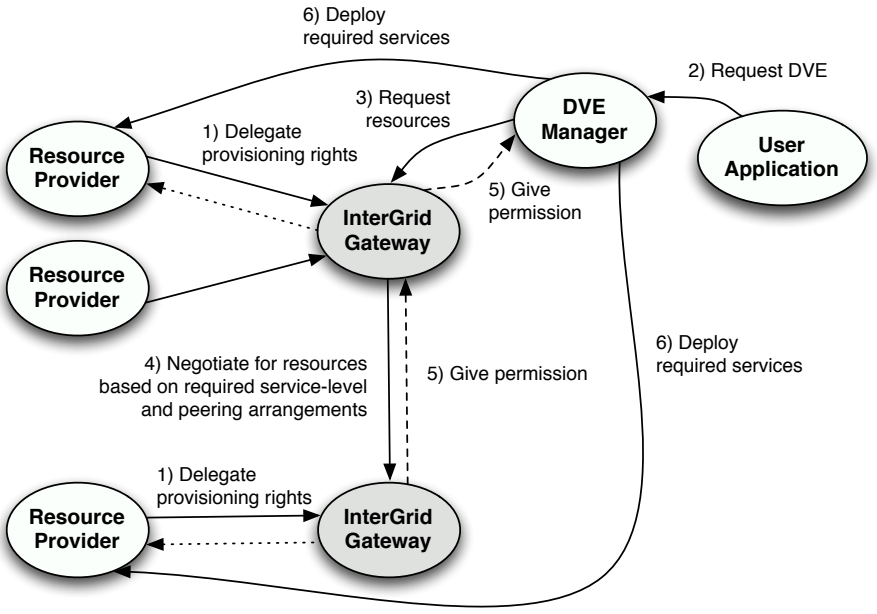


Figure 3.3: Resource allocation model for the proposed architecture.

3. The DVE Manager acquires resources, deploys the services, and manages the DVE composed of the allocated resources. The DVE manager tries to acquire the resources required by issuing requests to the IGG.

4. The IGG can provide all or part of the required resources based on the provisioning policies. If the individual Grid is not able to provide the required resources for performance or technical reasons, then the IGG selects a peering Grid in accordance with peering agreements and policies, from which the resources can be allocated.

5. Once the IGG allocates the resources, the DVE is given a permission to use them.

6. At the desired time, the DVE Manager transfers the permission to RPs, describing the share, the resources obtained, and the time duration of the permission. The DVE manager contacts the obtained resources and deploys the services on the user's behalf.

Once the resources are allocated and the services deployed, the final users can make use of the services and applications running on the DVE.

3.5 InterGrid Gateway Control Flow

This section presents the control flow within the IGG in detail. The IGG is the enabler for the InterGrid, a key component for managing the agreements between Grids and enabling co-ordinated resource allocation across Grids. The internal architecture of IGG is shown in Figure 3.4.[5] The control flow of this component is as follows:

1. RPs offer resources to a Grid by registering their availability with the IGG.

[5]The shaded components are studied in the remaining chapters.

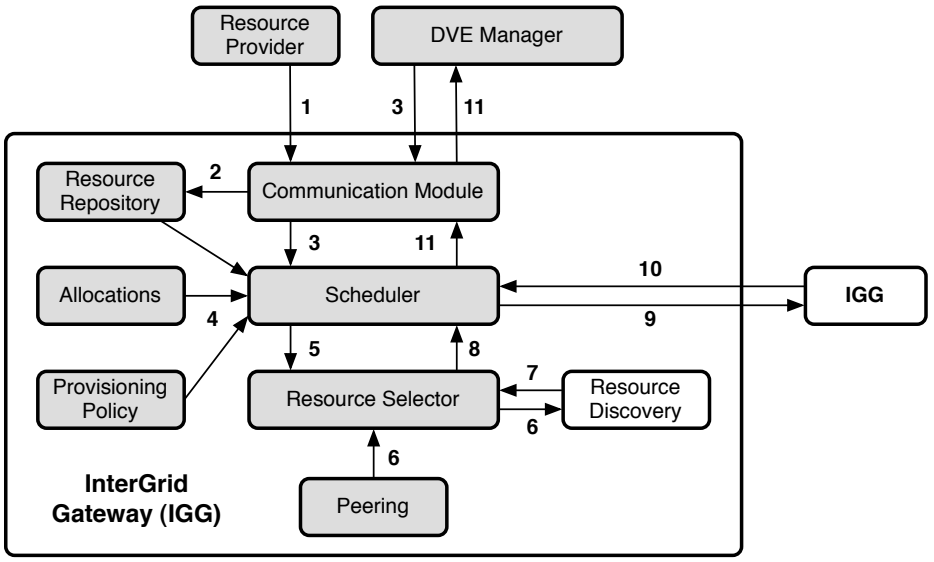


Figure 3.4: The InterGrid Gateway architecture.

2. The IGG stores the information about the resources in a *Resource Repository*.

3. Requests received by the IGG are passed to the *Scheduler*.

4. The *Scheduler* considers the availability of resources, the provisioning and admission control policies, and the current *Allocations* when making a decision on which resources to allocate to the DVE Manager.

5. If the current Grid does not have enough resources to provide, part or the whole request is passed to the *Resource Selector*, which searches for a peering Grid that can provide the required resources.

6. Descriptions of peering arrangements with other Grids, policies, and contracts are stored in a repository or maintained by a contract management system represented here by *Peering*. The *Resource Selector* utilises this data and resource information obtained from resource discovery networks established between the Grids to determine which peering Grid can provide the required resources.

7. Resource Discovery Networks (*i.e. Resource Discovery*), return a list of candidate Grids that can provide the required resources. The *Resource Selector* uses the stored policies and agreements, and the candidate resource list to determine which peering Grid can provide the resources.

8. The *Resource Selector* reports to the *Scheduler* once it finds a peering Grid.

9. The *Scheduler* then passes the resource request to the IGG selected.

10. The selected IGG returns either:

    - A list containing references to the resources it is willing to provide; or
    - A *reject* message if the Grid does not have the required resources available.

11. The Scheduler informs the DVE Manager about the list of resources allocated.

3.6 Resource Sharing and Provisioning Mechanisms

The previous sections have described the proposed architecture to enable resource sharing between Grids and some research challenges. However, due to the complexity of interconnecting Grids, this thesis focuses on two particular problems: resource sharing between Grids and resource provisioning within Grids. This thesis considers that IGGs have peering arrangements pre-established off-line requiring human intervention to configure aspects such as network links and access to shared infrastructures.

To function correctly, the architecture requires resource providers being able to provide information about resources that are available for the Grid, and resource provisioning mechanisms that permit the allocation of resources from the InterGrid to the execution environments. Therefore, the first issue to address is how information is obtained from providers such that resources can be safely provisioned to applications. Resource providers must be able to query their job schedulers and estimate which resources are available. With the information obtained from resource providers, IGGs can co-ordinate the use of resources within the Grids they represent. Moreover, mechanisms and policies that specify how interconnected Grids can share resources must be designed.

Furthermore, due to the increasing availability of commercial infrastructures, future Grids are likely to comprise both commercial and non-commercial resource providers. In this case, job scheduling and resource provisioning decisions must consider the cost of using commercial resources.

3.6.1 Resource Control Schemes

Current Grids generally use job abstraction to control resources, routing an encapsulated application as a job to the resource where it is executed. Server virtualisation technologies permit more flexibility in resource management by enabling performance isolation, and migration, suspension and resumption of virtual machines. It has also enabled resource control techniques where units of work are virtual machines on which jobs can execute.

However, current Grids utilise batch schedulers and other scheduling systems that use job abstraction. In the following chapters (Chapters 4 and 5), when we investigate the resource provisioning mechanisms we do not distinguish between requests for virtual machines and batch jobs, preferring to use the more general term *request*.

To realise the proposed architecture, we consider that the resource control between Grids can be performed using the abstraction of containers, as demonstrated by Ramakrishnan *et al.* [146]. Under this scenario, Grids exchange resources on which services are deployed according to their peering policies. The resources allocated by one IGG from another IGG are used to run virtual machines. *We have made this choice during the implementation of the system prototype as discussed in Chapter 7.*

3.7 Conclusion

This chapter described the InterGrid architecture and resource allocation model, which enables the provisioning of resources from multiple Grids to applications and Grids to connect with one another. The model is inspired by the way Internet Service Providers (ISPs) establish peering arrangements with one another in the Internet. The nature of these peering arrangements can vary. A flexible architecture based on InterGrid Gateways (IGGs) allows for changes of the provisioning policies used by the scheduler component.

To realise the goals of this architecture, in the next chapters we investigate strategies for provisioning resources within Grids, and introduce a mechanism for sharing resources between Grids. Furthermore, we present a provisioning model that allows the capacity of clusters participating in a Grid to be expanded by using resources from commercial providers.

Chapter 4

Multiple-Site Resource Provisioning

This chapter describes scheduling strategies that enable an InterGrid Gateway to obtain resource availability information from cluster resource managers to make provisioning decisions. Several emerging Grid applications are deadline-constrained, therefore require a number of computing resources to be available over a time frame, commencing at a specific time in the future. To enable resources to be provided to these applications, it is important to have a picture of cluster availability. Thus, the precision of the availability information is important as it may affect the applications' performance. Communication overheads may make requesting availability information upon scheduling every job impractical, therefore, in this chapter we investigate how the precision of availability information affects resource provisioning in multiple-site environments. We want to verify whether the proposed scheduling strategies based on multiple resource partitions can provide sufficient information to carry out resource provisioning in multiple-site environments. The performance evaluation considers both multiple scheduling policies in resource providers and multiple provisioning policies in the gateway, while varying the precision of availability information.

4.1 Introduction

As discussed in Chapter 2, the collaborations that Grids enable often require resources from multiple computing sites [82], which are generally clusters of computers managed by queuing-based Local Resource Management Systems (LRMSs) [25, 117, 188].

The execution environments considered in this thesis require provisioning of resources over well defined periods. Moreover, several emerging deadline-constrained Grid applications require access to several resources and predictable Quality of Service (QoS). A given application can require a number of computing resources to be available over a given period, commencing at a specific time in the future. However, given the LRMSs' current practice of optimising the First Come First Served (FCFS) policy, through techniques such as backfilling [128], to reduce scheduling queue fragmentation, improve job response time, and maximise resource utilisation, it is difficult to predict resource workloads and guarantee the start or completion times of applications currently in execution or waiting in the queue. Consequently it is difficult to predict resource availability over a period and provision resources to these time-constrained applications.

To complicate the scenario even further, users in a Grid may access resources via the Grid resource management system using mediators such as brokers or resource gateways. The design of gateways that can provision resources to deadline-constrained applications using information given by current LRMSs is complex and prone to scheduling decisions that are far from optimal. Moreover, it is not clear how gateways can obtain information from current LRMSs to provision resources to QoS-demanding applications without making resource provision just a consequence of scheduling.

Existing work on resource provisioning in Grid environments has used conservative backfilling, where the scheduling queue fragments, also termed *availability information* or *free time slots*, are given to a broker for provisioning [168]. However, the communication overheads may make requesting availability information upon scheduling every job impractical.

This chapter investigates how the precision of availability information affects resource provisioning in multiple-site environments (*i.e* within Grids). In addition, we enhance traditional schedulers to allow the gathering of availability information required by an InterGrid Gateway for resource provisioning. We evaluate the reliability of the provided information under various conditions by measuring the number of provisioning violations. A violation occurs when the information given by the resource provider proves to be incorrect when the gateway acts on it. We evaluate the impact of provisioning resources to Grid applications on providers' local requests by analysing the job bounded slowdown [74]. We also investigate whether aggressive backfilling [116] and scheduling strategies based on multiple resource partitions offers benefits over conservative backfilling if job backfilling is delayed, enabling large time slots to be provided to the gateway.

4.2 Multiple-Site Resource Provisioning

The multiple-site provisioning scenario is depicted in Figure 4.1 along with the entities considered in this chapter. This section describes the interactions between the entities whereas Chapter 3 provides details about the InterGrid architecture.

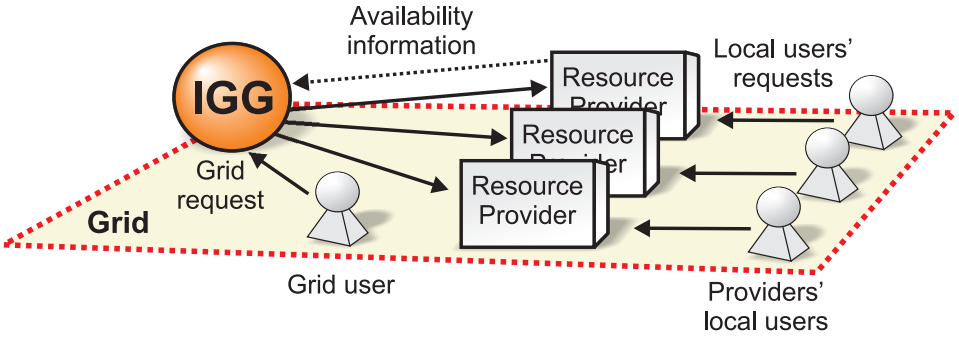


Figure 4.1: Resource providers contribute to the Grid but have local users.

A Resource Provider (RP) contributes a share of computational resources to a Grid in return for regular payments. An RP has local users whose resource demands need to be satisfied, yet it delegates provisioning rights over spare resources to the InterGrid Gateway (IGG) by providing information about the resources available in the form of

free time slots. A secure protocol, such as Secure Highly Available Resource Peering (SHARP) [86], could guarantee the authenticity and non-repudiation of these delegations. However, we focus on the resource provisioning aspect. A free time slot obtained by examining the status of the jobs in the scheduling queue, describes the:

- Number of resources available;
- Configuration of the processors/computing nodes available;
- Start time from which the processors would be free to use; and
- Fnish time from which the processors would no longer be available.

In the InterGrid, a Grid can have peering arrangements with other Grids managed by IGGs through which they co-ordinate the use of resources. In this chapter, however, we do not consider peering arrangements [46]; but rather investigate how an IGG can provision resources to applications based on the availability information given by RPs.

4.2.1 Problem Description and Propositions

An IGG attempts to provision resources to meet its users' needs, improve the job slowdown and minimise the number of violations. A violation occurs when a user tries to use the resources allocated by the IGG but they are no longer available due to wrong or imprecise availability information given by the resource providers. Resource providers are willing to increase the resource utilisation, including via IGGs, without compromising their local users' requests.

Therefore, we would like to specifically verify the following propositions:

- The use of free time slots information can allow for resource provisioning that provides better response time to Grid jobs.
- The use of aggressive backfilling with multiple resource partitions can allow for better provisioning and reduce a jobs' response time if the backfilling is delayed and larger free time slots are offered to the IGG.

4.2.2 Types of User Requests

We make a few assumptions about the types of requests that an IGG receives: a request is contiguous and needs to be served with resources from an individual provider, contains a description of the required resources and the time duration over which they are needed, and can demand either QoS or a best-effort service. A QoS constrained request has an *execution estimate*, a *deadline*, and a *ready time* before which the request is not available for scheduling, while a best-effort request has an execution time estimate but no deadline.

4.3 Provisioning Policies

We have extended traditional scheduling policies in order to obtain the availability information from resource providers in the form of free time slots. The policies utilise an 'availability profile' similar to that described by Mu'alem and Feitelson [128], which is a list whose entries describe the CPU availability at particular times in the future. These entries correspond to the start or completion times of jobs and advance reservations. A job *A* whose start time or completion time coincides with either the start or completion of another job *B*, may share entries with *B* in the profile.

Extending the idea of availability profile, we have implemented a profile based on a Red-Black tree that uses ranges of available processors/nodes rather than examining the availability of individual processors.[1] By scanning the availability profile and using load forecasts, the resource providers obtain the information about the free time slots and supply this information to the gateway. The gateway in turn can make provisioning decisions based on this information.

Although this thesis presents extensions to existing LRMSs for releasing resource availability information, throttling policies (*e.g.* those related the users' resource consumption) may make it difficult to obtain the free time slots. These policies commonly reflect users' or user groups' usage quotas and priorities, which derive from other factors such as research grants, an organisation's perception of urgency or importance, and characteristics of the user population. In addition, these policies may change from time to time. Therefore, there is certain complexity in announcing the free time slots available for a particular user. The existing literature on scheduling of parallel jobs often ignores these policies when modelling resources and evaluating scheduling algorithms because it is difficult to compare the performance of different algorithms as they may be favoured by specific throttling conditions. These issues can be circumvented by techniques such as those described for multiple-resource partitions in Section 4.3.2.

4.3.1 Conservative Backfilling Based Policies

Under conservative backfilling, a job is used to backfill and start execution earlier than expected, given that it does not delay any other job in the scheduling queue [128]. To reduce complexity, the schedule for the job is generally determined at its arrival and the availability profile is updated accordingly. Free time slots can be determined by scanning the availability profile. This approach, depicted in Figure 4.2, is also used by Singh *et al.* [166, 168].

With conservative backfilling, the availability profile is scanned until a time horizon, thus creating windows of availability, or free time slots; the finish time of a free time slot is either the completion time of a job in the waiting queue or the planning horizon itself. If the horizon is set to ∞, the provider will disclose all the information. The availability information can either be provided on a periodical basis, wherein provider and gateway agree on an interval at which the former sends the availability information, or upon the gateway's request. This thesis explores both scenarios.

[1] This extended profile is described in Appendix A.

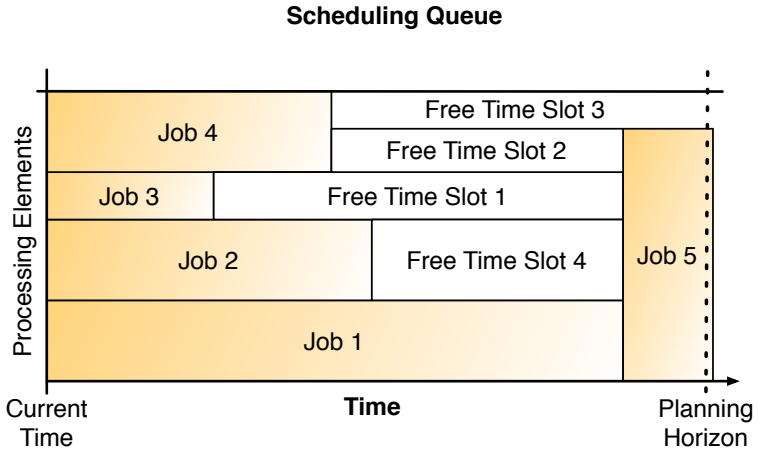


Figure 4.2: Obtaining free time slots under conservative backfilling.

4.3.2 Multiple Resource Partition Policies

This thesis presents three policies based on multiple resource partitions. These policies divide the resources available in multiple partitions and schedule jobs to these partitions according to predicates. The first policy we propose uses conservative backfilling, and one partition *A* can borrow resources from another partition *B* when they are not in use by *B* and borrowing is permitted by the scheduler.

The second policy implements the aggressive backfilling scheme described by Lawson and Smirni [114]. In this case, each partition uses aggressive backfilling and has a *pivot*, which is the first job in the waiting queue for that partition. A job belonging to a partition can start its execution if it does not delay the partition's pivot and the partition has sufficient resources. If insufficient resources, the job can still commence execution if additional resources can be borrowed from other partitions without delaying their pivots. Additionally, the policy uses priority scheduling where the waiting queue is ordered by priority when the scheduler is backfilling. To evaluate this policy, we attempt to maintain Lawson and Smirni's configuration [114] which selects partitions according to the jobs' runtimes. The partition $p \in \{1, 2, 3\}$ for a job is selected according to Equation 4.1, where t_r is the job's runtime in seconds.

$$p = \begin{cases} 1, & 0 < t_r < 1000 \\ 2, & 1000 \leqslant t_r < 10000 \\ 3, & 10000 \leqslant t_r \end{cases} \tag{4.1}$$

This thesis also introduces a new policy (*i.e.* the third option) depicted in Figure 4.3, where the scheduler resizes the partitions at time intervals based on load forecasts computed from information collected at previous intervals. In addition, the policy can alternate between aggressive backfilling and conservative backfilling. As load forecasts are prone to be imprecise, when the scheduler resizes partitions, it also schedules reallocation events. At a reallocation event, the scheduler evaluates whether the load forecast has turned out to be an underestimation; if underestimated, the policy resizes the partitions according to the load from the last resizing period and backfills the jobs, starting with the

local jobs. The policy uses aggressive backfilling with a configurable maximum number of pivots, similarly to the MAUI scheduler [104]. This allows the scheduler to convert the backfilling strategy to conservative by setting the number of pivots to a large value, here represented by ∞.

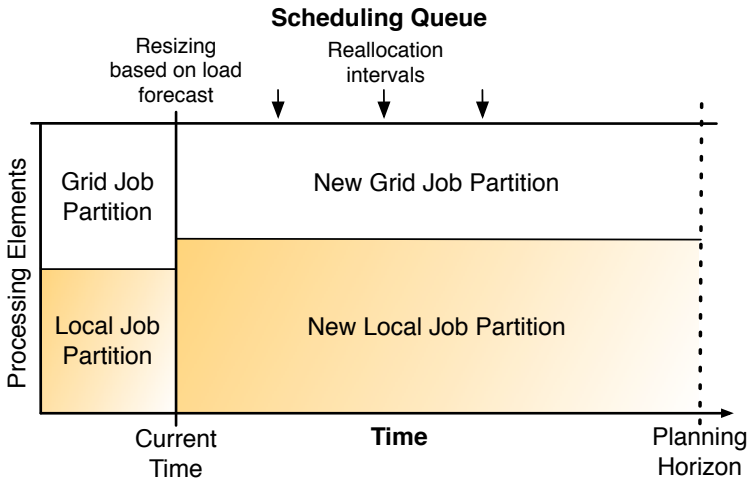


Figure 4.3: Obtaining free time slots using multiple resource partitions.

Algorithm 4.1 describes two procedures used by the load forecast policy. The policy invokes *getFreeTimeSlots* every time the provider needs to send the availability information to the gateway. Procedure *getFreeTimeSlots* schedules a later call to *reallocationEvent* to verify whether the previous forecast has turned out to be precise or if a reallocation is required.

Line 3 and 4 of Algorithm 4.1 change the scheduler's backfilling strategy to conservative by setting the number of pivots in each partition to ∞. They also schedule the jobs currently waiting in the queue. The algorithm then changes the scheduler's backfilling strategy to aggressive (line 5). From line 6 to 10, the scheduler obtains the load forecast and the free time slots and resizes the free time slots by modifying the number of CPUs according to the number of resources expected to be available over the next interval. Next, the scheduler triggers a reallocation event. From line 20 to 24 the scheduler verifies whether the forecast was underestimated. If so the scheduler turns its backfilling strategy back to conservative and informs the gateway about the availability.

4.3.3 IGG Provisioning Policies

The provisioning policies described earlier are suitable for resource providers to obtain the resource availability information and supply it to the IGG. Figure 4.4 depicts the information that the IGG has about the availability of resources from the providers. In order to store this availability information, the IGG uses a table of tree-based availability profiles. Appendix A presents details about the tree-based availability profile.

At the gateway level, the policies we consider are as follows:

- **Least loaded resource:** the gateway submits a job to the least loaded resource based on utilisation information sent by the resource providers every ten minutes.

Algorithm 4.1: Provider's load forecasting policy.

```
1  procedure getFreeTimeSlots()
2  begin
3      set the number of pivots of local and Grid partitions to ∞
4      schedule / backfill jobs in the waiting queue
5      set the number of pivots of local and Grid partitions to 1
6      actualLoad ← load of waiting/running jobs
7      forecast ← get the load forecast
8      percToProvide ← min{1 − forecast, 1 − actualLoad}
9      slots ← obtain the free time slots
10     slots ← resize slots according to percToProvide
11     if percToProvide > 0 then
12         inform gateway about slots
13         schedule reallocation event
14     schedule next event to obtain free time slots
15 end
16 procedure reallocationEvent()
17 begin
18     localLoad ← obtain the local load
19     forecast ← get the previously computed forecast
20     if localLoad > forecast then
21         set the number of pivots of local partition to ∞
22         schedule / backfill jobs in the waiting queue
23         set the number of pivots of Grid partition to ∞
24         schedule / backfill jobs in the waiting queue
25         slots ← obtain the free time slots
26         inform gateway about slots
27     else
28         schedule the next reallocation event
29 end
```

- **Earliest start time:** this policy is employed for best-effort and deadline-constrained requests when the resource providers are able to inform the IGG about the free time slots. When scheduling a job using this policy, the scheduler is given the job and provider's availability information. If the providers send the information at regular time intervals, this information is already available at the IGG. Otherwise, the IGG requests it from the resource providers. If the job is not deadline-constrained, the IGG selects the first provider and submits the job to it. When the job is deadline-constrained, the IGG attempts to make a reservation for it. If the provider cannot accept the reservation, the provider updates its availability information at the IGG.

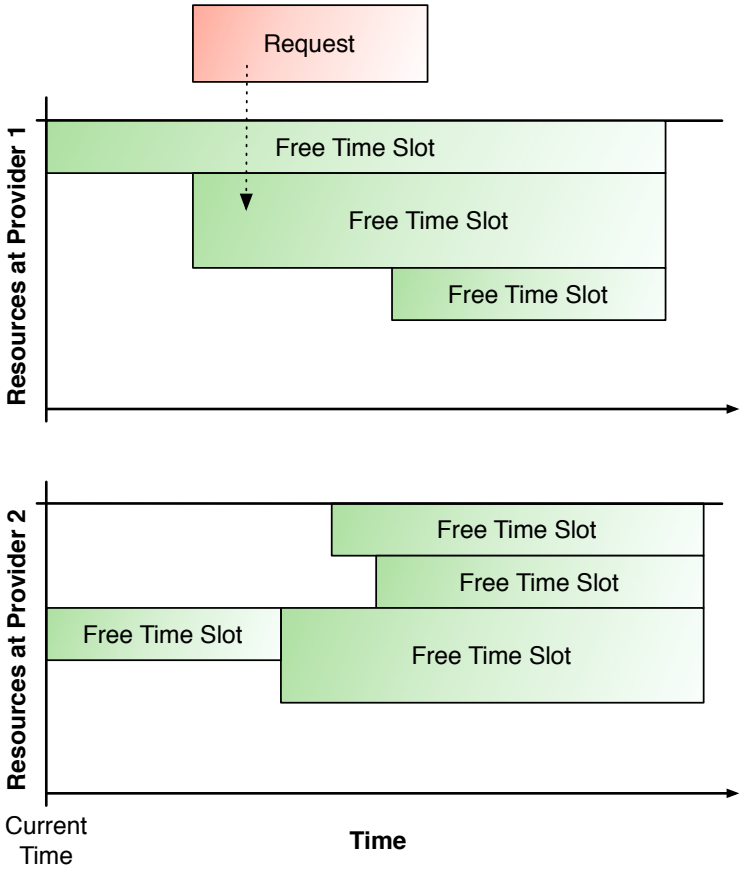


Figure 4.4: Selection of time slots by the InterGrid Gateway.

4.4 Performance Evaluation

4.4.1 Scenario Description

The experiments described in this chapter model the DAS-2 Grid [43] configuration because job traces collected from this Grid and its resources' configuration are publicly available and have been previously studied [101]. The multiple-site scenario simulated is depicted in Figure 4.5. DAS-2 is a Grid infrastructure deployed in the Netherlands comprising 5 clusters. The proposed techniques are evaluated through discrete-event simulation using GridSim, which has been extended to enable the scheduling of parallel jobs.[2] Simulations can provide a controllable environment that enables us to carry out repeatable experiments. In addition, as Grids are generally in use as testbeds or in production running scientific applications, it is difficult to gain administrative rights required to experiment with different scheduling strategies.

The resource providers' workloads have been generated using Lublin and Feitelson's workload model [118], hereafter referred to as Lublin99. Lublin99 has been configured to generate four-month-long workloads of type-less jobs (*i.e.* we do make distinctions between batch and interactive jobs). The maximum number of CPUs used by the generated jobs of a workload is set to the number of nodes in the respective cluster. Grid jobs' arrival

[2]More information available at: http://www.gridbus.org/intergrid/gridsim.html

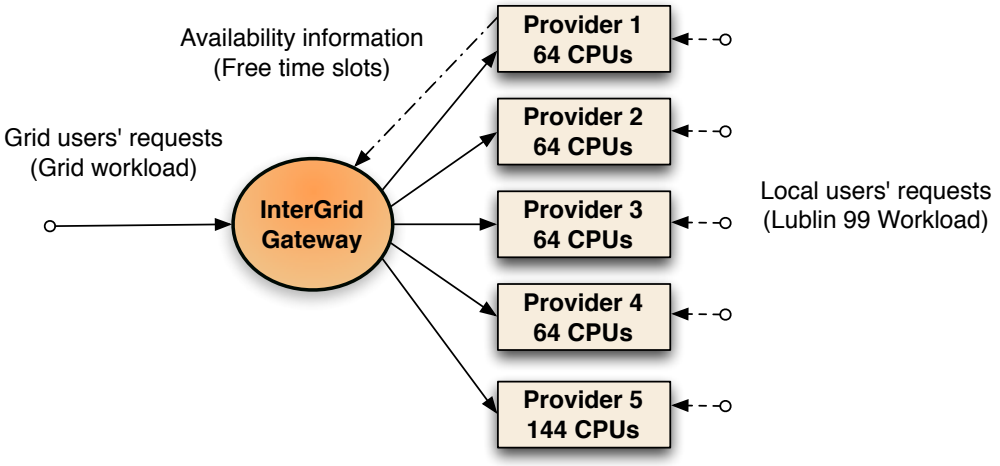


Figure 4.5: Multiple-site environment evaluated by the experiments.

rate, number of processors required and execution times are modelled using DAS-2 job trace available at the Grid Workloads Archive.[3,4] The jobs' runtimes are taken as runtime estimates. Although this may not reflect the reality, it provides the basis and bounds for comparison of scheduling approaches [74].

To eliminate the simulations' warm up and cool down phases from the results, we discard the first two weeks of the results and the last simulated event is the arrival of the last job submitted in any of the workloads. For the forecast-based policy, the second week is used as the training period. We randomly select the requests that are deadline-constrained. To generate the request deadlines we use a technique described by Islam *et al.* [103], which provides a feasible schedule for the jobs. To obtain the deadlines, we use the same Grid environment to perform the experiments using aggressive backfilling at the resource providers and 'submit to the least loaded resource' policy at the gateway. A request deadline is the job completion under this scenario multiplied by a *stringency factor*. The load forecasting uses a weighted exponential moving average [93], considering a window of 25 intervals.

Performance Metrics

One of the metrics considered is the bounded job slowdown ($bound$=10 seconds), hereafter referred to as job slowdown [74]. In fact, the experiments measure the bounded slowdown improvement ratio $\mathcal{R}$ given by Equation 4.2, where s_{base} is the job slowdown using a base policy used for comparison; and s_{new} is the job slowdown resulting from the evaluated policy. The experiments calculate the ratio $\mathcal{R}$ for each job and then take the average. The graphs presented in this section show average ratios.

$$\mathcal{R} = \frac{s_{base} - s_{new}}{min(s_{base}, s_{new})} \tag{4.2}$$

The experiments also measure the number of violations and messages exchanged be-

[3]Grid Workloads Archive website: http://gwa.ewi.tudelft.nl/pmwiki/
[4]Specifically, the experiments use the interval from the 9th to the 12th month.

tween providers and the IGG to schedule Grid jobs. The reduction in the number of messages required is used to estimate the trade-off between precision of information and communication overhead. A given job j faces a violation when the inequality $j_{pst} - j_{gst} > T$ is *true*, where j_{gst} is the job start time assigned by the gateway based on the free time slots given by providers; j_{pst} is the actual job start time set by the provider's scheduler; and T is a tolerance time. The experiments use a T of 20 seconds. A violation also occurs when a resource provider cannot accept a gateway's reservation request.

Policy Acronyms

We abbreviate the name of the evaluated policies in the following manner. A policy name comprises two parts separated by +. The first part represents the policy employed by the provider; the second is the IGG policy. On the resource provider's side, **Ar** stands for Advance reservation, **Eb** for aggressive backfilling, **Cb** for Conservative backfilling, **M** for Multiple partitions, and **Mf** for Multiple partitions with load forecast. For the IGG's policy, **least-load** means 'submit to the least loaded resource', **earliest** represents 'select the earliest start time', **partial** indicates that providers send free time slot information to the IGG on a periodical basis, and **ask** means that the IGG requests the free time slot information before scheduling a job. For example, **ArEbMf+earliest-partial** indicates that providers support advance reservation, aggressive backfilling, multiple resource partitions, and load forecasts; whereas the IGG submits jobs selecting the earliest start time based on the availability information sent by providers at regular intervals.

4.4.2 Experimental Results

The first experiment measures the number of messages required by the policies supporting advance reservation and conservative backfilling (*i.e.* ArCb). Some policies request the free time slots from providers and in others the time slots are informed by providers at time intervals. This experiment investigates whether the number of messages required can be reduced by making the resource providers publish the availability information at gateways at time intervals. The experiment varies the interval for providing the availability information, then measures the number of violations and average job slowdown to check the trade-off between the precision of scheduling decisions and the freshness of information. The planning horizon is set to ∞, hence a provider always informs all the free time slots available. In addition, advance reservations use a two phase commit protocol. The time interval for providing the time slots to the gateway is described in the last part of the name of the policies (*e.g.* 15 min., 30 min.). The stringency factor is 5 and around 20% of the Grid requests are deadline-constrained.

Figure 4.6a shows that for the policy in which the gateway asks for the time slots upon scheduling every job (*i.e.* ArCb+earliest-ask), the number of messages is larger compared to other policies. In contrast, policies that report the free time slots at regular intervals or when an advance reservation request fails, result in a lower number of messages.

The number of violations increases as providers send the availability information at larger intervals (Figure 4.6b). If scheduling is made based on information provided every 15 minutes, the number of violations is 973, which accounts for 0.43% of the jobs scheduled. To evaluate whether these violations impact on the resource provisioning for

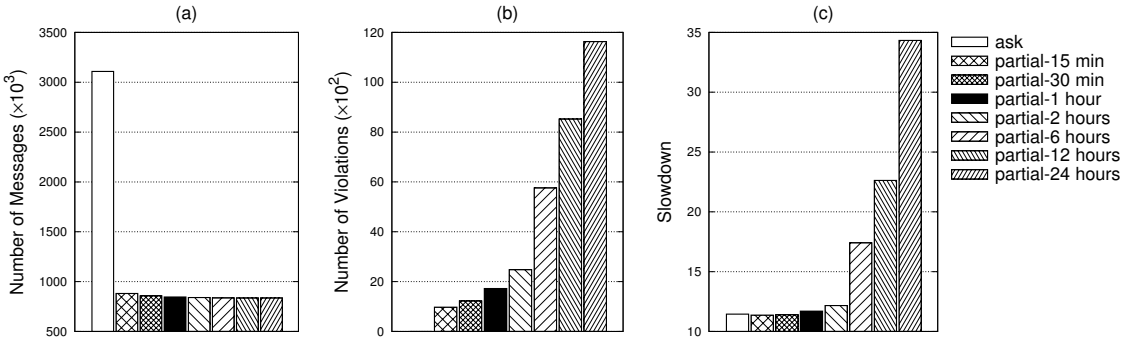


Figure 4.6: ArCb+earliest-* policies: (a) number of messages; (b) number of violations; and (c) average job slowdown.

Grid jobs, the experiments measure the average bounded slowdown of Grid jobs (Figure 4.6c). As shown in the figure, there is an increase in the job slowdown as the interval for providing the free time slots increases. However, when the providers send availability information every 15 minutes, the average slowdown is improved. We conclude that for a Grid like DAS-2, where providers send the availability information at intervals of 15 to 30 minutes, resource provisioning is possible using a simple policy supporting conservative backfilling.

The second experiment measures the average of jobs ratio $\mathcal{R}$ described in Equation 4.2. The values presented in the graphs of this section are averages of five simulation rounds, each round with different workloads for providers' local jobs. The set of policies used as the basis for comparison comprises aggressive backfilling in the providers and 'submit to the least loaded resource' in the IGG. The resource providers send the availability information to the gateway every two hours. This experiment does not consider deadline-constrained requests, as they could lead to job rejections by some policies, which would then impact on the average bounded slowdown.

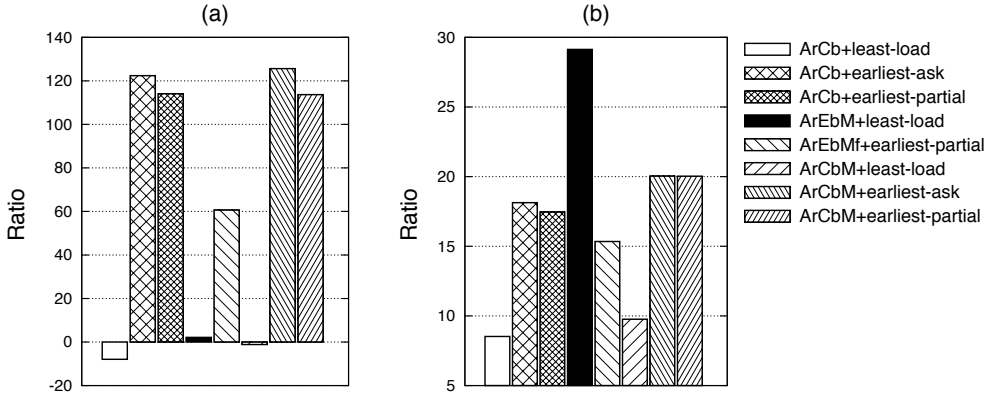


Figure 4.7: Slowdown improvement ratio: (a) Grid jobs and (b) providers' local jobs.

The results show that conservative backfilling and 'least loaded resource' policies (*i.e.* ArCb+least-load and ArCbM+least-load) tend to degrade the bounded slowdown of Grid jobs (Figure 4.7a). Submitting a job to the least loaded resource, where utilisation is computed by checking current CPU usage, does not ensure immediate start of the job

because other jobs in the waiting queue may have been already scheduled. Moreover, the gateway is not aware of the time slot the job will actually utilise.

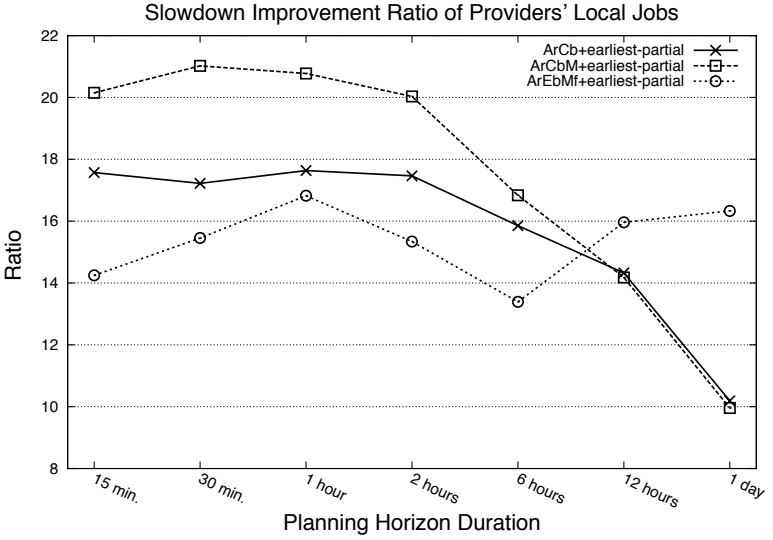


Figure 4.8: Slowdown improvement ratio of providers' local jobs.

The multiple resource partition policies with conservative backfilling without priorities and providing the free time slots to the gateway improve the average slowdown of both Grid jobs (Figure 4.7a) and providers' local jobs (Figure 4.8). ArEbM+least-load, proposed by Lawson and Smirni [114], improves the slowdown of local jobs (Figure 4.7b); but not the slowdown of Grid jobs as the original implementation of this policy gives higher priority to local jobs. The aggressive backfilling policy that resizes the resource partitions according to load estimates (*i.e.* ArEbMf+earliest-partial) improves the slowdown of both Grid jobs (Figure 4.7a) and providers' local jobs, but not as much as other multiple partition policies.

The last experiment varies the intervals for providing the free time slots. Figure 4.8 and Figure 4.9 show that for small planning horizons, the multiple resource partition policy with aggressive backfilling and load estimates (*i.e.* ArEbMf+earliest-partial) improves the average ratio, but not as much as the other policies. However, as the time interval for providing the availability information increases, the policy outperforms the other policies. For example, there is a sharp decrease in the ratios of the conservative backfilling policy (*i.e.* ArCb+earliest-partial) and the multiple partition aggressive backfilling policy (*i.e.* ArEbM+earliest-partial). The slowdown of the forecast-based policy improves compared to the other policies when the interval increases, possibly due the policy becoming multiple-partition with conservative backfilling when a load estimate is wrong. In a long interval, when an incorrect estimate is identified, there may be quite a lag before the policy resumes aggressive backfilling at the next interval; this conservative backfilling provides a better job slowdown. Updating availability in the middle of an interval may also provide an advantage over the other policies, however this has not been investigated during this thesis research. Better load forecast methods might further improve the jobs slowdown under varying intervals; again, not investigated during this research.

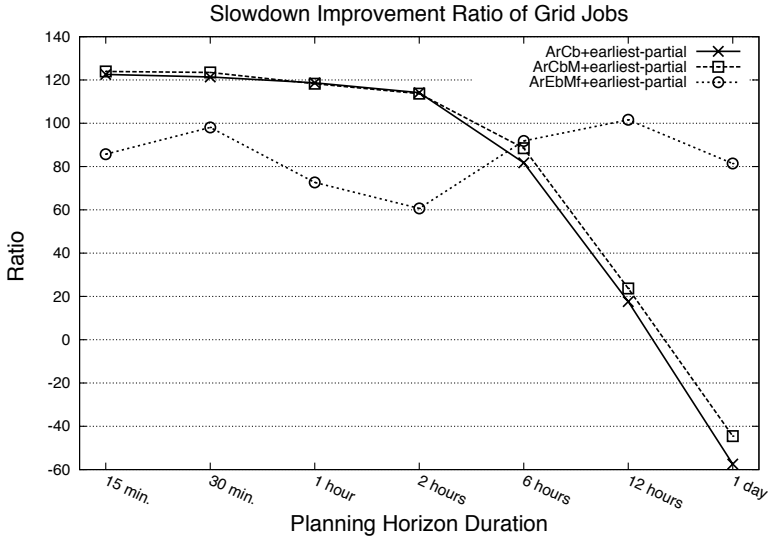


Figure 4.9: Slowdown improvement ratio of Grid jobs.

4.5 Conclusion

This chapter investigated resource provisioning in multiple-site environments, evaluating whether it is possible to provision resources for Grid applications based on availability information given by resource providers using existing resource management systems.

We presented simulation results that show that in an environment like DAS-2, a gateway can provision resources to Grid applications and achieve better job slowdown if resource providers inform the available time slots, at time intervals between 15 and 30 minutes. The experimental results showed that it is possible to avoid requesting availability information for every Grid job scheduled, thus reducing the communication overhead.

Multiple resource partition policies can improve the slowdown of both local and Grid jobs if conservative backfilling is used. In addition, the scheduling policy based on multiple resource partitions, aggressive backfilling, and load forecasts enabled the provision of larger free time slots, producing a balance of performance between the Grid jobs and the providers' local jobs.

Chapter 5

InterGrid Resource Provisioning

As the resource utilisation within a Grid has fixed and operational costs, a Grid can benefit by redirecting requests to another Grid, thus reducing the cost of over-provisioning. In this chapter, we enable load management across Grids through inter-Grid resource sharing and consider the cost of one Grid acquiring resources from another. However, enabling resource sharing among Grids is a challenging task: a Grid should not compromise the performance of its local user communities' applications, yet can benefit from providing spare resources to other Grids. The load management mechanism and related policies consider the economic compensation of providers for the resources allocated. Experimental results show that the mechanism achieves its goal in redirecting requests, increasing the number of user requests accepted and balancing the load among Grids.

5.1 Introduction

A Grid infrastructure is expensive to maintain as resource utilisation incurs fixed and operational costs, such as electricity, system administrators, or compensation to resource providers. Consequently, there can be financial benefits for a Grid to provide spare capacity to peering Grids, or acquire resources from peering Grids to service occasional internal peak demands. Such cross-Grid load management could reduce the costs incurred by over-provisioning, and provide the means for provisioning resources from multiple Grids to applications.

Enabling resource sharing between Grids, however, is complex due to the autonomy within each Grid for capacity planning and provisioning of resources to user communities. There is contention for resources and dynamicity of shares supplied by resource providers within each Grid. The main challenges when designing a load sharing mechanism are how a Grid can (i) meet the demand for resources by local user communities; (ii) coordinate with other Grids to acquire additional resources to satisfy excess demands; and (iii) provide spare resources to other Grids, preferably in return for payments.

This chapter investigates a resource sharing mechanism that takes into account the economic compensation of resource providers and considers the cost for one Grid to acquire computational resources from another. The proposed mechanism allows a Grid to redirect a request to a peering Grid when the cost of serving it locally is higher than the price the Grid would pay for the peering Grid to process the request. The redirection takes

place between Grids that have a pre-established peering arrangement. Experiments evaluate the proposed mechanism in terms of load balancing across the Grids, the increase in the overall request acceptance rate, and the response time of Grid applications.

5.2 Provisioning in InterGrid Environments

Chapter 3 has introduced the InterGrid architecture based on gateways that mediate resource exchange between Grids. It allows participants to seamlessly allocate resources from different Grids. Chapter 4 has explored a scenario in which one InterGrid Gateway (IGG) provisions resources from one Grid based on availability information given by resource providers. This chapter extends this scenario to consider multiple IGGs and consequently multiple Grids.

The environment considered in this chapter is depicted by Figure 5.1. A Resource Provider (RP) has local users whose resource demands need to be satisfied, yet it delegates provisioning rights over spare resources to an IGG by providing information about the resources available in the form of free time slots. A free time slot includes information about the number of resources available, their configuration and period over which they would be available.

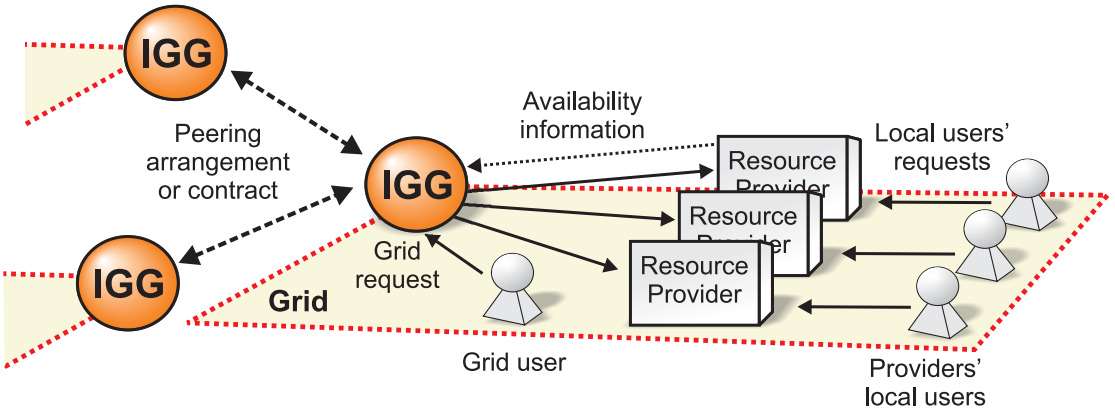


Figure 5.1: Provisioning scenario with multiple Grids considered in this chapter.

Moreover, we consider the case in which a Grid has pre-defined peering arrangements with other Grids, managed by IGGs and, through which they co-ordinate the use of resources of the InterGrid. An IGG is aware of the terms of the peering arrangements with other Grids, selects a suitable Grid capable of providing the required resources, and responds to requests from other IGGs. The peering arrangement between two Grids is represented as a contract that specifies a price range for the resources allocated from one another. Request redirection policies determine which peering Grid is selected to process a request and at what price the processing is performed.

When a Grid user needs to deploy or execute an application, she requests the IGG for a number of resources. When the individual Grid cannot provide the required resources, the IGG selects a peering Grid based on the agreements and the policies in place. The user is then given a resource ticket granting access to the resources, which is passed to the selected provider in return for the required resources.

5.2.1 Types of User Requests

A request corresponds to an individual job whereas an application can comprise several jobs. A request is contiguous and needs to be served with resources from a single resource provider. Allowing the request to be served with resources from multiple resource providers could require co-allocation, which is not addressed here. A request received by an IGG contains a description of the required resources and the usage time. The request can require a best-effort service, meaning that the resources can be provided at any time as long as they are made available for the requested usage time. Alternatively, the request can be deadline-constrained, so resources need to be allocated within a specified period.

5.2.2 Problem Description and Propositions

The peering agreements between Grids define which resources are exchanged between the Grids and the price of the resources exchanged. The policies specify when an IGG redirects requests to another, and when a request redirected by one IGG is accepted by another IGG. The goal of a participating IGG is to (i) serve its user communities by providing allocations that assign resources to satisfy their requirements; (ii) offer spare resources to peering Grids preferably under some compensation; and (iii) acquire resources from other Grids under peak load conditions to satisfy its users.

Specifically, we would like to verify the following propositions:

- Load balancing can be achieved by a mechanism that takes into account the marginal cost of accepting or redirecting requests across the interconnected Grids.
- A mechanism based on requests' marginal cost of allocation can enable the provisioning of resources from multiple Grids to applications and improve the jobs' response times.
- The provisioning schemes studied in Chapter 4 provide the resource availability information required to carry out resource sharing decisions, thus improving the response time of Grid jobs without sacrificing providers' local jobs.

5.3 Resource Provisioning and Load Sharing

The adoption of economic principles for load sharing amongst Grids comes from observing how economic institutions in the real world regulate the allocation of resources, goods and the use of services [24]. Economic approaches can cope with problems like providing Grid resources to different users with diverging quality-of-service requirements, and how to reward resource suppliers. As described beforehand, the interconnection of Grids involves problems that are similar to those faced by ISPs peering in the Internet. ISPs operate to make a profit – they see one another as competitors or sources of revenue – but they interconnect their networks for economic or technical reasons [12, 13, 39, 194]. ISPs have fixed and variable costs with network infrastructure, yet interconnect their network domains to benefit from having a larger network coverage or offloading expensive links. Similarly, the use of Grid resources has fixed and variable costs. This section describes a mechanism for request redirection amongst Grids that considers their cost.

For each IGG_i, the allocation of its resources by its user communities over a unit of time represents a cost. The real-valued total cost function of IGG_i is represented by $cost_i(L)$, where $0 \leqslant L \leqslant 1$ is the current load determined by the number of resource units available in its Grid. For simplicity, a resource unit corresponds to one resource per second (*i.e.* a second of a CPU). Therefore, the total cost given by $cost_i(L)$ depends on the number of resources allocated. Although each Grid could have its own cost function, in this thesis the participating Grids utilise a quadratic cost function. Such a function reflects a network infrastructure that does not offer any provision for handling peak demands, consequently it is very costly to keep the resources operating at full capacity. A function with a steep cost when the system approaches full utilisation therefore reflects current computing and network infrastructures. Moreover, a non-linear function is required to specify contracts with price ranges, as discussed later in this section.

The cost function $cost_i(L)$ is given by $[L_{units} * (p_{cost} + (p_{cost} * (\beta L)^2))]$, where L_{units} is the number of units in use at load L, β is a small constant value that determines how steep the cost curve is as the load approaches 1 and p_{cost} is the average price that IGG_i pays to resource providers for a resource unit. The price of a resource unit within IGG_i is given by the second part of the cost function (*i.e.* $p_{cost} + (p_{cost} * (\beta L)^2)$. We derive the average price p_{cost} paid by IGG_i to resource providers for a resource unit using Equation 5.1:

$$p_{cost} = \sum_{i=1}^{n} \left(cp_i \left(\frac{ru_i}{\sum_{j=1}^{n} ru_j} \right) \right) \tag{5.1}$$

where n is the number of resource providers in IGG_i's Grid; cp_i is the price of a resource unit at resource provider i; and ru_i is the number of resource units contributed by provider i until a given time horizon, or request deadline. When updating the prices for resource units specified in the contracts, the horizon is the time of the next contract update (*i.e.* the next time when the IGGs update the prices of units negotiated). In this way, L depends on how many resource units are available from the start time until the horizon and how many units are in use.

A request redirection is decided based on the per request cost $mc_i : (u, L) \rightarrow \Re$ which is the increment in total cost for IGG_i for agreeing to provide resource units required by request u given its current load or allocations. If request u requires resource units that place u_{load} load in IGG_i's Grid, then the cost of serving u is derived by Equation 5.2. If request u requires one resource unit, then the request cost is equal to a *unit cost*.

$$mc_i = cost_i(L + u_{load}) - cost_i(L) \tag{5.2}$$

IGG_i has a load threshold, if crossed, IGG_i considers itself overloaded. The redirection of requests is enabled between Grids that have negotiated contracts, at within the contracted price range. A contract $C_{i,j}$ between IGG_i and IGG_j has a price range $PR(C_{i,j}) : [price_{min}, price_{max}]$, where $price_{min}$ and $price_{max}$ are the minimum and maximum prices respectively paid by IGG_i for a resource unit allocated from IGG_j. IGG_i can have contracts with multiple Grids. During periods of peak load, IGG_i can redirect requests to IGG_j if and only if both have a contract. Based on the current load levels, they agree on a final price $price_{final}$ within $PR(C_{i,j})$. IGG_i pays the amount equivalent to ($price_{final} *$ ***number of units***). The redirection occurs when a Grid forwards requests to another Grid because the cost of fulfilling the requests is higher than the charge for

another Grid to service them.

5.3.1 Contract Types

The proposed mechanism supports two kinds of contracts: fixed price and price range contracts. A fixed price contract is given by $PR(C_{i,j}) : [price_{max}, price_{max}]$ where $price_{max}$ is the fixed price and a price range contract corresponds to $PR(C_{i,j}) : [price_{max} - \Delta, price_{max}]$, where Δ determines the price range. In the case of price range contracts, participating Grids have to negotiate the final price at runtime. As discussed by Balazinska *et al.* [14], a load management mechanism based on fixed price contracts may present disadvantages in some cases. For example, it reduces the flexibility in redirecting requests as a Grid can only offload requests if its cost of allocation is higher than the price it would pay to another Grid (*i.e.* the number of resource units required by the request multiplied by the unit cost specified in the contract).

The price range for a resource unit is determined by the decrease of load *k* from the load *L*. Let *u* be a request that requires u_{units} resource units and causes an increase in load u_{load}. The decrease in the per-unit cost due to removing *k* from the Grid's *L* is represented by δ_k, which is defined by Equation 5.3.

$$\delta_k(L) = \frac{mc(u, L - u_{load}) - mc(u, L - k - u_{load})}{u_{units}} \tag{5.3}$$

δ_k is the approximate difference in the cost function evaluated at the load level including and excluding load *k*. Given a contract with fixed price $price_{max}$, L is the maximum load that an IGG can approach before its per resource unit cost exceeds $price_{max}$. In order to estimate the price range for a resource unit in the contracts, the experiments presented in this chapter take L as the load threshold; $u_{units} = 1$ and $\Delta = \delta_k$. The experiments consider different values for L and k.

5.3.2 Provisioning Policies

The policies described in this section define how an IGG redirects requests to peering Grids considering a contract network and how it accepts requests from other Grids.

During a time interval, IGG_i stores the requests in the waiting queue. After the interval, IGG_i orders the contracts in ascending order of price and for each contract IGG_i evaluates whether there are requests that can be redirected to the peer IGG. Figure 5.2 illustrates the negotiation between IGG_i and IGG_j under a price range contract. The scenario is as follows:

1) IGG_i sends an *offer* to IGG_j to use its resources under the terms of their contract when IGG_i's unit cost to service the request is higher than the minimum price of the contract with IGG_j. The price in the offer p_{offer} is the minimum price specified in the contract between IGG_i and IGG_j.

2) IGG_j, in turn, replies with one of the following messages:

   2.1) IGG_j sends an *accept* message whose price is the price in the initial offer if the request's cost is lower than or equal to the amount that IGG_i is willing to pay

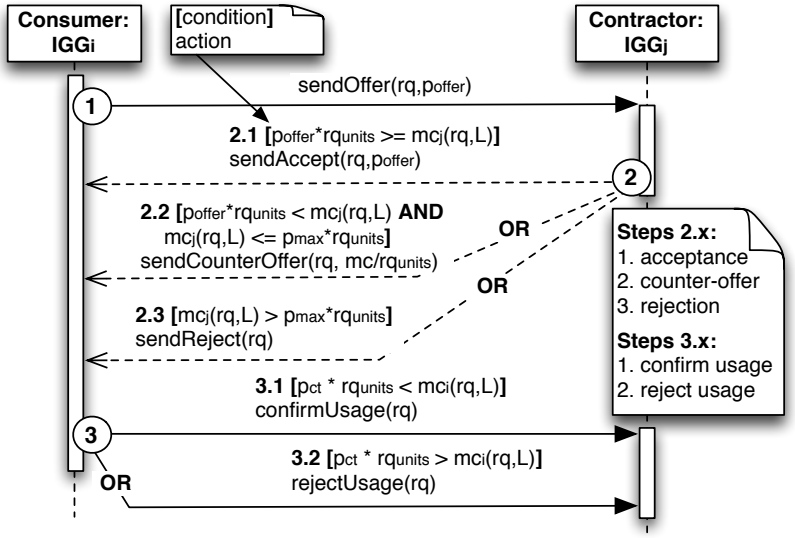


Figure 5.2: Redirection negotiation.

(*i.e.* p_{offer} multiplied by the number of resource units required by the request rq_{units}).

2.2) If IGG_j's request cost is greater than the amount offered by IGG_i, but less than the maximum amount that IGG_i would possibly pay (*i.e.* the contract's maximum price p_{max} multiplied by rq_{units}), then IGG_j sends a *counter-offer* whose price is mc_j/rq_{units}. For simplicity the counter-offer contains the peering IGG_j's unit cost for the request, but the mechanism can easily be extended to incorporate a profit margin or use profit maximisation techniques.

2.3) If IGG_j's request cost is higher than the maximum amount IGG_i is willing to pay, the offer is rejected.

3) After receiving IGG_j's message, IGG_i replies as follows:

3.1) IGG_i accepts the counter-offer if its request cost is still higher than the amount asked by IGG_j (*i.e.* number of resource units required rq_{units} multiplied by the counter-offer's price p_{ct}.

3.2) Otherwise, the counter-offer is rejected. IGG_i keeps the request in the queue and repeats the whole process for the next contract.

IGG_j stores the offers and evaluates them at time intervals. The evaluation algorithm sorts the offers by decreasing order of price. In addition, IGG_j maintains a list of tickets created to serve the requests whose negotiations are in progress. In this way, the evaluation of the request cost considers the requests being served as well as those whose negotiations are in progress. Creating a ticket corresponds to finding a time slot for the job. Moreover, in order to reduce the number of messages exchanged by IGGs, when IGG_i sends an offer to IGG_j, the offer contains a list of requests that IGG_i is willing to redirect to IGG_j. That is, a negotiation is performed for a group of requests and not on a per-request

basis. IGG_j can accept all or part of the requests whose price is within the accepted price range up to its capacity.

As described beforehand, there are two types of requests, namely best-effort and deadline constrained. The IGG uses an earliest start time policy to select the resources to serve a request. The request's deadline is the time horizon used to calculate the load in the Grid, the load imposed by the request and consequently the request cost. In this way, the Grid load for example, is determined by the resource shares provided by RPs and the allocations until the horizon. For best-effort requests, the IGG creates a virtual deadline given by the latest start time based on the time slots held by the IGG plus the runtime estimate; the virtual deadline is used as the horizon.

5.3.3 Storing Free Time Slots at the Gateway

The resource providers issue free time slots and send them to the IGG periodically. The IGG maintains a provider's availability information on availability profiles, which are modified red-black trees [40]. Each node of the tree has two references, namely to its predecessor and successor nodes, thus forming an additional linked list. The IGG has a table of availability profiles wherein one individual profile stores the availability information from one resource provider. Appendix A provides details about the design and implementation of the availability profile.

5.4 Performance Evaluation

The simulated environment is composed of three Grids, namely DAS-2 in the Netherlands, and Grid'5000 and AuverGrid in France. The Grids DAS-2 [43], Grid'5000 [21] and AuverGrid [106] comprise 5, 15 and 5 clusters respectively.[1] Figure 5.3 depicts the environment simulated: layers 1, 2 and 3 represent a pictorial view of the physical location of provider sites, their Grid organisation and the InterGrid configuration respectively.

The evaluation is performed using a discrete-event simulator (*i.e.* GridSim). We extended GridSim to support the scheduling of parallel jobs, gateways and the resource allocation schemes described in this chapter.[2]

This section presents two sets of experiments. The first set of experiments investigates the load management mechanism under a scenario with no resource contention at the resource providers. That is, the availability information given by providers is precise, which means that it does not change due to the arrival of jobs dispatched by the providers' local users. In this sense, this experiment models the availability information obtained from resource providers using an on-off scheme, wherein on and off intervals correspond to off-peak and peak periods respectively [11]. This experiment aims to investigate the efficiency of the proposed load management mechanism. In the second set of experiments (Section 5.4.2), the availability information is obtained from providers by using a sub-set of the provisioning strategies discussed in Chapter 4.

[1] For detailed information on the characteristics of the clusters we refer to Iosup *et al.* [101] and the Grid Workloads Archive website at http://gwa.ewi.tudelft.nl/pmwiki/

[2] More information about the changes in the simulator is available at http://www.gridbus.org/intergrid/gridsim.html

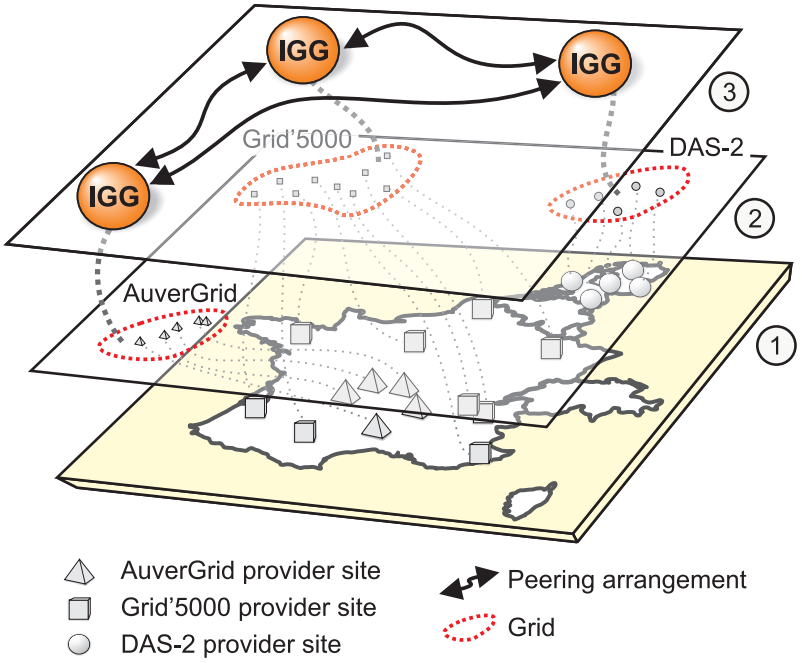


Figure 5.3: Grid environment simulated.

5.4.1 Providers Using an On-Off Scheme

The parameters used by the experiments are summarised in Table 5.1. The resource providers use a pricing function for a resource unit given by Equation 5.4:

$$price = cost + (cost * load) \tag{5.4}$$

where $cost$ is drawn from a uniform distribution; $load$ is generated using an on-off model as described by AuYoung *et al.* [11]. The duration of the on and off intervals and the load in each interval are also modelled using uniform distributions as described in Table 5.1. The on-off model is used to model the availability of part of the providers, around 50 per cent. This means that all the remaining resource providers have their resources dedicated to the Grids. Table 5.1 also shows the number of CPUs dedicated in each Grid.

To model the workloads of the Grids we use job traces of real Grids obtained from the Grid Workloads Archive.[3] We divided the traces into four-month intervals. The experiments use the interval between the 9^{th}-12^{th} months of DAS-2' trace, the 5^{th}-8^{th} months of AuverGrid's and the 17^{th}-20^{th} months of Grid'5000's. We make an effort to eliminate the systems' warm-up and cool-down phases by disregarding the first week of the experiment results and deeming the last simulated event as the arrival of the last job submitted in any of the Grid workloads, respectively. As described beforehand, there are two types of requests, namely deadline constrained and best-effort. We randomly selected the requests that are deadline constrained. To generate the request deadlines we use a technique described by Islam *et al.* [103], which provides a feasible schedule for the jobs. We perform the experiments using the same Grid environment with no contracts among the gateways, using an aggressive backfilling policy at the resource providers and a 'submit to the least

[3]Grid Workloads Archive website: http://gwa.ewi.tudelft.nl/pmwiki/

Table 5.1: Parameters used by the first set of experiments.

Parameter	Value
Number of Grids	3
Contract topology	all-to-all (Figure 5.3)
Load at off-peak (ON) intervals (%)	50 – 100
Load at peak (OFF) intervals (%)	20 – 50
ON interval duration (hours)	24 – 72
OFF interval duration (hours)	12 – 48
Cost of a resource unit	0.90 – 1.00
Number of dedicated CPUs (DAS-2)	192
Number of dedicated CPUs (AuverGrid)	234
Number of dedicated CPUs (Grid'5000)	648
Deadline constrained requests (%)	30
Stringency factor	5.00

loaded resource' policy at the gateway. A request's deadline is the completion of the corresponding job under this scenario multiplied by a stringency factor (Table 5.1).

We calculate the price of a resource unit in the contracts between the Grids by assigning different values to *L* in Equation 5.3. The experiments evaluate cases with *L* equal to 0.99 and 0.95 and different values for *k*: 0.01, 0.05, 0.1 and 0.2. For example, when *L*=0.99 and *k*=0.01, the fixed price ($price_{max}$) of a contract is the cost of a request requiring one resource unit of the Grid's capacity when the Grid is 99% utilised. The price range contract has a maximum price of $price_{max}$ and a minimum price given by $price_{max}$ minus the difference between the request cost at 99% and at 98% of utilisation.

Performance Metrics

We select two metrics, namely increase in requests accepted and the percentage of the generated load redirected by the IGGs. The redirected load shows the performance of the mechanism in terms of managing peak loads The increase in requests accepted, on the other hand, demonstrates whether the IGG compromises local users by peering with other IGGs. The experiments also compute the increase in utilisation for comparison against the migration of load.

Experimental Results

All the results presented in this section are averages of 10 simulation rounds using different simulation seeds and excluding the best and worst results. The global increase in the number of requests served under different types of contracts is shown in Table 5.2. Overall, there is an increase in the number of requests accepted, except for DAS-2, whose acceptance rate is decreased. Further investigation revealed that DAS-2 has a lower utilisation than the other two Grids. When we calculate the deadlines using the 'submit to the least loaded resource' and aggressive backfilling policies, the deadlines become very tight as many jobs are started immediately while others backfill easily, thus generating very tight deadlines. As IGGs run the provisioning algorithm at time intervals, some requests are rejected. When the deadlines are not used, all the requests are processed. As the price

range increases, more load can be migrated. However, in the experiments, as k becomes large (*i.e.* $k > 0.2$), the mechanism becomes unstable as Grids tend to both redirect and accept too many requests.

Table 5.2: Increase in both requests accepted and resource utilisation.

		$L = 0.99$				
		Fixed Price	k			
Metric	**Grid**		0.01	0.05	0.1	0.2
Increase in number of requests served	DAS-2	-6.50	-4.50	-3.38	-5.25	-10.00
	AuverGrid	658.00	661.12	661.12	657.00	663.00
	Grid´5000	19.12	8.88	10.62	4.88	6.38
Increase in resource utilisation (%)	DAS-2	7.62	7.99	9.42	12.56	8.39
	AuverGrid	-23.30	-24.15	-26.45	-27.56	-29.15
	Grid´5000	6.12	6.38	6.92	6.35	7.98
		$L = 0.95$				
		Fixed Price	k			
Metric	**Grid**		0.01	0.05	0.1	0.2
Increase in number of requests served	DAS-2	-5.38	-4.25	-8.00	-4.25	-4.88
	AuverGrid	650.38	659.50	658.12	661.00	662.00
	Grid´5000	18.25	7.00	13.12	10.50	14.88
Increase in resource utilisation (%)	DAS-2	7.72	8.93	8.18	8.69	9.24
	AuverGrid	-22.20	-23.45	-25.42	-27.52	-30.96
	Grid´5000	5.71	5.51	6.78	7.25	8.13

The table also shows the increase in resource utilisation at each Grid. Also, Grid'5000 redirects smaller amounts of load (Figure 5.4). Even though Grid'5000 does not have a substantial increase in the acceptance rate of the requests originated by its users, it increases its resource utilisation without compromising the acceptance rate. This could lead to an increase in profit as they can receive a payment from the other Grids for the resources provided. However, the experiments do not measure the profits of each Grid. Overall, the algorithms achieve their goal, which is to redirect requests driven by the requests' marginal cost of allocation. AuverGrid has a decrease in utilisation in contrast to DAS-2 and Grid'5000. As AuverGrid has a higher utilisation than DAS-2 and Grid'5000 when they are not redirecting requests, this decrease in utilisation shows that the mechanism is effective in redirecting AuverGrid's requests to other Grids. Figure 5.5 shows that there is an initial load imbalance between the Grids, as the utilisation of AuverGrid without contracts is close to 70%, while DAS-2 and Grid'5000 are both close to 10%. The table shows that the mechanism helps to balance the load across the Grids.

Figure 5.4 presents the percentage of the load generated by each Grid redirected to other Grids. When Grids define the maximum price for a resource unit as the unit cost at 99% of utilisation (*i.e.* $L = 0.99$), they exchange load and the overall number of requests accepted is improved in almost all the cases (Table 5.2). The acceptance is better when contracts define a price range, which allows Grids to redirect more load.

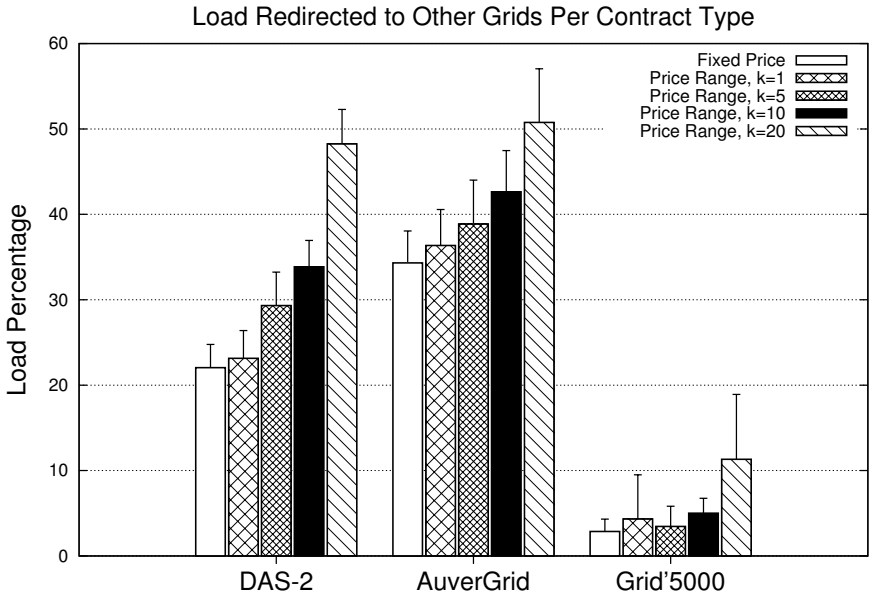


Figure 5.4: Load redirected (L=0.99).

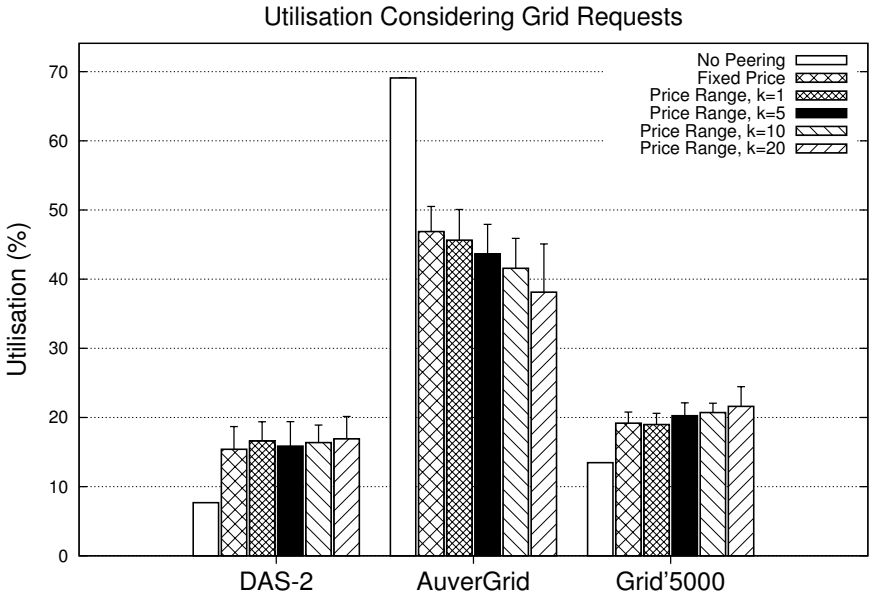


Figure 5.5: Resource utilisation at Grids (L=0.99).

When IGGs set the maximum price as the unit cost at a low value for L and the price range is large, the number of requests accepted increases whereas the amount of load exchanged decreases. That occurs because overloaded IGGs tend to be more conservative in accepting load from other IGGs, even though they try to migrate load more easily. An IGG_i that needs to offload will redirect requests willing to pay a price lower or equal to the maximum price of a contract. If the unit cost of the IGG considering accepting the load is slightly above the maximum price, it will not accept the load. In our experiments, the increase of accepted requests reflects this behaviour.

The experiments show that load management across Grids through resource exchange incorporating economic compensation for resource providers is possible. The resource utilisation and increase in accepted requests show that Grids balance their load and redirect requests, which could help minimise the costs with resource usage.

5.4.2 Providers Using Provisioning Policies

These experiments also model the workloads of the Grids using traces obtained from the Grid Workloads Archive. However, here we divide the traces into 2-month intervals. Each simulation randomly selects one interval from the trace of each Grid to model the load of that Grid. Attempting to eliminate the system warm-up, we disregard the first two weeks of results; for the load forecast policy, the second week is used for training.

The resource providers' local jobs are generated according to the Lublin99 model [118]; similarly to Section 4.4. In these experiments, however, we generate 2-month long workloads. We also change two parameters of the Lublin99 model when generating the workload for each cluster. The medium size of a parallel job (specified in log_2) is set to $log_2 m - \theta$ where m is the number of CPUs in the system and θ is drawn uniformly from 1.5 to 3.5. In addition, the inter-arrival rate of jobs is modified by setting the β of the used gamma distribution to a value uniformly distributed between 0.4871 and 0.55. These changes lead to workloads with different loads and different job arrival rates, which is representative of Grid resources.

Performance Metrics

The performance evaluation considers two metrics: the Average Weighted Response Time (AWRT) [90] of jobs and the percentage of the generated load redirected by the IGGs. The AWRT measures how long on average users wait to have their jobs executed; a short AWRT indicates that users do not wait long for their jobs to complete on average. The redirected load demonstrates the performance of the mechanism in terms of managing peak loads; the AWRT, on the other hand, demonstrates whether the response time of user requests is improved through peering of IGGs.

$$AWRT_k = \frac{\sum_{j \in \tau_k} p_j \cdot m_j \cdot (ct_j - st_j)}{\sum_{j \in \tau_k} p_j \cdot m_j} \tag{5.5}$$

The AWRT is given by Equation 5.5, where m_j is the number of processors required by job j, p_j is the execution time of the job, ct_j is the completion time of the job and st_j

is the job's submission time. The resource consumption ($p_j \cdot m_j$) of each job j is used as the weight.

Policy Acronyms

Similar to Chapter 4, the name of the policies are abbreviated. A policy name comprises two parts separated by +. The first part represents the policy employed by the provider whereas the second represents the IGG policy. In the resource provider's side, **Eb** stands for aggressive backfilling, **Cb** for Conservative backfilling, **M** for Multiple partitions and **Mf** for Multiple partitions with load forecast. On the other side, for the IGG's policy, **least-load** means 'submit to the least loaded resource', **earliest** represents 'select the earliest start time' based on the free time slots given by providers on a periodical basis. In this way, **EbMf+earliest-partial** for example, indicates that providers use aggressive backfilling, multiple partitions and load forecasts, whereas the IGG submits jobs selecting the earliest start time based on the availability information sent by providers at regular intervals.

Experimental Results

The parameters used for the experiments are summarised in Table 5.3. The fixed cost of a resource in Equation 5.4 is drawn uniformly from 0.9 to 1. The load threshold (L) and k are set to 95% and 5% respectively. The IGGs inform one another about their fixed prices or price ranges in their contracts based on the current resource demand at intervals between 1 and 6 hours. The results are averages of 10 simulation rounds excluding the best and worst results. The simulation seed to generate the providers' local workloads, the prices and the contract update intervals is changed at each round.

Table 5.3: Parameters used in the second set of experiments.

Parameter	Description
Number of Grids	3
Contract topology	all-to-all (see Figure 5.3)
Number of simulation rounds	10
Cost of a resource unit	0.90-1.00
Load threshold (%)	95
Value of k (%)	5
Time between contract updates (hours)	1-6
Number of clusters at DAS-2	5
Number of CPUs at DAS-2	400
Number of clusters at AuverGrid	5
Number of CPUs at AuverGrid	475
Number of clusters at Grid'5000	15
Number of CPUs at Grid'5000	1368

First Experiment: The first experiment evaluates the AWRT of both Grid and local jobs in a scenario where the providers send the availability information to the IGG every 12 hours. Figure 5.6 shows the AWRT of Grid applications for four sets of allocation policies (*i.e.* Eb+least-load and EbMf+, Cb+ and CbM+earliest-start). The initial four bars

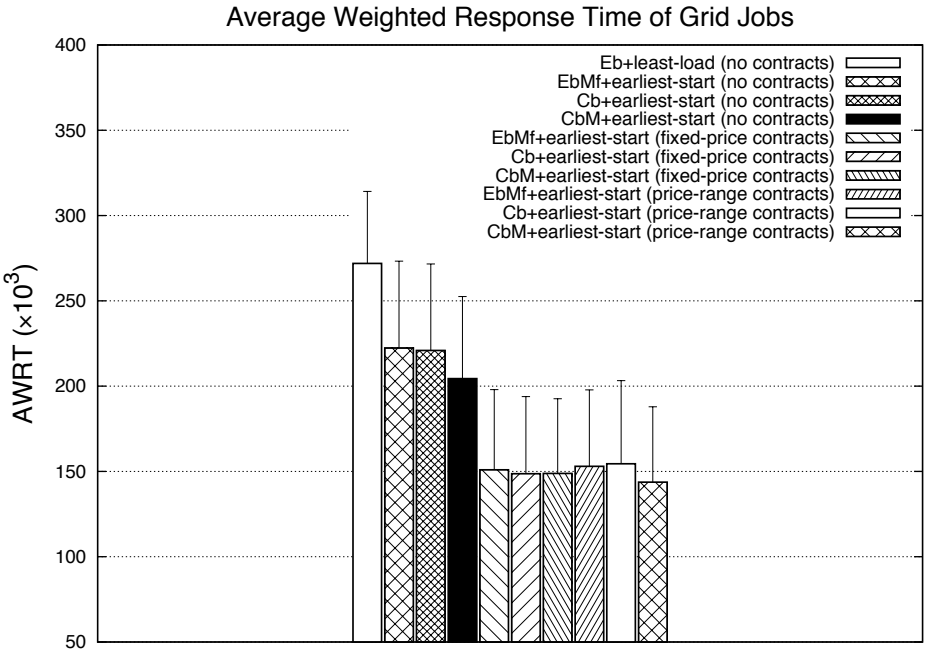


Figure 5.6: Average Weighted Response Time (AWRT) of Grid jobs.

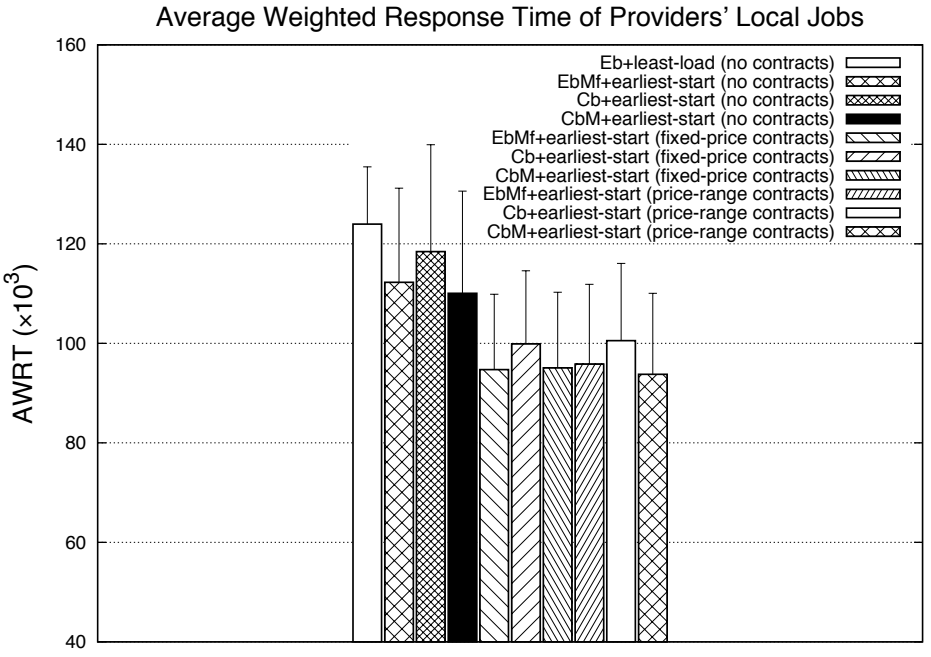


Figure 5.7: Average Weighted Response Time (AWRT) of providers' jobs.

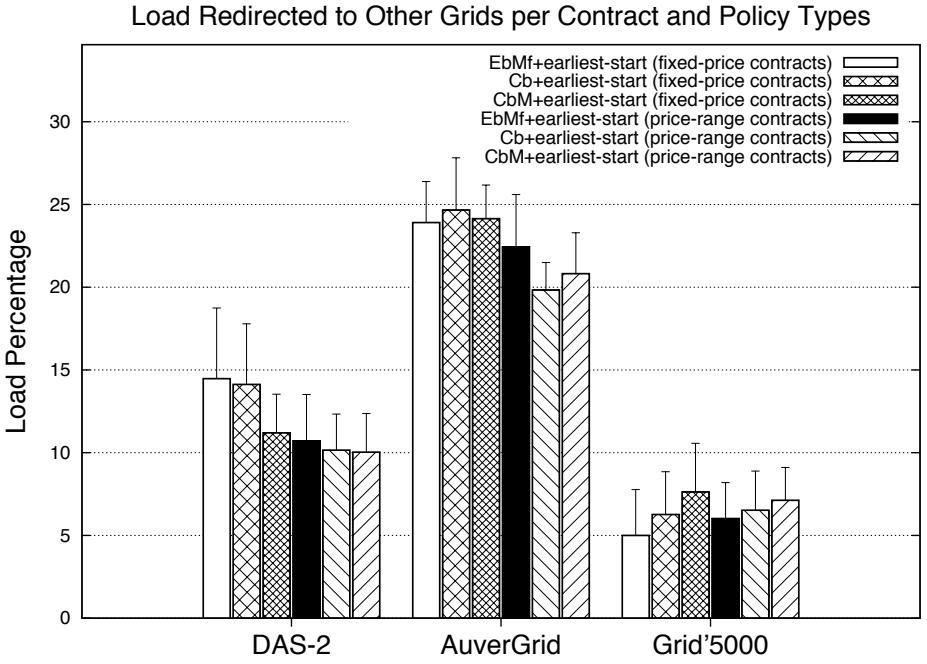


Figure 5.8: Percentage of load generated by each Grid that was redirected to other Grids.

represent the AWRT under no peering between IGGs, that is, the IGGs have no contracts with one another and therefore do not redirect requests. Bars 5 to 7 represent the AWRT of Grid jobs when fixed-price contracts are established amongst IGGs, whereas bars 8 to 10 show the AWRT under price range contracts. The aggressive backfilling with 'submit to the least loaded resource' (*i.e.* bar 1) is shown for the sake of comparison. We observe that overall, the AWRTs of local and Grid jobs are reduced by the peering of Grids under both fixed-price and price-range contracts. That occurs despite IGGs accumulating a number of requests to be handled at random intervals between 1 and 5 minutes when contracts exist, in contrast to Eb+least-load in which requests are handled upon their arrival at the IGG. The load forecast based policy (EbMf+earliest-start) leads to a decrease in the AWRT of Grid jobs in both fixed-price and price-range contracts, but it does not perform as good as the conservative backfilling based policies. However, initially we expected that this policy would have less impact on the providers' local jobs because they resize the free time slots given to the IGG based on load forecasts. In addition, previous results showed that the load forecast policy is influenced by the length of the horizon (Chapter 4).

The AWRT of local jobs show the impact of peering of Grids in the providers' user applications (Figure 5.7). Similarly to the Grid applications, the AWRT of local jobs is reduced with the peering of IGGs. The reduction is more accentuated for the load forecast based policy, confirming our expectations that by providing load forecasts, even if not very precise, the gateway can schedule jobs accounting for the providers' local load. Intriguingly, the AWRT of both Grid and local jobs under price range contracts is similar to, and in same cases worse than, that of fixed-price contracts. We initially expected that; although Grids can redirect more requests under price-range contracts, IGGs handling the requests and offers at random intervals between one and five minutes might account for the increase in AWRT. However, as described later, with the chosen price range contract, some

IGGs in fact redirect fewer requests. Therefore, we conclude that the increase in AWRT is caused by a Grid handling requests locally after a period of unsuccessful negotiations. This scenario may be improved by introducing a buy-it-now mechanism where a Grid could make an offer for immediate access to resources [10], however, investigation of such a mechanism is outside the scope of this thesis.

Figure 5.8 presents the percentage of the load from each Grid migrated to other Grids when providers send availability information every 12 hours. The first set of experiments revealed that the job acceptance is higher when the contracts define a price range, which allows Grids to redirect more load. However, with a price range defined by $k = 5\%$, Grids do not redirect more load in all the cases. For example, Figure 5.8 shows that when providers use conservative backfilling without multiple partitions, DAS-2 and AuverGrid in fact redirect less load.

Second Experiment: the second experiment evaluates the AWRT of Grid jobs in three situations where the providers send the availability information to the gateway every 24 hours, 12 hours and 6 hours respectively. Table 5.4 shows the AWRT of Grid jobs per Grid under each scenario. As noted earlier, AuverGrid has a higher load than DAS-2 and Grid'5000. Table 5.4 shows that Grids with a low utilisation (*i.e.* DAS-2 and Grid'5000) do not have a decrease in the AWRT of their Grid users' applications. In fact, the AWRT is worsened. In contrast, AuverGrid has a substantial reduction in the AWRT of its Grid jobs. The conclusion, therefore, is that for improving the AWRT at least, the peering of Grids with very different utilisation levels benefits highly-utilised Grids and the mechanism achieves its goal of redirecting requests to those with lower utilisation levels, as shown in Figure 5.8.

The second experiment also evaluates the AWRT of providers' local jobs for different Grids under different horizons. The results are presented in Table 5.5 and follow those of the AWRT of Grid jobs: AuverGrid benefits from the peering, with the AWRT of its providers' local jobs decreasing, Grid'5000 has small AWRT benefits for fixed-price contracts when providers utilise a conservative backfilling policy with multiple partitions, however DAS-2 has the AWRT of its providers' job worsened by the peering.

Third Experiment: the third experiment investigates the peering between DAS-2 and Grid'5000. Having the same characteristics as the second, this experiment investigates the AWRT of Grid jobs when the peering Grids are both less utilised. Table 5.6 shows the results: the AWRT of Grid'5000's jobs improves when the horizon is of 6 hours, and DAS-2 experiences small AWRT increases under the same horizon and policies. For the other horizons (*i.e.* 12 hours and 24 hours), however the results are mixed. Some small improvements are offset by some increases when IGGs store messages to be handled at time intervals when they have contracts with other IGGs. Hence, some requests have an additional delay incurred by the negotiation.

Overall, however, the experiments show that load management across Grids through resource exchange that considers the compensation of resource providers is possible with selected provisioning schemes. The load migrated shows that Grids balance their load and redirect requests. The allocation policies allow gateways to make decisions on resources provided to peering Grids. In addition, the overall AWRT of both Grid jobs and providers' local jobs is improved, however, some Grids have increases in the AWRT incurred by the negotiation time, which could be minimised through future research.

Table 5.4: AWRT of Grid jobs under different policies and intervals.

	Providers sending availability information every 24 hours								
	No contracts			Fixed-price contracts			Price-range contracts		
Grid	EbMf	Cb	CbM	EbMf	Cb	CbM	EbMf	Cb	CbM
DAS-2	39170	45644	43972	46025	50263	51258	47082	55502	47715
AuverGrid	419362	436861	394436	210710	200719	195419	193491	197075	207255
Grid´5000	191685	181689	176156	194614	182955	177300	189055	181915	175958
	Providers sending availability information every 12 hours								
DAS-2	37902	41295	40611	49190	44040	45101	50509	44800	43770
AuverGrid	433547	432191	391018	208548	218451	219517	214272	226716	220365
Grid´5000	190568	179463	174029	188503	178315	174342	188572	178078	173047
	Providers sending availability information every 6 hours								
DAS-2	39344	39063	38788	53297	42468	41154	51907	42818	40771
AuverGrid	439807	427912	389705	233263	240290	216316	227300	225625	215831
Grid´5000	189918	174233	170628	189295	175202	168825	191194	173863	171537

Table 5.5: AWRT of providers' local jobs under different policies and intervals.

	Providers sending availability information every 24 hours								
	No contracts			Fixed-price contracts			Price-range contracts		
Grid	EbMf	Cb	CbM	EbMf	Cb	CbM	EbMf	Cb	CbM
DAS-2	62030	65112	61095	83605	87580	87795	80370	90657	78051
AuverGrid	287352	289922	262810	168008	163892	155444	152469	158253	160937
Grid´5000	77573	76926	73146	76620	78199	72868	74580	78154	73939
	Providers sending availability information every 12 hours								
DAS-2	60814	63256	60500	76554	82402	75124	81889	80374	73920
AuverGrid	269261	287900	261086	156361	172751	169217	166731	179381	169700
Grid´5000	74285	75484	71831	73788	76657	71632	73263	75857	71478
	Providers sending availability information every 6 hours								
DAS-2	61371	62180	59326	78647	77866	74680	79951	78018	72136
AuverGrid	284865	285926	260735	182447	190212	166531	170587	180327	169151
Grid´5000	71993	74611	71098	72005	74084	70519	71216	74336	70774

Table 5.6: AWRT of Grid jobs for the interconnection of DAS-2 and Grid'5000.

	Providers sending availability information every 24 hours								
	No contracts			Fixed-price contracts			Price-range contracts		
Grid	EbMf	Cb	CbM	EbMf	Cb	CbM	EbMf	Cb	CbM
DAS-2	39139	45694	44001	40799	46058	45303	40623	46455	44495
Grid´5000	140843	145003	142751	139269	146913	142429	139719	144835	142768
	Providers sending availability information every 12 hours								
DAS-2	37881	41315	40560	39230	41330	41424	38984	41680	41806
Grid´5000	138999	145607	140184	143663	143766	140379	141989	142437	140435
	Providers sending availability information every 6 hours								
DAS-2	39325	39054	38741	39103	39427	38989	39410	39413	39043
Grid´5000	152476	141030	137761	139586	140367	137390	144104	140241	137596

5.5 Conclusion

This chapter presented the performance evaluation of schemes for resource provisioning across Grids. It demonstrated how a Grid can redirect requests to other Grids during periods of peak demand using a cost-aware load sharing mechanism. The mechanism relies on availability information obtained via different scheduling policies at provider sites, which were studied in Chapter 4. The provider policies enable information about fragments in the scheduling queue of clusters (*i.e.* the free time slots) to be obtained using ordinary resource management systems. The load sharing mechanism utilised these free time slots as the basis for load sharing among Grids.

We presented simulation results that demonstrate that the mechanism and policies are effective for redirecting requests across Grids leading to a reduction in the overall Average Weighted Response Time (AWRT) of Grid applications. Moreover, we showed that, overall, the proposed policy of a network of contracts amongst interconnected Grids improves the AWRT of providers' local jobs in comparison to traditional policies when the Grids are interconnected. However, some Grids have increases in the AWRT incurred by the negotiation time. The experiments demonstrated that, despite the imprecise resource availability information supplied by providers, the load management across Grids through resource sharing is possible while accounting for the compensation of resource providers.

Chapter 6

Mixing Commercial and Non-Commercial Providers

The maturity of virtual machine and network technologies has led to the emergence of commercial providers, who offer virtually unlimited numbers of resources to users, charging for the usage. This model is generally termed as "Cloud Computing" as the resources are on a "Cloud" whose physical infrastructure is unknown to the users. The emergence of these commercial infrastructure providers, their economies of scale, and the increasing costs of operating Grids may lead to future Grids comprising both commercial and non-commercial infrastructures. In this scenario, resource management systems should make provisioning and scheduling decisions taking into account the cost of using commercial infrastructure. This chapter investigates the benefits that organisations can reap by using commercial providers to augment the computing capacity of their local infrastructure. We evaluate the cost of six scheduling strategies used by an organisation that operates a cluster managed by virtual machine technology and seeks to utilise resources from a remote commercial provider to reduce the response time of its user requests. Requests for virtual machines are submitted to the organisation's cluster, but additional virtual machines are instantiated in the remote provider and added to the local cluster when there are insufficient resources to serve the users' requests. Naïve scheduling strategies can have a great impact on the amount paid by the organisation for using remote resources, potentially increasing the overall cost with the use of commercial infrastructure. Therefore, this chapter investigates scheduling strategies that consider the use of resources from commercial providers, to understand how these strategies achieve a balance between performance and usage cost, and how much they improve the requests' response times.

6.1 Introduction

Managing and providing computational resources to user applications is one of the main challenges for the high performance computing community. As discussed in previous chapters, to manage resources existing Grid solutions rely on a job abstraction for resource control, where users submit their applications as batch jobs to a resource management system responsible for job scheduling and resource allocation. This usage model has served the requirements of a large number of users and the execution of numerous

scientific applications. However, this usage model demands the user to know very well the environment on which the application will execute. In addition, users can sometimes require administrative privileges over the resources to customise the execution environment by updating libraries and software required, which is not always possible using the job model.

As discussed in Chapter 2, the maturity and increasing availability of virtual machine technologies has enabled another form of resource control based on the abstraction of containers. A virtual machine can be leased and used as a container for application deployment [146]. Under this scenario, a user can lease a number of virtual machines with the operating system of choice. These virtual machines can be further customised to provide the software stack required to execute the user applications. This form of resource control has enabled a number of usage models, including that of batch job scheduling [171].

The creation of customised virtual machine environments atop a physical infrastructure has also enabled another model recently known as "Cloud Computing" [9, 193]. Based on the economies of scale and recent Web and network technologies, commercial resource providers, such as Amazon Inc., aim to offer resources to users in a pay-as-you-go manner. These commercial providers, also known as Cloud providers and Infrastructure as a Service (IaaS) providers, allow users to set up and customise execution environments according to their application needs.

This chapter investigates whether an organisation operating its local cluster can benefit from using Cloud providers to improve the performance of its users' requests. We propose and evaluate six scheduling strategies suitable for a local cluster that is managed by virtual machine technology to improve its Service Level Agreements (SLAs) with users. These strategies aim to utilise remote resources from the Cloud to augment the capacity of the local cluster. However, as the use of Cloud resources incurs a cost, the problem is to find the price at which this performance improvement is achieved. This chapter aims to explore the trade-off between performance improvement and cost.

Specifically, the research described in this chapter attempts to:

- Describe how the proposed system can enable an organisation to extend its computing infrastructure by allocating resources from a Cloud provider.
- Provide various scheduling strategies that aim to improve the performance of applications using resources from Cloud providers.
- Evaluate the proposed strategies, considering different performance metrics; namely average weighted response time, job slowdown, number of deadline violations, number of jobs rejected, and the money spent for using Cloud resources.

6.2 Infrastructure as a Service and Lease Abstractions

Virtualisation technologies have enabled the realisation of new models such as Cloud Computing [9, 193] and IaaS. The main idea is to supply users with on-demand access to computing or storage resources and charge fees for their usage. In this model, users pay only for the resources they utilise. A key provider of this type of on-demand infrastructure is Amazon with its Elastic Compute Cloud (EC2) [4].

To use Amazon's infrastructure, users deploy instances of pre-submitted virtual machine images. They can also upload their own virtual machine images to EC2. The EC2 service utilises the Amazon Simple Storage Service (S3), which aims to provide users with a globally accessible storage system. Another example of commercial Cloud computing solution is provided by 3Tera[1].

Lease abstractions relying on virtual machine technology have been proposed [102, 108, 171]. Sotomayor *et al.* [171] explored a lease abstraction to handle the scheduling of a combination of best-effort jobs and advance reservations. Keahey *et al.* [108] demonstrated how to create customised execution environments for a Grid community via Globus Virtual Workspaces. As noted in Chapter 2, Shirako's brokers enable the leasing of various types of resources including virtual machines [102]. Previous work has also shown how to enable virtual clusters that span multiple physical clusters [63, 157, 164]. Emeneker *et al.* [63] evaluated the overhead of creating virtual clusters using Xen [15] and the Moab scheduler.

The applicability of Amazon services for Grid computing has been demonstrated in existing work. Palankar *et al.* [138] evaluated the use of Amazon S3 for Science Grids with data-intensive applications and concluded that Amazon S3 can be used for some of the operations required by data-intensive Grid applications. Although Grid applications can benefit from using Amazon services, such as improved data availability, Palankar *et al.* highlighted that a balance between the benefits of Amazon services and the cost of using Amazon's infrastructure should be taken into account. This balance involves performing expensive operations that generate large amounts of temporary data locally at the Grid infrastructure. Deelman *et al.* [48] evaluated the cost of using Amazon EC2 and S3 services to serve the resource requirements of a scientific application.

This thesis research improves previous work by proposing scheduling strategies that aim at improving the response time of applications running on a cluster by extending this cluster's capacity with resources acquired from Amazon EC2. This thesis evaluates the ratios of performance improvements to the money spent for using resources from Amazon EC2. This model can further extend the provisioning techniques proposed in Chapter 4.

6.3 Resource Provisioning Scenario

This thesis considers the case where an organisation manages a cluster of computers through virtual machine technology to supply its users with the resources required by their applications. The scenario is depicted in Figure 6.1, which can also represent a centre that provides computing resources to scientific applications or a commercial organisation that provisions resources to its business applications. The organisation wants to provision resources for its user applications in a way that guarantees acceptable response time.

The resources of the *local cluster* are managed by a Virtual Infrastructure Manager (VIM) such as Open Nebula [76] and Eucalyptus [133]. The VIM can start, pause, resume, and stop virtual machines on the physical resources offered by the cluster. The scheduling decisions at the cluster are performed by the *Scheduler*, which leases the site's virtual machines to the users. The scheduler also manages the deployment of virtual machines on a *Cloud Provider* according to provisioning strategies, which are detailed in the next

[1] http://www.3tera.com/

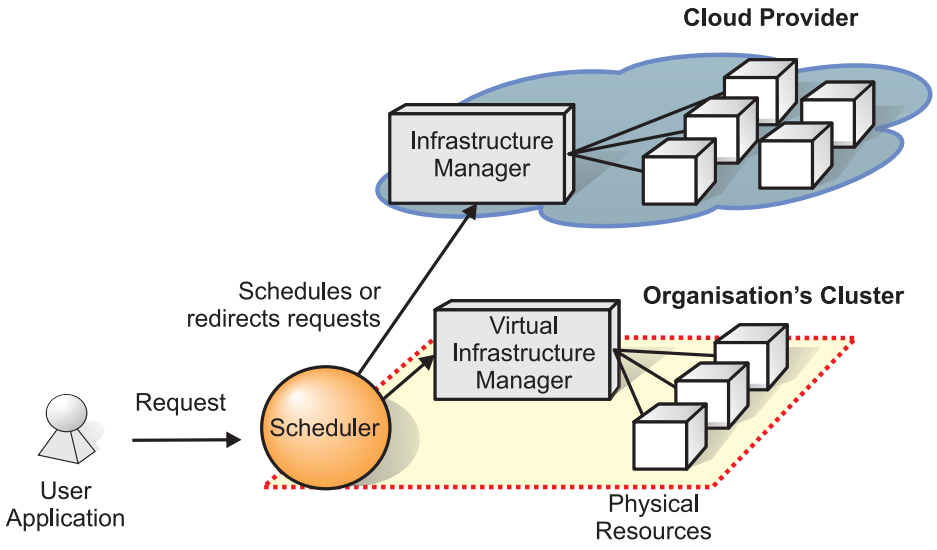


Figure 6.1: The resource provisioning scenario.

section. In order to deploy virtual machines on the Cloud provider, the scheduler uses the APIs offered by the provider, which are represented by the *Infrastructure Manager*.

6.3.1 Scheduling and Redirection Strategies

The strategies proposed and evaluated in this chapter define how the scheduler performs the scheduling of leases and when it borrows resources from the Cloud. The scheduler is divided into two sub-scheduler modules, one managing the scheduling of requests at the local cluster, hereafter termed the *Site Scheduler*, and another managing the scheduling on the Cloud resources, termed as the *Cloud scheduler*. In the system implementation, these sub-schedulers are actually two interconnected InterGrid Gateways (IGGs) that communicate their allocation decisions with one another.

We term a strategy or algorithm used by a sub-scheduler module to schedule the leases as a *scheduling strategy*. The algorithm that defines when the sub-scheduler managing the cluster borrows resources from the Cloud and which requests are redirected to the Cloud resources is called a *redirection strategy*. A combination of scheduling and redirection strategies is a *strategy set*. As discussed later in Section 6.4, a redirection strategy can be invoked at different times (*e.g.* a job arrival or completion) in different strategy sets.

6.3.2 Types of User Requests

The users of the infrastructure run different applications with different computing requirements. Some applications need resources at particular times to meet application deadlines, whereas other applications are not strict about the time when they are given resources to execute as long as they are granted the resources required. The first category of applications is termed as *deadline-constrained* and the second category is termed as *best-effort*.

For the purpose of this research, user requests are to be serviced by virtual machines hosted by an individual computing site; thus the same user request cannot receive re-

sources from both the Cloud provider and the organisation's cluster. Applications that rely heavily on message passing interfaces are generally sensitive to network delays and, despite advances in virtualisation technology [181], they may not benefit heavily from using resources from multiple computing sites. In practice, the execution of these applications is generally confined to an individual computer cluster.

6.4 Evaluated Strategy Sets

As described earlier, a strategy set consists of strategies for scheduling requests at the site and the Cloud, and a redirection strategy that specifies which requests are redirected to the Cloud. As scheduling strategies, we use conservative [128], aggressive [116], and selective backfilling [172]. This section summarises the request backfilling techniques, whereas Chapter 2 provides a more detailed discussion. With conservative backfilling, each request is scheduled (*i.e.* it is granted a reservation) when it arrives in the system, and requests are allowed to jump ahead in the queue if they do not delay the execution of other requests. In aggressive backfilling, only the request at the head of the waiting queue – called *the pivot* – is granted a reservation. Other requests are allowed to move ahead in the queue only if they do not delay the pivot. Selective backfilling grants reservations to requests that have waited long enough in the queue. Under selective backfilling a request is granted a reservation if its expected slowdown exceeds a threshold. The expected slowdown of a request r is also called *eXpansion Factor* (XFactor) and is given by Equation 6.1.

$$XFactor = (wait\ time + run\ time)/run\ time \tag{6.1}$$

In fact, this thesis employs the Selective-Differential-Adaptive scheme proposed by Srinivasan *et al.* [172], which lets the XFactor threshold be the average slowdown of previously completed requests.

6.4.1 Deadline Unaware Scheduling Strategies

The following strategy sets are considered for scheduling requests that arrive at the cluster:

Naïve: both Site and Cloud schedulers use conservative backfilling to schedule the requests. If the site scheduler cannot start a request immediately, the redirection algorithm checks whether the request can be started immediately using Cloud resources. If the request can start on the Cloud resources, then it is redirected to the Cloud, otherwise it is placed in the site's waiting queue.

Shortest Queue: requests at the site's cluster are scheduled in the First-Come-First-Served manner with aggressive backfilling [116]. The redirection algorithm executes as each request arrives or completes, and computes the ratio of virtual machines required by requests currently waiting in the queue to the number of processors available, similar to the work of England and Weissman [66]. If the Cloud's ratio is smaller than the cluster's, the redirection algorithm iterates the list of waiting requests and redirects requests until both ratios are similar.

Weighted Queue: this strategy is an extension of the Shortest Queue strategy. As each request arrives or completes, the scheduler computes the number of virtual machines required by waiting requests on the cluster, and the strategy computes how many virtual machines are in execution on the Cloud. The site scheduler then computes the number of virtual machines that can be started on the Cloud, num_vms, as the minimum between the number of virtual machines demanded by the site's requests and the Cloud's virtual machine limit, and redirects requests to the Cloud until num_vms is reached.

Selective: the local site uses the selective backfilling scheme described earlier. As each request arrives or completes, the scheduler checks which requests can be started, then initiates them. It then checks the sizes of the queues, using the same approach based on queue ratios used in the Shortest Queue strategy. If the queues have different ratios, the algorithm iterates the list of waiting requests and checks their XFactors. For each waiting request, if the expansion factor exceeds the threshold, the algorithm checks the potential start time for the request at both the Cloud and the site. The algorithm finally makes a reservation at the site that provides the earliest start time.

6.4.2 Deadline Aware Scheduling Strategies

This thesis also investigates strategies to schedule deadline constrained requests using resources from the site and the Cloud provider. The additional deadline aware strategies are:

Conservative: both local site and Cloud schedule requests using conservative backfilling. As each request arrives, the scheduler checks if the site can meet the request's deadline. If the deadline cannot be met, the scheduler checks the availability at the Cloud. If the Cloud can meet the request's deadline, then the request is scheduled on the Cloud resources. If the request deadline cannot be met, the scheduler schedules the request at the local site if it provides a better start time than the Cloud. Otherwise, the request is redirected to the Cloud.

Aggressive: both local cluster and the Cloud use aggressive backfilling to schedule requests. Similarly to the work of Singh *et al.* [167], as a request arrives the scheduler builds a tentative schedule. Using aggressive backfilling, it sorts the requests using an Earliest Deadline First (EDF) scheme and checks whether the acceptance of the request would break any deadline. If there are no potential deadline violations, the request is scheduled locally; otherwise, a schedule is built for the Cloud resources. If the request does not break deadlines of requests scheduled to use the Cloud, the request is served with resources from the Cloud provider. If the request deadline cannot be met, the scheduler schedules the request using the local cluster's resources if they provide a better start time than the Cloud. Otherwise the request is served by resources from the Cloud.

6.5 Performance Evaluation

This section describes the scenario considered for performance evaluation, the performance metrics, and experimental results.

6.5.1 Experimental Scenario

The evaluation of the strategies is performed using GridSim [31]. We use simulation because it enables us to perform repeatable experiments and the cost incurred by performing experiments on real infrastructure would be prohibitively expensive. To store the information about resources available for running virtual machines, the scheduler uses the data structure described in Appendix A.

The experiments model the San Diego Super Computer (SDSC) Blue Horizon machine because job traces collected from this supercomputer are publicly available[2] and have been studied previously [118]. The Blue Horizon machine comprises 144 nodes. The limit of virtual machines that the site can host is the same as the number of nodes. In addition, in this work the maximum number of virtual machines that can be in execution by a Cloud provider at a particular time is the same as the maximum in the local cluster.

To compute the cost of using resources from the Cloud provider, we use the amounts charged by Amazon to run basic virtual machines at EC2 (*i.e.* as of writing of this thesis the rate was US$0.10 per virtual machine/hour). The experiments in this chapter consider only the amount charged to run virtual machines, but in practice Amazon charges for the usage of other resources, such as network and storage. Other usage fees are not considered in this thesis because they depend on the applications' communication and data requirements. In addition, as Amazon commences charging users when the virtual machine process starts, the experiments consider that the booting time is already included into the request's duration.

6.5.2 Performance Metrics

Some metrics related to requests' response times include the bounded request slowdown ($bound$=10 seconds), hereafter referred only as request slowdown [74], and the Average Weighted Response Time (AWRT) [90]; described in Chapter 5. The AWRT measures how long on average users wait to have their requests completed. A short AWRT indicates that on average users do not wait long for their requests to complete.

$$AWRT = \frac{\sum_{j \in \tau_k} p_j \cdot m_j \cdot (ct_j - st_j)}{\sum_{j \in \tau_k} p_j \cdot m_j} \tag{6.2}$$

The AWRT is given by Equation 6.2, where m_j is the number of virtual machines required by request j, p_j is the execution time of the request, ct_j is the time of completion of the request and st_j is its submission time. The resource consumption ($p_j \cdot m_j$) of each request j is used as the weight.

In order to compute the benefits of using one strategy over another, we also compute the cost ratio between AWRT and the amount spent in running virtual machines on the Cloud. In addition, we measure the number of deadline violations and request rejections in scenarios where some requests are deadline constrained. More information about the ratios is provided along with respective experiments.

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

6.5.3 Experimental Results

The first experiment evaluates the performance improvement of different strategy sets by running virtual machines on the Cloud provider and the cost of such improvement in each case. This experiment uses a metric hereafter termed as *performance cost*. The performance cost of a strategy st is given by Equation 6.3.

$$perf.\ cost_{st} = \frac{Amount\ spent}{AWRT_{base} - AWRT_{st}} * AWRT_{st} \tag{6.3}$$

where $Amount\ spent$ is the amount of money spent running virtual machines on the Cloud provider, $AWRT_{base}$ is the AWRT achieved by a base strategy that schedules requests using only the site's resources and $AWRT_{st}$ is the AWRT under the strategy st when Cloud resources are also utilised. This metric aims to quantify the improvement achieved in AWRT and its cost. The smaller the performance improvement cost, the better the strategy performs. In the experiments described in this section, the base strategy is FCFS with aggressive backfilling.

For this experiment, the site's workloads have been generated using Lublin and Feitelson [118]'s model (*i.e.* Lublin99). Lublin99 has been configured to generate two-month-long workloads of type-less requests (*i.e.* no distinction is made between batch and interactive requests). The maximum number of CPUs used by the generated requests is set to the number of nodes in the cluster. This experiment evaluates the performance cost under different types of workloads. In order to generate different workloads, we modify three parameters of Lublin99's model, one at a time. First, we change the mean number of virtual machines required by a request (specified in log_2), which is set to $log_2 m - umed$ where m is the maximum number of virtual machines allowed in the system. We vary $umed$ from 1.5 to 3.5. The larger the value of $umed$, the smaller the requests become in terms of numbers of virtual machines required and consequently result in lighter loads. The second parameter changed in the experiments affects the inter-arrival time of requests at rush hours. The inter-arrival rate of jobs is modified by setting the β of the gamma distribution (hereafter termed $barr$), which we vary from 0.45 to 0.55. As the values for $barr$ increase, the inter-arrival time of requests also increases. The last parameter impacts the request duration by changing the proportion of the first gamma in the hyper-gamma distribution used to compute the requests' runtimes. The proportion p of the first gamma in Lublin99's model is given by $p = pa * nodes + pb$. We vary the parameter pb from 0.5 to 1.0. The larger the value of pb, the smaller the duration of the requests.

The results of this experiment are shown in Figure 6.2. Each data point is the average of 5 simulation rounds. Graphs (a), (b) and (c) show the site's utilisation under aggressive backfilling scheduling when the Cloud resources are not used. These illustrate the effect of the parameter changes on the load. Graphs (d), (e) and (f) show the performance cost when we vary: the number of virtual machines required by a request, the inter-arrival interval and the request's duration, respectively. The higher values obtained by the naïve strategy show that more money is spent to achieve an improvement in AWRT, especially under heavy loads, as shown in graph (d). From graphs (a) and (d), we also observe that the performance cost of using the Cloud is linear with the decrease in number of virtual machines of requests except the for naïve, which is very expensive for small requests. Under lighter loads, all strategies tend to yield the same ratio of cost and performance.

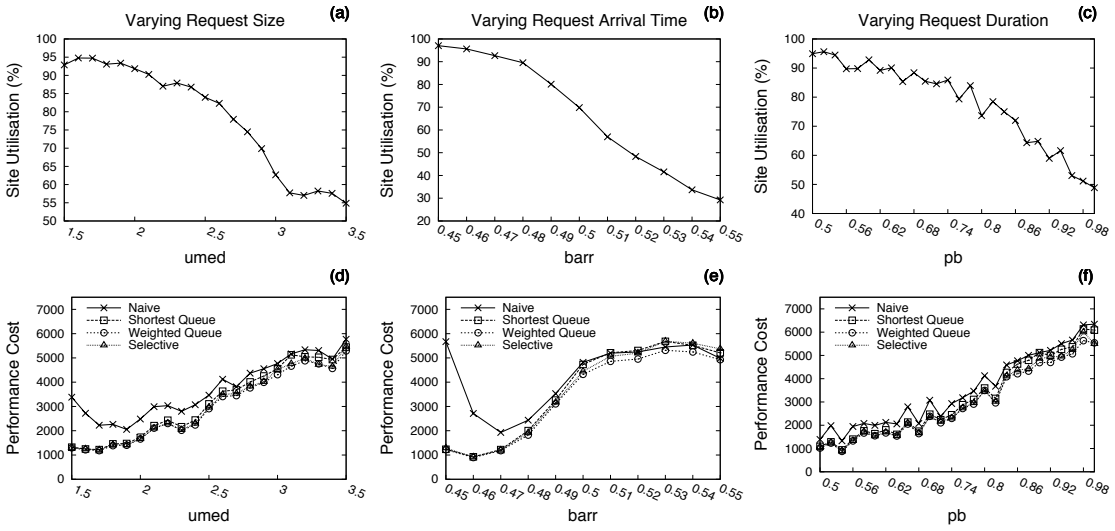


Figure 6.2: The top three graphs show the site's utilisation using the base aggressive backfilling strategy without Cloud resources; the bottom three graphs show the performance cost under different workloads. Higher values of $umed$ result in requests requiring larger numbers of virtual machines. The larger the value of $barr$, the greater the inter-arrival time of requests at rush hours. The time duration of the requests decrease as the value of pb increases. Each data point presented in the graphs is the average of 5 simulation rounds.

With small inter-arrival periods, all strategies have similar peformance, except the naïve strategy. The naïve strategy again produces a high performance cost, as shown in graph (e). With the variation of request arrival time, the experiments show a limit on the performance cost, which is close to 5,500. The cost increases until this limit and then decreases, due to the increase of the request inter-arrival time. More time between requests allows using fewer resources, which makes it more costly to rely on the Cloud to improve the request response time. For smaller inter-arrival time values, there is an important difference in cost of performance for the Naïve strategy in comparison to other strategies. In the last part of the experiment, graphs (c) and (f), all strategies return similar performance cost for the same request duration variation. The performance cost is inversely proportional to the cluster usage.

The second experiment evaluates the site using resources from the Cloud to meet service level agreements with consumers. In this experiment the requests have deadlines. We measure the cost of reducing deadline violations, or requests completing after their deadlines. The cost of reducing deadlines using a strategy st is given by Equation 6.4.

$$non - violation\ cost_{st} = Amount\ spent_{st}/(viol_{base} - viol_{st}) \tag{6.4}$$

where $Amount\ spent_{st}$ is the amount spent with Cloud resources, $viol_{base}$ is the number of violations using a base strategy and $viol_{st}$ is the number of violations under the evaluated strategy. The base policy is aggressive backfilling sorting the jobs for scheduling and backfilling in an EDF manner.

This experiment uses real job traces collected from the SDSC Blue Horizon machine to model the workload of the site's cluster. As the job trace spans a period of two years,

we divide it into intervals of two months each. For each experiment, we performed 5 simulation rounds using a different workload for each round. As the deadline information is not available in the trace, we use a Bernoulli distribution to select from the trace the requests that should have a deadline. In this way, a request read from the job trace file has a probability of being deadline constrained. The experiments consider different numbers of deadline constrained requests.

To generate the request deadlines we use a technique described by Islam *et al.* [103], which provides a feasible schedule for the requests. To obtain the deadlines, we perform the experiments by scheduling requests on the site's cluster without the Cloud using aggressive backfilling. After that, the deadline d_j of a request j is calculated using Equation 6.5:

$$d_j = \begin{cases} st_j + (ta_j * sf), & if\ [st_j + (ta_j * sf)] < ct_j \\ ct_j, & otherwise \end{cases} \tag{6.5}$$

where st_j is the request j's submission time, ct_j is its completion time, ta_j if the job's turn around time (*i.e.* the difference between the request's completion and submission times) and sf is a stringency factor that indicates how urgent the deadlines are. If $sf = 1$, then the request's deadline is the completion time under the aggressive backfilling scenario. We evaluate the strategies with different stringency factors (*i.e.* 0.9, 1.3 and 1.7 termed tight, normal, and relaxed deadline scenarios respectively).

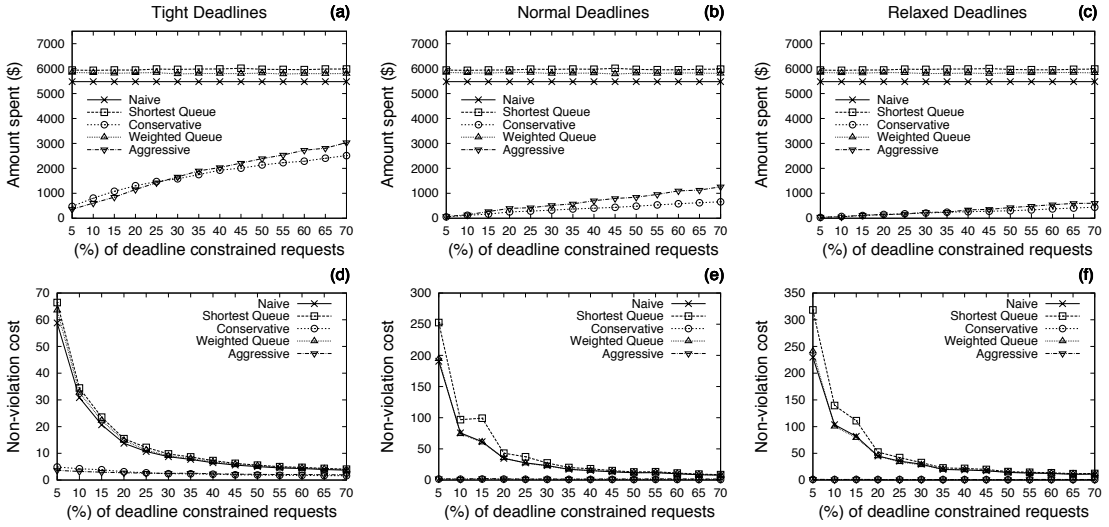


Figure 6.3: The top graphs show the amount spent using resources from the Cloud provider; the bottom graphs show the cost of decreasing deadline violations under different numbers of deadline constrained requests and different types of deadlines. Each data point is the average of 5 simulation rounds.

The results of this experiment are depicted in Figure 6.3. The top graphs show the amount spent using resources from the Cloud provider to reduce the number of deadline violations. The Conservative and the Aggressive deadline strategies spend smaller amounts than the remaining strategies because they are designed to consider deadlines. Other strategies – except the naïve – sort the requests according to deadlines; however

take into account other performance aspects such as minimising response time when redirecting requests to be scheduled on the Cloud. With a small proportion of deadline constrained requests with tight deadlines, the aggressive strategy had a smaller cost than the conservative strategy. With normal deadlines and a large number of deadline constrained requests, the aggressive strategy spends more than the conservative strategy.

We decided to evaluate the aggressive deadline strategy further in a scenario considering only the site's resources and a case considering the site and the Cloud. If the deadline of a request cannot be met, the request is rejected. This experiment evaluates how much the organisation would need to spend to decrease the number of requests rejected. The results are summarised in Figure 6.4.

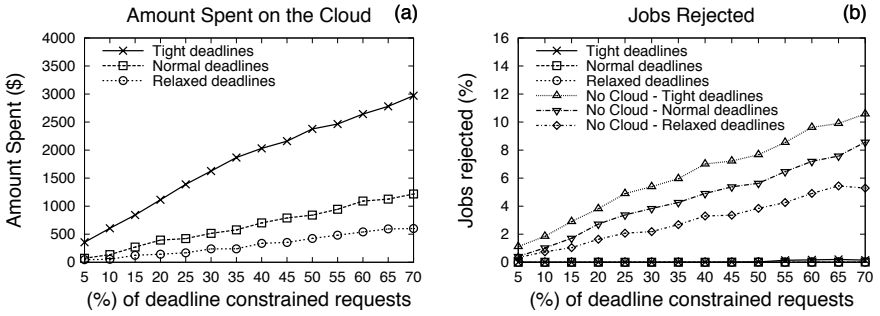


Figure 6.4: (a) amount spent using resources from the Cloud provider; (b) the decrease of requests rejected. Each data point is the average of 5 simulation rounds.

Figure 6.4 (a) shows the amount spent on the Cloud and (b) depicts the percentage of requests rejected when the Cloud is used and not used. An amount of US$3,000 is spent on the Cloud to keep the number of requests rejected close to zero under a case where 70% of the requests have deadlines. With normal deadlines, the strategy did not spend more than US$1,500 in any quantity of deadline constrained requests.

Again using traces from the SDSC Blue Horizon, the last experiment evaluates the amount of money spent using the Cloud infrastructure under different scheduling strategies, and compares the improvement of the strategies to a scenario where requests were scheduled using only the site's resources under aggressive backfilling. Table 6.1 summarises the results. All strategies perform similarly in terms of AWRT improvement. However, the proposed strategy set based on selective backfilling yields a better ratio of slowdown improvement to amount of money spent for using Cloud resources.

Table 6.1: Performance using workload traces (averages of 5 simulation rounds).

Metric description	**Naïve**	**Shortest Queue**	**Weighted Queue**	**Selective**
Amount spent with VM instances ($)	5478.54	5927.08	5855.04	4880.16
Number of VM instances/Hours	54785.40	59270.80	58550.40	48801.60
AWRT (improvement)	15036.77	15065.47	15435.11	14632.34
Overall request slowdown (improvement)	38.29	37.65	38.42	39.70

The experimental results described in this section show that the cost of increasing the

performance of application scheduling is higher under a scenario where the site's cluster is under-utilised. However, the cost-benefit of using a naïve scheduling strategy can be smaller than using other approaches as a large cost is incurred under scenarios of high system utilisation. In addition, request backfilling and redirection based on the expansion factors (*i.e*, selective backfilling) have shown a good ratio of slowdown improvement to amount of money spent for using Cloud resources.

6.6 Conclusion

This chapter considered the case of an organisation that operates its computing infrastructure, but wants to allocate additional resources from a Cloud infrastructure. The experiments evaluated the cost of improving the performance under different strategies for scheduling requests at the organisation's cluster and the Cloud provider. Naïve scheduling strategies can result in a higher cost under heavy load conditions. Experimental results showed that the cost of increasing the performance of application scheduling is higher under a scenario where the site's cluster is under-utilised. In addition, request backfilling and redirection based on the expansion factors (*i.e* selective backfilling) showed a good ratio of slowdown improvement to the money spent for using Cloud resources.

As shown in this chapter, commercial providers are increasingly relying on virtualisation technology to manage their infrastructure. The model presented in this chapter has used a lease abstraction, which has also been followed in realising the architecture as discussed in Chapter 7.

Chapter 7

Realising the InterGrid

This chapter presents the realisation of the architecture for interconnecting Grids. The resulting system aims to provide an execution environment for running applications on top of interconnected Grids. The system uses virtual machines to construct distributed virtual environments that span multiple computing sites across the Grids. A distributed virtual environment, also termed execution environment, is a network of virtual machines created to fulfil the requirements of an application, running isolated from other execution environments.

7.1 Introduction

The complexity of deploying applications in Grid environments has increased due to the hardware and software heterogeneity of the resources provided by the organisations within a Grid. This scenario makes efforts on interconnecting Grids even more difficult. Recently, virtualisation technologies have facilitated the deployment of applications. Virtual Machine (VM) technologies have enabled the creation of customised environments atop a physical infrastructure and the emergence of new models such as Infrastructure as a Service (IaaS) and Cloud computing [9, 193].

As described in Chapter 6, these factors can result in the creation of execution environments that span both commercial and non-commercial computing sites. Furthermore, virtualisation technologies minimise previous barriers to the inter-operation of Grids, such as the execution of unknown applications and the lack of guarantees over resource control, and can ease the deployment of applications spanning multiple Grids, by allowing resource control in a containment manner [146]. Under this scenario, the resources one Grid allocates from another are used to deploy virtual machines that run isolated from the physical host's operating system.

To realise the InterGrid architecture, we consider a scenario where applications are deployed across several computing sites, or Grids, by means of virtualisation technologies. These applications run on networks of virtual machines, or execution environments, created on top of the physical infrastructure. It is therefore essential to provide a system that allows the allocation of resources to run virtual machines across multiple sites. The work in this chapter describes design and implementation details of the InterGrid. The work realises the architecture proposed in Chapter 3, and describes a system that enables the

allocation of virtual machines and the deployment of applications at multiple computing sites.

7.2 Requirements and Component Interactions

During the system design and implementation, we opted for using virtual infrastructure managers that enable the management of virtual machines on physical resources (discussed in Section 7.3). This section presents the requirements of the resource management system employed by the provider sites, the Distributed Virtual Environment (DVE) Manager utilised by users to instantiate a DVE, and the InterGrid Gateway (IGG) that represents a Grid or organisation participating in the InterGrid.

The InterGrid expects a minimum set of features from the resource provider. It expects the resource provider to supply information about the availability of resources. In addition, when the IGG presents a permission to the provider, the latter must be able to allocate and initialise the resources, which can correspond to fetching the disk image required by the virtual machine and performing initial network configuration.

The major requirements of the InterGrid are organised according to its main components. The features of the resource provider are as follows:

- Collect information from the resource management system at the provider's site.
- Supply information about the availability of resources to IGGs as *free time slots*. The decision on which resources to offer to the IGG is performed based on the provider's provisioning policies.
- Initialise resources and perform initial host and network configuration.

The main features of IGGs are to:

- Receive free time slots and update the *resource repository*.
- Select and assign resources to DVEs based on the Grid-level provisioning policies.
- Negotiate upon and acquire resources from other IGGs.
- Provide resources to other IGGs based on the IGG's resource sharing policies.

DVE Managers present the following requirements:

- Handle requests from a user application.
- Acquire resources from the InterGrid.
- Contact the resource provider sites to utilise the resources allocated by IGGs.
- Deploy services on the resources allocated.
- Manage resources allocated to the DVE, trigger allocation of additional resources, or release resources.

7.3 System Design and Implementation

This section provides details about the implementation of the InterGrid. We first justify our design decision on using virtual machine technology. After that, we describe the design and implementation choices on the InterGrid Gateway. Later, we describe the Virtual Machine Manager, which interacts with virtual infrastructure managers such as Open Nebula [76] and IaaS providers such as Amazon Elastic Compute Cloud (EC2) [4]. After that, we explain how the DVE Manager works.

7.3.1 Virtualisation Technology

The use of virtual machines [15] in distributed systems brings several benefits such as:

- Server consolidation, allowing workloads of several under-utilised servers to be placed in fewer machines.
- The ability to create virtual machines to run legacy code without interfering in other applications' APIs.
- Improved security through the creation of sandboxes for running applications with questionable reliability.
- Dynamic provision of virtual machines to services, allowing allocation of resources to applications on the fly.
- Performance isolation, thus allowing providers to offer some levels of guarantees and better quality of service to user applications.

As discussed in Chapter 2, existing systems can manage a physical infrastructure by enabling users to create virtual workspaces [108] or virtual clusters [38, 76, 77, 133] atop the actual physical infrastructure. These systems can bind resources to virtual clusters or workspaces according to the demands of user applications. They also provide an interface through which the user can allocate virtual machines and configure them with the operating system and software of choice. These resource managers allow the user to create customised virtual clusters using shares of the physical machines available at the site.

Virtualisation technology minimises some security concerns inherent to the sharing of resources among multiple computing sites. Although users can have administrative rights on the operating system running on their allocated virtual machines, in some categories of virtualisation users do not have rights on the host's operating system. Due to the benefits of using virtual machines, we decided to use virtualisation technology to enable the creation of the execution environments described earlier in this thesis. In addition, relying on virtual machines eases the deployment of the execution environments on multiple computing sites. A user application can have better control over the update of software and libraries installed on the resources allocated from the sites without compromising the operation of the hosts' operating systems.

7.3.2 Virtual Infrastructure Managers (VIMs)

The IGG utilises Open Nebula [76] to manage the virtual machines at computing sites. Open Nebula, termed here as a Virtual Infrastructure Manager (VIM), provides a software layer between a service and physical infrastructure, which enables the dynamic provision of resources to services. This software layer brings several benefits to the resource provider as it allows for dynamic resizing of the physical infrastructure, and partitioning of the managed cluster. OpenNebula allows a user to start, manage, and stop virtual machines according to the provisioning policies in place. The IGG is also able to deploy virtual machines on Amazon EC2 [4].

7.3.3 InterGrid Gateway

The InterGrid Gateway has been implemented in Java. The main components of the IGG are depicted in Figure 7.1.

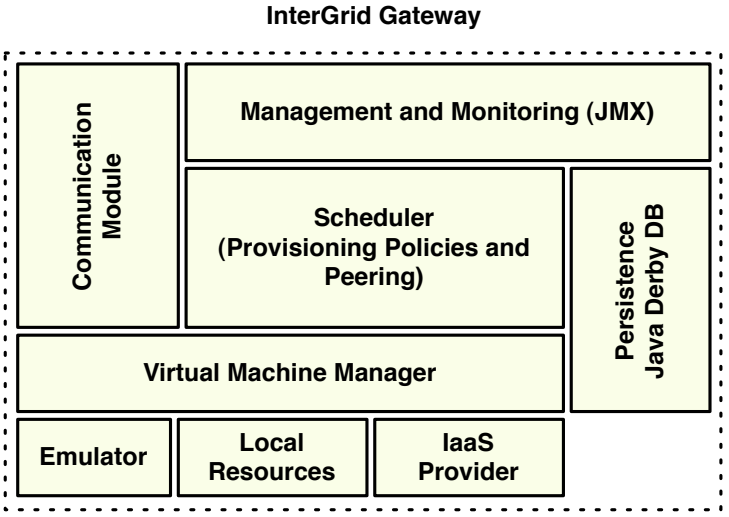


Figure 7.1: Main components of the InterGrid Gateway.

Communication Module: is responsible for message passing. This module receives messages from other entities and delivers them to the components registered as listeners for those message types. This module also provides the functionality to send messages, allowing system entities to communicate with one another. Message-passing makes gateways loosely coupled and allows for more failure-tolerant communication protocols. In addition, sender and receiver are de-coupled, which makes the system more resilient to traffic bursts. Figure 7.2 describes the design of the communication module. The messages are handled by one central component, the *Post Office*, which associates each incoming message with a thread that in turn forwards the message to all listeners. Threads are provided by a thread pool. If the thread pool is empty when a message arrives, the Post Office puts the message in a queue to wait for an available thread. *Listeners* are message handlers. Each listener is notified at the arrival of a new message; a listener can decide to handle the message or not based on the message type.

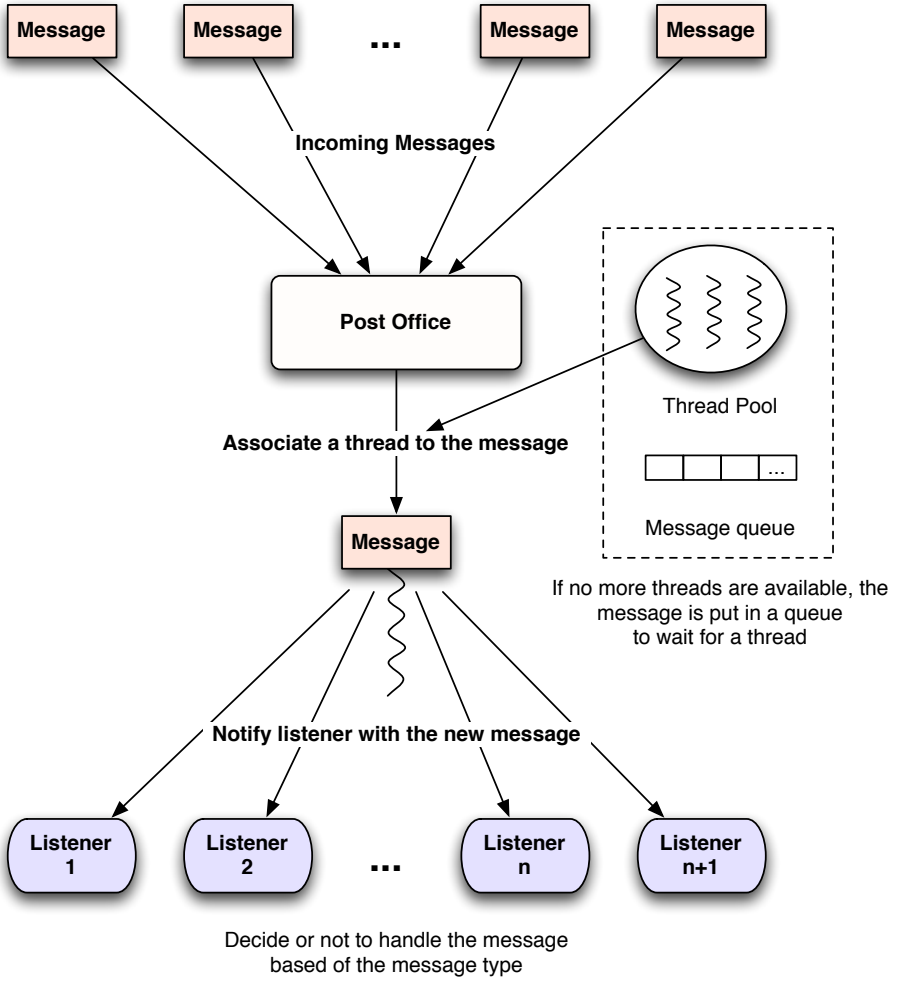


Figure 7.2: Design of the communication module.

Management and Monitoring: the management of the IGG is performed via Java Management Extensions (JMX).[1] JMX is a standard API for management and monitoring of resources such as Java applications. It also includes remote access, hence a remote program can interact with a running application for management purposes. Remote access handles secured user identifications and authorisations. Typical uses of the JMX are querying or changing application configuration, accumulating statistics about application behaviour, and notification of state changes.

The IGG exports, via JMX, management operations such as establishment of peering agreements, connection and disconnection to other IGGs, shutdown, and management of virtual machine services. These operations are accessible through JConsole, which is a graphical client provided by the standard distribution of Java to connect to any application using JMX. In addition, we provide a command line interface that interacts with the components via JMX. The use of JMX enables further instrumentation and management of the gateway by third party management tools.

Persistence: the IGG uses a relational database in order to guarantee the persistence of in-

[1] http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/

formation. The database persistently stores information such as peering arrangements and templates for virtual machines (explained in Section 7.3.4). The IGG uses the database provided by the Apache Derby project.[2]

Scheduler: this component comprises other sub-components, namely *resource provisioning policy*, *peering directory*, *request redirection*, and *enactment module*. The enactment module interacts with the Virtual Machine Manager to create, start, or stop virtual machines to fulfil the requirements of the scheduled requests. The Virtual Machine Manager is described in the next section. The Scheduler maintains the availability information obtained from the Virtual Machine Manager and schedules requests for virtual machines. The Scheduler keeps track of the resources available using the same data structure utilised by the discrete-event simulations described in previous chapters.

7.3.4 Virtual Machine Manager (VM Manager)

The VM Manager is the link between the IGG and the resources. As described beforehand, the IGG does not share physical resources directly, but depends on virtualisation technology to abstract them. Hence, resources allocated by the IGG to a DVE are virtual machines. The VM Manager relies on a VIM, which controls the virtual machines on a set of physical resources. Typically, VIMs are able to create, pause, resume, and stop virtual machines on a physical cluster. In addition, the VM Manager controls the deployment of virtual machines on an IaaS provider. Before describing the VM Manager, this section introduces the concept of virtual machine templates and a directory service.

Virtual Machine Template: Open Nebula's terminology is used to explain the idea of templates for virtual machines. A template is analogous to a computer's configuration. It describes a type of virtual machine, and contains the following information:

- Number of cores or processors to be assigned to the virtual machine.
- Amount of memory required by the virtual machine.
- Kernel used to boot the virtual machine's operating system.
- The disk image that contains the operating system.
- Price for using a virtual machine of this type for one hour.

The information provided in a virtual machine template is static, described once and reused every time a new virtual machine is created. A virtual machine under deployment or in execution is termed an *instance*. The administrator responsible for the IGG provides this static information at the set-up phase of the infrastructure. Moreover, the administrator can update, insert, or remove templates at any time. However, as of writing this chapter, each IGG in the InterGrid network should have a list of templates that are common across all IGGs; that is, a template created at one IGG must have equivalent templates at other IGGs.

During the deployment of a virtual machine, the VM Manager creates a *virtual machine descriptor*. The descriptor contains the information from the original template and

[2] http://db.apache.org/derby/

additional information about the virtual machine instance under deployment or in execution. The additional information is:

- The actual path to the disk image that contains the file system for the virtual machine.
- The IP address of the physical machine that hosts the virtual machine. This information is required in testbeds with a mixture of public and private IP addresses.
- The network configuration of the virtual machine.
- For deploying a virtual machine on an IaaS provider, the required information to access the remote infrastructure such as account information.

The deployment process configures the network interface for the virtual machine. The VM Manager module currently works with a pool of Media Access Control (MAC) addresses and corresponding IP addresses. At the deployment phase, the VM Manager clones the template, adds the necessary network configuration to it, such as the MAC address, and converts the template to the format recognised by the underlying VIM. In the current implementation, the only information added to the template passed to the VIM is the MAC address. We rely on Dynamic Host Configuration Protocol (DHCP) to guarantee that the instance with the assigned MAC address is configured with the corresponding IP address from the pool.

In addition, the VM Manager updates the disk image information obtained from the template with the path to a copy of the original image. This allows the system to deploy several instances of the same template without requiring the instances to share the same file system. There are implementation details that are specific to each interface with underlying VIMs. For example, the VM Manager that interfaces with OpenNebula builds an initial pool of disk images. When a virtual machine is requested, the VM Manager assigns a disk image copy from the pool.

When deploying a virtual machine on an IaaS provider, not all the information described earlier is required. For example, network information such as MAC and IP addresses are not mandatory for using Amazon EC2 because a public IP address is automatically assigned by EC2. In addition, EC2 is responsible for seamlessly cloning the disk images for running several instances of the same template in parallel. However, before creating an instance on Amazon EC2, the disk image must be uploaded to Amazon's infrastructure. Currently, we upload the disk image beforehand and ensure that all templates registered in the IGG have their disk images in EC2.

Virtual Machine Template Directory: the IGG works with a repository of virtual machine templates; that is, templates can be registered to the repository by the administrator responsible for the IGG. A user or a DVE manager, can request instances of virtual machines whose templates are registered on the IGG's repository. In addition, the IGG administrator has to upload the images to Amazon if the IGG uses the Cloud as a resource provider. At the moment, it is not possible for the user to submit her own virtual machine templates or disk images to the IGG.

Virtual Machine Manager: allows the gateway to submit and deploy virtual machines on a physical infrastructure, and interacts with a VIM to create or stop virtual machines on

the cluster. The VM Manager implementation is generic to connect with different VIMs. We have developed connectors to Open Nebula and Amazon EC2. The connection with Open Nebula uses a Java API to submit and stop virtual machines and to transform our virtual machine template to the format that Open Nebula recognises. Open Nebula runs as a daemon service on a master node, hence the VM Manager works as a remote user of Open Nebula.

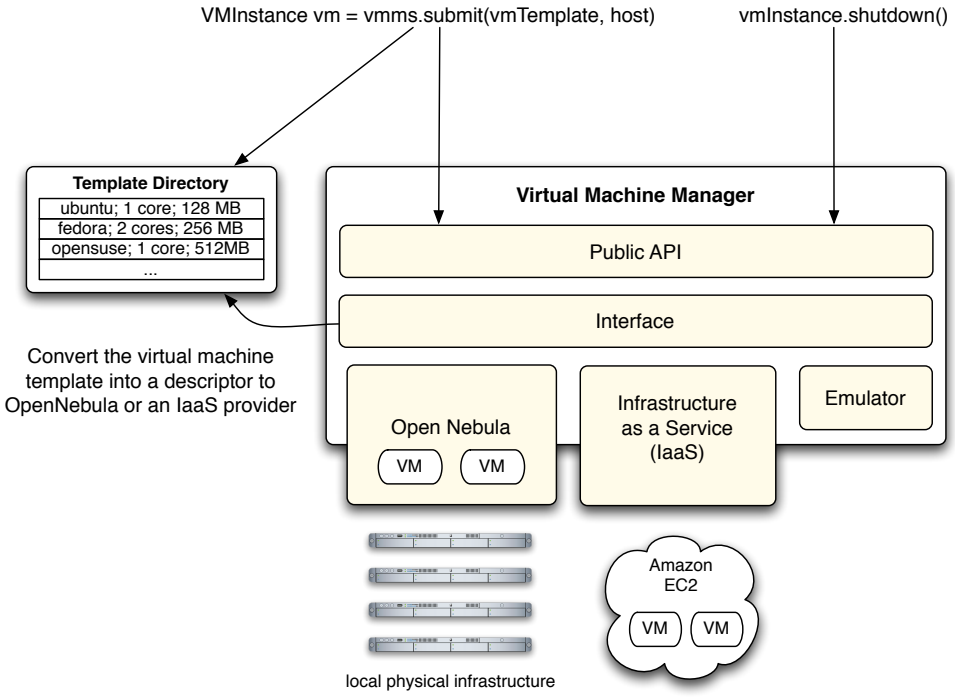


Figure 7.3: Design of the Virtual Machine Manager Service.

Figure 7.3 shows the interaction between the IGG, the template directory, and the VM Manager. We have also implemented a connector to deploy virtual machines on Amazon EC2. The connector wraps the command line tools provided by Amazon. In addition to the Open Nebula and Amazon EC2 connectors, we have developed a VIM emulation for testing and debugging purposes.

7.3.5 Distributed Virtual Environment Manager (DVE Manager)

A DVE Manager interacts with the IGG by making requests for virtual machines or querying their status on behalf of the user application it represents. Once a request is sent to the IGG, the DVE Manager starts to query the IGG to obtain the request status. The communication is asynchronous and a call back mechanism is not provided at the moment because the DVE Manager may not be in execution during the whole request's life cycle. For example, a DVE Manager can make reservations for virtual machines for a few days in the future, stop its execution, and restart when the start time of the reservations approaches.

A virtual machine request can be in one of the following statuses:

- **Unknown:** the request has been created, but has not been submitted to any IGG.

- **Pending:** an IGG has received the request, but has not scheduled it yet. The IGG's scheduler is either waiting for a free time slot in which the request can be placed, or is trying to redirect the request to another IGG.
- **Scheduled:** the request has been scheduled by an IGG. The potential start and finish times for the request have been established.
- **Cancelled:** either the user or the IGG have cancelled the request's execution.
- **Failed:** the request has failed due to an error, which is reported by the IGG.
- **In progress:** the request has started, the virtual machines have been created, and the binding information has been obtained.
- **Completed:** the request's finish time has lapsed and the request has finished.

At the moment, when the reservation starts, the DVE Manager receives a list of virtual machines as tuples of public IP/private IP, with which the DVE Manager sets Secure Shell (SSH) tunnels[3] to access the virtual machines. The DVE manager then handles the deployment of the user applications. With EC2, virtual machines have public IPs, hence the DVE can access the virtual machines directly without tunnels.

7.4 Conclusion

This chapter presented the realisation of the architecture for interconnecting Grids. The system prototype relies on virtualisation techniques because they minimise security concerns present in the sharing of resources between Grids. Resources allocated by an InterGrid Gateway from another are used to run virtual machines on which the user applications are deployed.

The InterGrid Gateway relies on virtual infrastructure managers to control the execution of virtual machines on a cluster of physical machines. This architecture can also be used in scenarios such as those described in Chapter 6 where the capacity of an organisation's cluster is extended by allocating resources from commercial providers. The provided implementation of InterGrid Gateways can allow the co-ordination required to enable the use of commercial and non-commercial infrastructures. In this way, the system provides the means to create execution environments for running applications on top of interconnected Grids. These Grids can comprise commercial and non-commercial resource providers.

[3] http://en.wikipedia.org/wiki/Secure_Shell

Chapter 8

Conclusions and Future Directions

8.1 Discussion

We began this PhD research with one general challenge: the deployment of applications using resources from multiple computational Grids. We opted to approach the challenge of deploying applications across Grids from a resource management perspective, thus this thesis focused on mechanisms that can allow Grids to share computational resources. Solutions for resource sharing between computational Grids need to meet requirements such as interoperability between Grid middleware, interfaces with existing Grid resource management systems, provision of a general infrastructure and dynamic collaborations, use of a decentralised architecture, respect for administrative separation, deal with two levels of resource contention, and provide incentives for interconnected Grids.

This thesis then investigated existing work on resource management systems for clusters and Grids, and enumerated several characteristics of existing systems such as their architectural views, operational models, arrangements between schedulers, resource control techniques, and support for virtual organisations. This investigation revealed:

- An extensive literature on job scheduling among clusters (*i.e.* within a Grid);
- Some efforts on interoperability between Grid middlewares; and
- Attempts to federate large-scale testbeds.

The investigation revealed a lack of mechanisms that build on these efforts, enable resource sharing between Grids, and meet the requirements described earlier. These lessons led to a proposed architecture for interconnecting Grids. The proposed architecture is based on InterGrid Gateways that mediate access to the resources from the interconnected Grids. The architectural views contribute to the Grid computing area by enhancing and building on existing work on job scheduling within Grids. This architecture is inspired by observing the manner in which Internet Service Providers (ISPs) establish peering arrangements in the Internet. ISP peering arrangements define the terms under which they allow traffic into one another's networks. The concept of peering arrangements, which has allowed the Internet to grow to its current stage, was missing in Grid computing. Therefore, the proposed architecture considers that Grids establish peering arrangements with

one another, and that these arrangements define the conditions under which resources are shared by the Grids.

Enabling resource sharing between Grids, and the consequent use of resources from multiple Grids by applications, has proven difficult. The resources within a Grid are generally clusters of computers managed by queue-based resource management systems. Moreover, these clusters of computers generally agree to provide resources to a Grid, but have local users whose demands also have to be satisfied. This scenario leads to contention for resources between the Grid applications and the providers' local users. To enable resource sharing between Grids, each individual Grid should be able to estimate the availability of resources from its providers in order to accommodate the requests of its Grid users and from other Grids. Further on, this thesis focused on improving the provisioning of resources to applications within Grids.

With respect to improving resource provisioning within Grids, the thesis proposed extensions to existing batch schedulers that ease the gathering of availability information to provision resources between Grids. The thesis also investigated provisioning techniques based on multiple resource partitions that enable providers to offer resources to the Grid and satisfy the requirements of their local users. In addition, experiments evaluated the reliability of the availability information obtained with different strategies and the impact of this reliability in resource provisioning within a Grid. Simulation results obtained by modelling the DAS-2 Grid environment showed that it is possible to provision resources to Grid applications without requesting providers to disclose their resource availability information too frequently. A very small number of violations (*i.e.* cases of contentions for resources) and improved job slowdown are achieved if the information is disclosed every 15 to 30 minutes. These results showed an overhead reduction compared to other techniques wherein the availability information is obtained upon the schedule of each individual request. Moreover, the provisioning strategies based on multiple resource partitions improved the job slowdown of both providers' local jobs and Grid jobs, thus showing the viability of such approaches to enable provisioning of resources to Grid applications and further resource sharing between Grids.

This exercise on provisioning policies later motivated the investigation of a load management mechanism to provide resource sharing between Grids. The mechanism relies on the resource availability information obtained using the provisioning strategies described earlier; information ultimately used to guide InterGrid Gateways on decisions about resource request redirection. Aware that Grid infrastructures are expensive to maintain and operate, the proposed resource sharing mechanism redirects requests based on their marginal cost of allocation. Simulation results showed that overall Grids have incentives to interconnect, demonstrated for example by slowdown improvements of Grid jobs. The experiments also showed that the mechanism is efficient in balancing the load across the interconnected Grids. The proposed provisioning strategies described earlier yielded good performance, and interestingly, improved the slowdown of jobs even under scenarios with no interconnection between Grids. These improvements with no Grid interconnection were achieved when compared to aggressive backfilling, which is a strategy used frequently by Grid resources. The interconnection of Grids, therefore, showed a positive impact on the execution of Grid applications.

While investigating mechanisms to enable sharing resources between Grids, we noticed increasing availability of commercial infrastructures. These infrastructures are of-

fered to users by resource providers in a pay-per-use manner and are gaining popularity. Evidence has shown that these infrastructures can be suitable for executing certain types of Grid applications [48, 138]. This thesis later investigated scenarios wherein commercial infrastructure can be used to improve resource provisioning in Grids. Under these scenarios, Grids are composed of resources from commercially available providers and academic institutions. This thesis investigated a provisioning scenario where the capacity of clusters of computers is expanded by allocating resources from commercial infrastructure, giving consideration to the cost of performance improvements obtained by using commercial infrastructure. The thesis further investigated whether different provisioning strategies yield different ratios of performance improvement and money spent using resources from commercial providers. Simulation results showed that strategies can indeed have different costs for the performance improvement they yield. Strategies based on selective backfilling can have better slowdown improvements using fewer resources from the commercial providers. Experiments with deadline constrained requests demonstrated that for a cluster of 144 nodes and any combination of deadline urgency and number of deadline constrained requests, the amount of money spent to keep the number of rejected requests close to zero over a two month period, remained below US$3,000. This represents a small cost when operating an infrastructure of 144 nodes.

Furthermore, this thesis provided the building blocks to propose and evaluate mechanisms for deploying applications in environments that are a mixture of commercial and non-commercial infrastructures. The study on commercial providers lays the basis to investigate adaptive provisioning techniques that strike the balance between performance improvement and money spent with commercial infrastructure. This research also provides a basis for resource provisioning in scenarios where a mixture of commercial and non-commercial infrastructure is used.

We realised the proposed architecture by presenting a system prototype that relies on virtual machine techniques to accommodate user requests. Resources exchanged by Grids are used to deploy virtual machines. Some design decisions, such as the use of virtualisation technology, were made because they minimise some security concerns inherent to the sharing of resources among organisations and consequently, the sharing of resources between Grids. The proposed system can allocate virtual machines from different sites by utilising Virtual Infrastructure Managers (VIMs) such as OpenNebula [76]. These VIMs are used to start, pause, and stop virtual machines. The system also enables the management of virtual machines at commercially available resource providers who use virtualisation solutions to manage their infrastructures.

8.2 Future Directions

Some research topics relevant to the interconnection of Grids and the deployment of applications on multiple Grids have not been addressed in this thesis. Some of these issues have been discussed in previous work [46]; others were identified here. This thesis has focused on resource provisioning and sharing, thus leaving future work in the following areas:

8.2.1 Resource Discovery and Availability Information

The InterGrid can use discovery systems and networks to locate resources to execute a job. This thesis focused on computational resources and considered a resource provisioning model where providers either (i) inform the InterGrid Gateway about the resource availability or (ii) the gateway queries providers for the resource availability. As the providers have their local users, the gateways have partial information about the actual availability of resources. Gateways further use this information to redirect requests across Grids.

One important issue to investigate is an approach to disseminate the availability information across Grids in a manner that is useful for provisioning resources to deploy execution environments spanning multiple Grids. Gossiping protocols can be relevant to disseminate the information on resource availability [68]. Alternatively, gateways can make provisioning decisions based on probabilistic resource claims that are verifiable and traceable back to the issuer [86]. The proposition here is that resource discovery networks based on imprecise information such as that disseminated by gossiping protocols can produce acceptable provisioning solutions and improve the performance of Grid applications.

8.2.2 Co-ordination Mechanisms

As described by Boghosian *et al.* [20], to execute applications on multiple Grids it is necessary to have some degree of co-ordination between Grids. However, this co-ordination is currently only achieved at a person-to-person instead of a system-to-system level. Thus, it is important to investigate mechanisms that enable the co-ordination between Grids in order to co-allocate resources.

8.2.3 Virtualisation Technology and Commercial Resource Providers

The use of virtualisation technology in Grid computing brings several benefits, but also creates a new set of challenges. If physical resources are used to run virtual machines, which in turn are used to execute applications and services, we have two instances of the allocation problem: the first being where physical resources are provisioned to virtual machines, the second being where applications are scheduled on available virtual machines. Furthermore, the deployment of virtual machines requires the transfer of disk image files. These conditions can be seen simply as new constraints of the scheduling problem, but in practice they impact the performance of applications.

Some systems consider that image files required by virtual machines are pre-deployed on all computing sites. In future work, it would be interesting to relax this assumption and enable users to upload their virtual machine images.

When using commercial infrastructure, it would be important to investigate adaptive provisioning strategies that can strike a balance between performance improvements and the money spent. In this way, under peak-load conditions, a resource provider can opt for allocating resources from a commercial infrastructure to honour existing user contracts.

In addition, it would be relevant to study the performance of Grid applications in these scenarios. If Grids incorporate commercial providers, it is important to evaluate the performance of applications such as bags-of-tasks under scenarios that mix commercial and non-commercial organisations.

8.2.4 Application Models and Distributed Virtual Environments

Another important area of future study relates to application models that can benefit from using resources from multiple Grids. The InterGrid environment requires application models that are able to adapt to the dynamicity of the environment and report the need of resources to management entities [184]. We envision that the execution environments described in this thesis can increase or decrease their allocations based on the needs of the applications they encapsulate. However, allocating resources to these environments depends on several factors such as:

- Cost of using the resources;
- Time and overhead for changing allocations; and
- Peering arrangements established between Grids.

This thesis has neither focused on application models nor on the adaptation of the execution environments to the demands of the applications and the conditions of multi-Grid environments.

8.2.5 Fault Tolerance

The issues described here are also related to the topics discussed earlier. In multi-Grid environments, resource failures can occur for various reasons: variations in the configuration of the environment, non-availability of required virtual machines, overloaded resource conditions, and faults in computational and network fabric components [75]. One way of dealing with this problem is to find alternative resources and restart or migrate the execution of applications. Current check-pointing mechanisms based on the job abstraction may not be enough, as the migration of execution environments requires the check-pointing of virtual machines. As the migration of virtual machines in local area networks becomes more reliable, virtual machine technology can be used to provide the means for migrating execution environments and the recovery from failures. However, these requirements demand advancements both in strategies and mechanisms for handling fault tolerance in applications for the InterGrid, which have not been addressed in this thesis.

8.2.6 Peering Arrangements and Policy Enforcement

This thesis has considered that Grids have pre-defined peering agreements that specify the conditions under which they exchange resources. This assumption is based on the fact that interconnecting Grids can require negotiations over network links and resources that are made available to users of each Grid.

It would be relevant to consider transitive relationships between the InterGrid gateways, in which one Grid A can allocate resources from a Grid C via an agreement with a third Grid B. That is, Grid A has an agreement with B, which in turn has an agreement with C. Grid A could access Grid C's resources if a delegation of resources has been made previously by Grid C to Grid B and Grid B decides to grant access to Grid A over the resources of Grid C.

In addition, Grids can have internal policies that define how many resources can be used by a given user group or virtual organisation. Enforcing allocation policies within a Grid is a challenge that becomes more complex if peering arrangements between Grids are considered. These issues have not been addressed in this thesis and would be relevant to future work. Other efficient mechanisms for resource sharing between Grids are also worth investigating.

Moreover, similarly to the peering arrangements between ISPs, Grids can have asymmetric agreements that define bulk shares of resources allocated from one another. The investigation of these peering relationships and their implications on applications' performance are further relevant topics to investigate.

8.2.7 Implementing Provisioning Strategies on a Real Grid Testbed

The evaluation of the resource provisioning and peering strategies proposed in this thesis was carried out through discrete-event simulation. In future work, it would be relevant to implement some of the proposed strategies on a real Grid testbed. In addition, early attempts have been made to evaluate the scenario considered in Chapter 6 using the real system implementation proposed in Chapter 7.

Appendix A

A Data Structure to Facilitate Provisioning

This appendix presents details about a data structure for managing job scheduling and reservations for computing resources in clusters of computers and virtualised infrastructures. The data structure uses a red-black tree whose nodes represent the start times or completion times of application jobs and reservations. The tree is enhanced by a linked list that facilitates the iteration of nodes once the start time of a reservation is found using the tree. We present an implementation of this data structure suitable for both simulations and real systems, describe its main features and provide an example on its use.

A.1 Introduction

Deadline-constrained applications demand predictable Quality of Service (QoS) [168], often requiring a number of computing resources to be available over a period, commencing at a specific time in the future. Advance reservations are attractive in these scenarios as they provide means for reliable allocations and allow users to plan the execution of their applications.

Recent advances in computing technology have led to the emergence of systems that rely on virtual machines (VMs) for allocating resources to user applications according to their demands. For their scheduling decisions, these systems generally maintain information about resource availability in data structures or databases [104]. Resource management systems often handle numerous requests per minute, with each job arrival or completion triggering scheduling operations requiring several accesses to the data structure. Efficient data structures are essential to timely check whether advance reservations can be accommodated or to provide alternatives to users with flexible requests [153]; operations generally termed as *admission control*. Moreover, if resource availability exists and a request is accepted, the data structure must be accordingly updated.

In this appendix, we describe a data structure for storing computing resource availability and performing admission control of application jobs and reservations. The data structure relies on a red-black tree; a binary search tree with one additional attribute per node: its colour, which can be either red or black [40]. A red-black tree is approximately balanced as the manner nodes are coloured from the root to a leaf ensures that no path

is more than twice as long as any other. After modifying the red-black tree, rotation and colour change operations guarantee the tree remains approximately balanced.

The nodes of the red-black tree contain the processing elements or computing nodes available at particular times in the future; these times correspond to the start and/or completion times of requests. The red-black tree is used to locate the node that represents the start of a request, also termed as anchor node, whereas the iteration of nodes once the anchor is found is facilitated by a double linked list. Using the linked list, all nodes until the supposed completion of the job are verified to check if there are processors available to admit the job into the system.

The data structure is termed "Availability Profile", or "Profile" for short, as it represents the availability of a specified cluster; the information of processing elements as jobs or reservations start or complete.

A.2 Background and Related Work

A.2.1 Advance Reservations

Emerging deadline-constraint applications require resources to be allocated over a predefined period. Moreover, large scale experiments may demand the co-allocation of resources across several computing sites [20]. This co-allocation of resources and provision of QoS guarantees is achievable in resource management systems by allowing the user to make reservations of resources in advance.

A data structure generally stores the information of resources available until a particular time in the future. The data structure is examined in order to allow a request to be admitted or not. This period over which the availability information is stored depends on the resource allocation policy in use. For example, for a computing resource allocation policy that schedules jobs following a conservative backfilling approach [128], this period may vary if each job's schedule is decided when it arrives at the cluster. For an aggressive backfilling policy [116], this period will probably be shorter as the scheduler maintains the scheduling details about running jobs and the first job in the waiting queue.

A.2.2 Data Structures Using Slotted Time and Non-Slotted Time

Some data structures, generally referred to as slotted-time-based, divide the period over which the availability information is stored in time slots of equal length [26, 173]. In this case, a request, if accepted, is allocated consecutive slots for a period long enough to accommodate the request. In addition, the request is allocated a number of slots whose total length may be equal or greater than the initial time requested by the reservation.

The data structure presented in this appendix uses non-slotted times, allowing for a finer time granularity for the requests accepted as the period allocated to reservations does not depend on the time duration of slots. Moreover, as discussed in the next section, the profile uses the concept of ranges of processing elements available as it needs to ensure that the same processing elements are allocated to a request during the whole execution. With slotted time and short slots, the profile would have this range information replicated at all time slots, and iterating the slots would be time consuming.

A.3 The Availability Profile

The proposed data structure is based on the idea of availability profile described by Mu'alem and Feitelson [128], which utilises a list whose entries contain information about the number of processors available after completion of jobs. The work in this appendix augments this data structure by:

- allowing it to maintain information about reservations;
- using a tree to search for the start of a free time slot suitable for scheduling a job, thus reducing the complexity from $O(n)$ when using a sorted list to $O(log\ n)$ by using the tree; and
- storing information about the ranges of processing elements available at each node, hence enabling various time slot selection policies, such as first-fit, best-fit and worst-fit.

The profile utilises the idea of ranges of Processing Elements (PEs) as it needs to know whether the selected PEs would be available over the entire period requested. This contrasts with data structures that store bandwidth available in a network link over a period, as the availability at a particular time is generally given by one number [26]. The profile needs to ensure that the same PEs are available over the period requested in a reservation as it is difficult to start a job on a set of PEs and migrate the job across PEs several times during its execution. In this work, a PE represents a processor or a core, but the data structure is generic enough to allow the scheduler to work with computing nodes or virtual machines. For example, if a resource with 10 PEs is idle, then a PE range from 0 to 9 is available (*i.e.* [0..9]). The profile can also store availability information about virtual machines, with each PE representing a VM on the infrastructure.

We provide an example of a resource with 13 PEs to illustrate how the data structure looks like (see Figure A.1). The time of the entries was chosen just for the sake of illustrating how the data structure works. The scheduling queue of the computing resource at time 0 is shown in Figure A.1a; the queue contains both *best-effort jobs* and *reservations*. A best-effort job is a job that is started by the scheduler as soon as enough resources are available to handle it whereas a reservation requires resources for a well defined time frame. We detail later the operations for obtaining a time slot to place a job or a reservation. As jobs are inserted, the profile is updated accordingly to reflect the new resource availability. One node is inserted for the completion of the job, containing the time at which the job is expected to complete, the number of PEs available after the job's completion, and the specific ranges of PEs available once the job completes.

Figure A.1b illustrates the resulting red-black tree; shaded circles representing black nodes. Each node represents a particular time and contains the information presented in Figure A.1c. Each line of Figure A.1c represents the information of one node. The dashed lines represent the linked list connecting sibling nodes, which is used to iterate the tree and check whether there are enough PEs to serve a job over a given time interval.

For example, for the profile presented in Figure A.2, in order to perform the admission control of a reservation request whose start time is 220, finish time is 700, and requires 2 PEs, the algorithm:

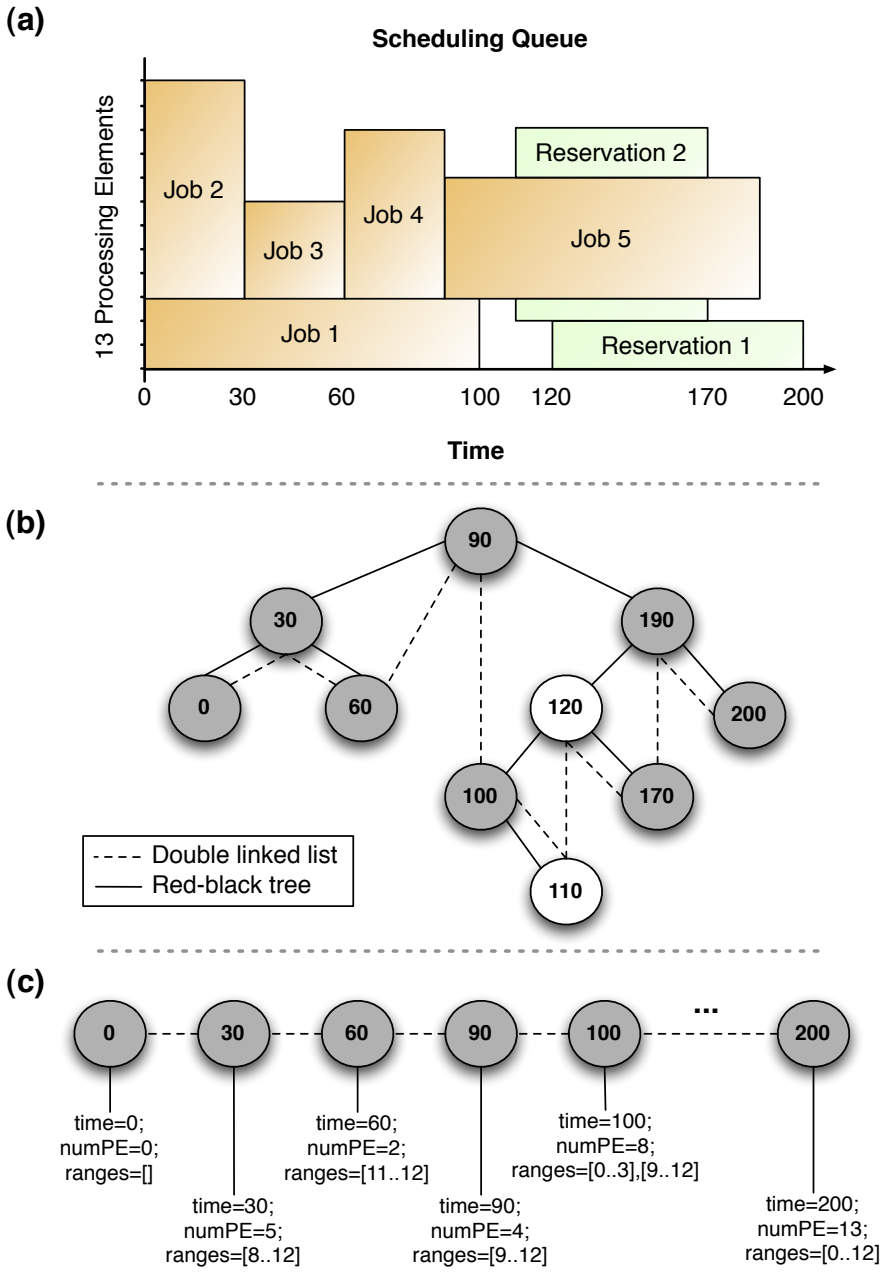


Figure A.1: Pictorial view of the data structure: (a) a scheduling of a resource with 13 processing elements; (b) the representation of availability information as a red-black tree; and (c) details about the information stored by the nodes.

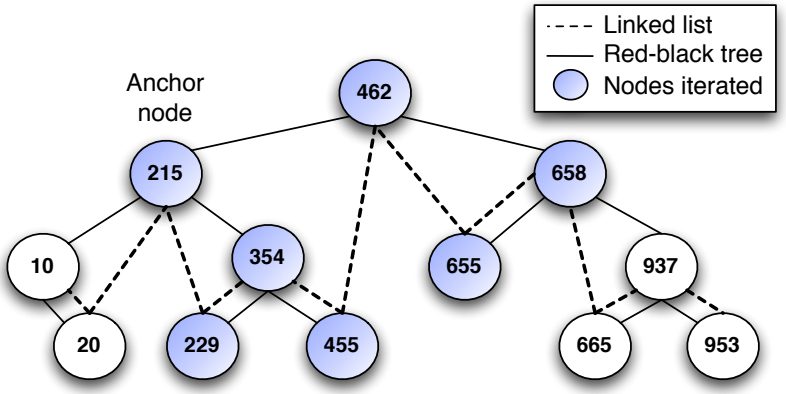


Figure A.2: Iterating the tree using the linked list to perform the admission control of a reservation request to start at time 220 and finish at time 700.

1. Obtains from the reservation the start time, finish time, and number of processors required.

2. Uses the start time of the reservation is to find the node whose time precedes the reservation's start time or is equals to its start time. This node is termed as the *anchor*. If the node does not have enough PEs to serve the reservation, then the request is rejected.

3. Examines the ranges if the anchor has enough PEs to serve the reservation. Then, the linked list is used to iterate the tree and examine all nodes whose times are smaller than the reservation's finish time. For each node, the algorithm computes the intersection of the node's ranges with the ranges of previous nodes examined. If the algorithm examines all the ranges and the resulting intersection has enough PEs to serve the reservation, then the request is accepted.

4. Stops its execution and requests the rejected if at any time an intersection is computed, the intersection does not have enough processing elements.

In summary, the tree is used to obtain the anchor for the reservation request, in this case the node with time 215, and the list is used afterwards to iterate all nodes whose times are smaller than the request's finish time. The intersection of ranges is computed during this iteration. Figure A.3a illustrates the relevant part of the profile represented as lists of ranges of PEs over the scheduling queue, whereas Figure A.3b shows the actual scheduling queue with the corresponding reservations. The request can be accepted because the intersection of PE ranges has more PEs than what the reservation requires.

A.3.1 Operations

The implementation of the availability profile contains several operations namely to:

- Check whether a reservation with strict start and finish times can be accommodated.

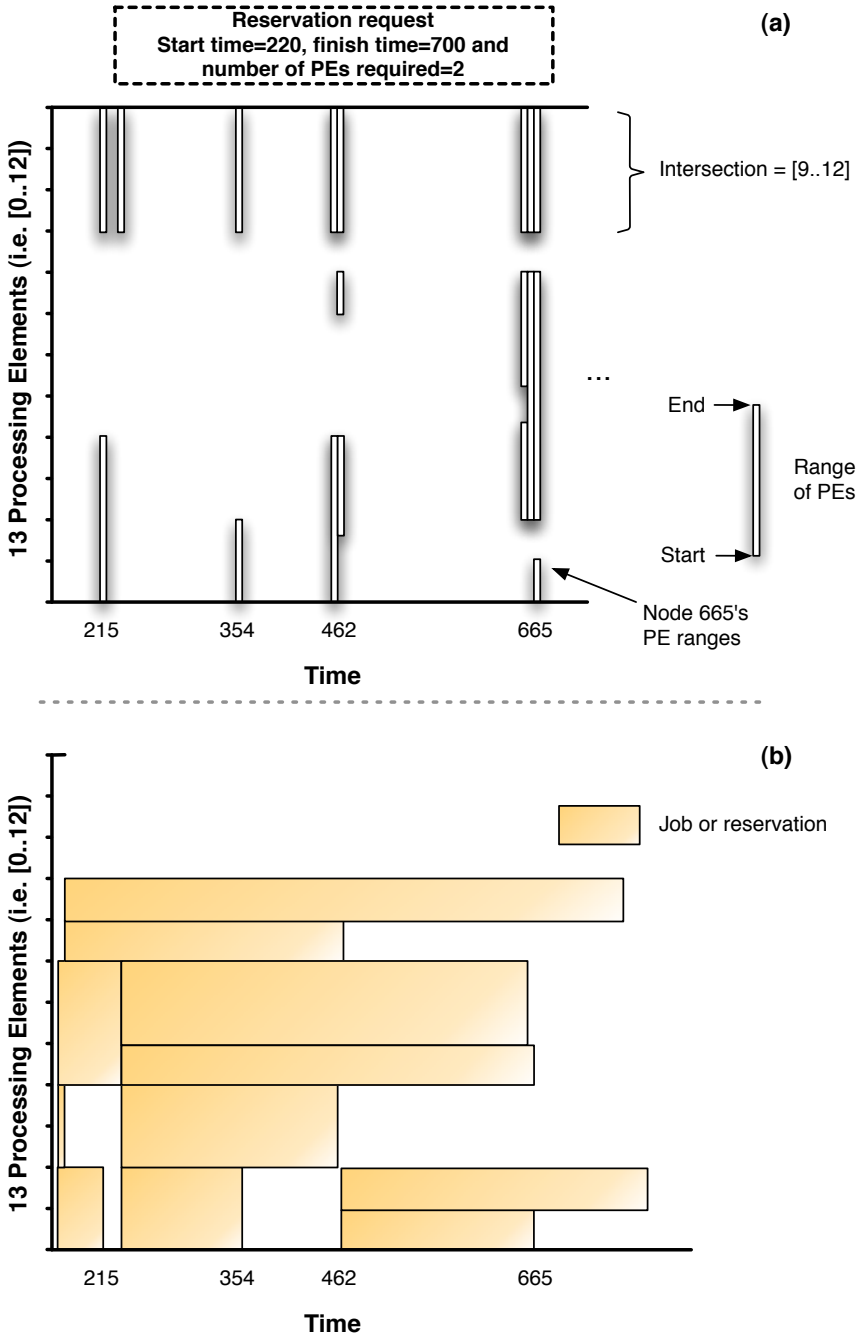


Figure A.3: A representation of part of a scheduling queue as (a) ranges of free PEs and; (b) corresponding reservations. In order to accommodate a reservation request, the intersection of ranges available needs to have enough processing elements.

- Find a time slot over which a job or reservation with flexible start and finish times can execute.
- Obtain the availability information (*i.e.* free time slots) in the profile.
- Get the scheduling options for a job or reservation, which are important for schedulers based on strategies such as best-fit and worst-fit.
- Add free time slots back to the profile in case jobs are cancelled or paused.
- Reconstruct an availability profile from a list of free time slots.
- Allocate time slots to jobs or reservations.

In this section, we describe two particular operations namely the option to check the availability in the profile (*i.e.* to check whether a request can be served) and to update the profile by allocating the PE ranges selected for the request.

Check Availability

There are two important types of requests, namely those that require resources at a well established time frame, which are here termed reservations, and those that require resources when the resource provider is able to offer them, termed best-effort jobs. The first request type represents applications that have stringent QoS requirements or require the co-allocation of resources. Co-allocation of resources is eased by allowing a user to reserve resources at the computing sites she wants to use. The second type of request reflects the scenario of most applications where users are not very strict about the period over which their jobs will run.

We have described beforehand the process of checking whether a reservation request can be accommodated. In summary, first the node whose time is equals the start time of the reservation or precedes the reservation's start time is found using the red-black tree. This node is the anchor. Secondly, the linked list is used to check all the nodes in the interval between the anchor node and the last node before the finish time of the reservation. In this case, the complexity for checking whether a reservation can be admitted into the system or not is $O(log\ n + m)$ or $O(m)$, where $log\ n$ is the cost of finding the anchor node in the red-black tree and m is the number of nodes of the sub-list between the anchor node and the last node before the finish time of the reservation request.

If we need to schedule a best-effort job, which can be served at any time, then the algorithm that needs to schedule a best-effort job can start iterating the tree using the current time as the start time. However, differently from the admission control of a reservation request, in order to find a time slot in which a job fits, the algorithm starts with a potential anchor node. That is, a node with enough PEs to serve the job. The intersection of the potential anchor's PE ranges with the following nodes' ranges until before the expected completion of the job needs to have enough PEs to accommodate the job. The pseudo-code for this procedure is depicted in Algorithm A.1.

The profile also provides operations that allow the scheduler to obtain the *free time slots*. A free time slot contains information about the resources available over a given time frame. In this way, a time slot has a start time, a finish time, and the ranges of

Algorithm A.1: Pseudo-code to find a time slot to accommodate a job.

input : the job's runtime (duration) and number of PEs (reqPE)
output: a profile entry with the job's start time and ranges available

```
1  ctime ← the current time
2  iter ← iterator for the profile starting at the node that precedes ctime
3  intersec ← null
4  pstime ← ctime // stores the job's potential start time
5  pftime ← −1 // stores the job's potential finish time
6  anchor ← null
7  while iter has a next element do
8      anchor ← the next element of iter
9      if anchor.numPE < reqPE then
10         continue
11     else
12         // a potential anchor has been found
13         pstime ← anchor.time // the potential start time is the anchor's time
14         pftime ← pstime + duration // the potential finish time
15         intersec ← anchor.ranges // stores the intersections of PE ranges
16         ita ← iterator for the profile starting at the node after pstime
17         while ita has a next element do
18             nxnode ← the next element of ita
19             // it does not need to check the nodes beyond the potential finish time
20             if nxnode.time ⩾ pftime then
21                 break
22             else
23                 if nxnode.numPE < reqPE then
24                     // there are not enough processing elements available
25                     intersec ← null
26                     break
27                 intersect ← intersect ∩ nxnode.ranges
28                 if intersec.numPE < reqPE then
29                     // there are not enough processing elements available
30                     break
31         if intersec.numPE ⩾ reqPE then
32             // it found a slot with enough processing elements
33             break
34 entry ← new entry with time = pstime
35 entry.ranges ← intersec
36 return entry
```

PEs available during this period. The availability profile has two operations to obtain the free time slots. In the first operation, the resulting free time slots do not overlap with one another. This approach is similar to that used by Singh *et al.* [168] in their extended conservative backfilling policy. The complexity of this operation is in the worst case scenario $O(log\ n + m^2)$ or $O(m^2)$, where $log\ n$ is the cost of finding the node that represents the start time of the given period (*i.e.* the anchor) and m is the number of nodes in the list between the anchor node and the last node before the finish time of the time frame. In real scenarios, this operation is not invoked often, as the operations required to check whether a request can be admitted do not rely on obtaining a list of free time slots

from the availability profile.

The second operation returns the scheduling options [104]. In this case, the time slots overlap. The free time slots returned by this operation are termed as *scheduling options* because they represent the places in the queue where a given job or reservation can be placed. This operation is useful if the scheduler is required to perform a more complex selection of PE ranges for a best-effort job or a reservation request. For example, in some systems the users are allowed to extend previous reservations or resource leases [102]. The scheduler may be required to select a free time slot for a reservation that can accommodate a potential extension or renewal of the resulting resource lease.

Update the Profile

Once a reservation request has been accepted or the time slot for a job has been found, the availability profile is updated to reflect the changes. In summary, the profile needs to:

- Update the node that has been selected as the request's anchor or insert a new anchor in case of a reservation whose start time does not coincide with an existing node of the tree.
- Update all entries from the anchor until before the request's completion time, removing the PE ranges selected.
- Insert a new node marking the completion time of the request and containing the PE ranges that will be available after the request is complete.

In order to minimise the number of nodes in the tree, requests with the same start time or completion time share nodes in the profile. The complexity of the update operation is $O(log\ n + m)$ or $O(m)$ as it consists in inserting one element in the tree (*i.e.* $log\ n$) and updating the m nodes until before the completion of the job, using the list to iterate them.

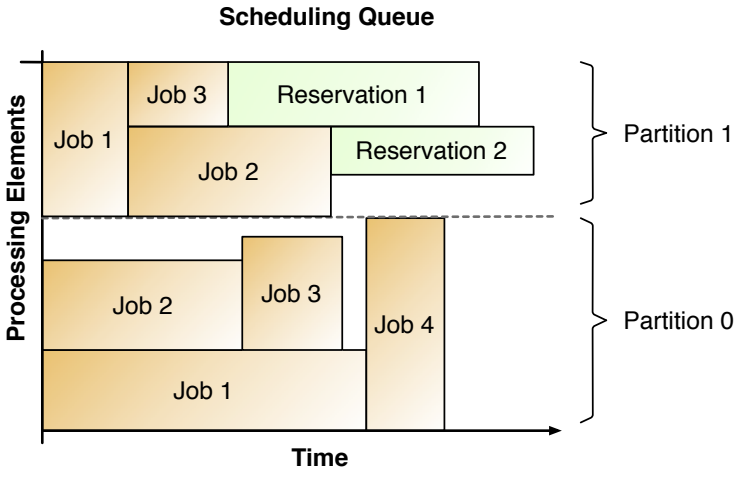


Figure A.4: Example of a profile with two resource partitions.

A.3.2 Multiple Resource Partitions

We have also introduced an availability profile to control the allocation of ranges of PEs to different resource partitions [114]. This availability profile is termed as *partitioned profile*. The idea of multiple resource partitions is depicted in Figure A.4. The nodes in the profile store the ranges available at more than one resource partition. A user can check the availability of a given partition as well as update that particular partition. As the profile extends the normal profile, it is possible to create allocation polices based on the partitioned profile, that allow a partition to borrow resources from another. In order to enable that, the user just needs to use the operations offered by the normal profile.

A.3.3 Details of the Implementation

The data structure has been implemented in Java. We have modified the *TreeMap* implementation provided by Java 1.5 in order to maintain a double linked list among sibling nodes. The modified version is called *LinkedTreeMap*. Figure A.5 presents the main classes that compose the availability profiles.

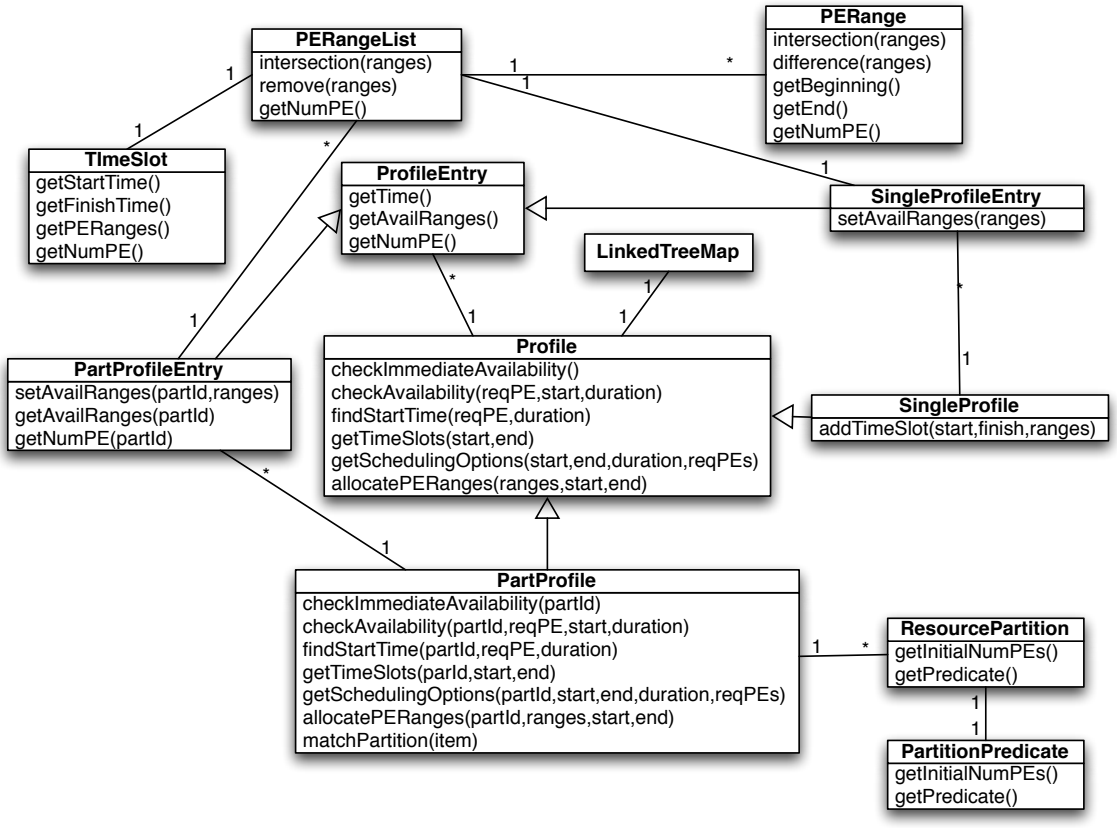


Figure A.5: Diagram containing the relevant classes of the availability profiles.

Profile and *ProfileEntry* are abstract classes. *Profile* has methods that are common to a single partition profile (*i.e. SingleProfile*) and a multiple resource partition profile (*i.e. PartProfile*). Similarly, *PartProfileEntry* extends *ProfileEntry* and stores information about ranges of PEs available at the multiple resource partitions created in the profile.

The availability profile with multiple resource partitions extends the normal profile and provides methods to check the availability and update the profile per partition as well as all partitions at once. The nodes in a partitioned profile contain information about the ranges of PEs available at all partitions at a specific time.

A.4 Using the Profile

This section provides an example on how to utilise the availability profile. We show an example of a scheduling policy based on conservative backfilling [128] for a Grid simulator. It should be straightforward to implement various policies relying on availability profiles and the idea of ranges of PEs. We also demonstrate how to obtain the scheduling options for a job so the user can perform the selection of PEs using approaches such as best-fit and worst-fit to minimise the scheduling queue's fragmentation and improve resource utilisation [104].

Algorithm A.2: Sample pseudo-code of a conservative backfilling policy.

```
1  procedure jobSubmitted(Job j)
2  begin
3      success ← startJob(j)
4      if success = false then
5          success ← enqueueJob(j)
6  end
7  procedure startJob(Job j)
8  begin
9      ctime ← gets the current time
10     // obtain a node that will be used as anchor for the job
11     anchor ← profile.check(j.numPE, ctime, j.runtime)
12     if anchor does not have enough PEs then
13         return false
14     else
15         selected ← select ranges from anchor
16         profile.allocate(selected, ctime, j.runtime)
17         j.ranges ← selected
18         j.starttime ← ctime
19         return true
20 end
21 procedure enqueueJob(Job j)
22 begin
23     // search for a node that will be used as anchor for the job
24     anchor ← profile.check(j.numPE, j.runtime)
25     selected ← select ranges from anchor
26     profile.allocate(selected, anchor.time, j.runtime)
27     j.ranges ← selected
28     j.starttime ← anchor.time
29 end
```

Obtaining the scheduling options is useful to generate alternatives for advance reservations that cannot be accommodated but have flexibility regarding the time frame over

which they require the resources or the number of resources required [153, 154, 196]. A user can obtain and utilise the free time slots or provide them to another entity, which can further carry out provisioning decisions [45, 168].

Algorithm A.2 shows sample operations required by a conservative backfilling policy. In order to simplify the operations and the update of availability information, the schedule of a job is generally determined at its arrival at the resource [128]. The operation *jobSubmitted(Job j)* represents the submission of a job to the scheduler. The scheduler initially tries to start the job immediately by calling *startJob(Job j)*. For simplicity, the algorithm illustrates only the selection of ranges of PEs and the anchor point for a job, and does not focus on the operations to start the execution of the job. Note that *startJob(Job j)* performs the same operations require to admit a reservation. In fact, to start a job immediately upon its arrival at the resource, a reservation needs to be made starting at the present time. If a job cannot start upon its arrival to the resource, then the scheduler needs to find the anchor point containing the time at which the job can start. This procedure is depicted by *enqueueJob(Job j)*. Once the anchor node is obtained, the scheduler selects the ranges of PEs required to serve the job and updates the profile accordingly.

A.5 Summary

This appendix presented a data structure to facilitate the scheduling of best-effort jobs and reservation requests, and resource provisioning in traditional resource management systems. We provided details about the data structure, which relies on a red-black tree to find the potential start time of reservations and a double linked list to iterate the tree's nodes. We provided an example that demonstrates how the availability profile can be utilised to create scheduling policies and generate alternative offers for advance reservation requests.

REFERENCES

[1] Adabala, S., Chadha, V., Chawla, P., Figueiredo, R., Fortes, J., Krsul, I., Matsunaga, A., Tsugawa, M., Zhang, J., Zhao, M., Zhu, L., and Zhu, X. (2005). From virtualized resources to virtual computing Grids: the In-VIGO system. *Future Generation Computer Systems*, 21(6):896–909.

[2] Alfieri, R., Cecchini, R., Ciaschini, V., dell'Agnello, L., Àkos Frohner, Lörentey, K., and Spataro, F. (2005). From gridmap-file to VOMS: Managing authorization in a Grid environment. *Future Generation Computer Systems*, 21(4):549–558.

[3] Almond, J. and Snelling, D. (1999). UNICORE: Uniform access to supercomputing as an element of electronic commerce. *Future Generation Computer Systems*, 15(5-6):539–548.

[4] Amazon Inc. (2008). Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2.

[5] Anand, S., Yoginath, S. B., von Laszewski, G., Alunkal, B., and Sun, X.-H. (2003). Flow-based multistage co-allocation service. In *International Conference on Communications in Computing (CIC 2003)*, pages 24–30, Las Vegas, USA. CSREA Press.

[6] Andrade, N., Brasileiro, F., Cirne, W., and Mowbray, M. (2007). Automatic Grid assembly by promoting collaboration in peer-to-peer Grids. *Journal of Parallel and Distributed Computing*, 67(8):957–966.

[7] Andrade, N., Cirne, W., Brasileiro, F., and Roisenberg, P. (2003). OurGrid: An approach to easily assemble Grids with equitable resource sharing. In *9th Workshop on Job Scheduling Strategies for Parallel Processing*, volume 2862, pages 61–86, Berlin/Heidelberg. Springer.

[8] Andrade, N., Costa, L., Germoglio, G., and Cirne, W. (2005). Peer-to-peer Grid computing with the OurGrid community. In *23rd Brazilian Symposium on Computer Networks, IV Special Tools Session*. Brazilian Computer Society.

[9] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., and Zaharia, M. (2009). Above the clouds: A Berkeley view of Cloud computing. Technical report UCB/EECS-2009-28, Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, USA.

[10] AuYoung, A., Chun, B., Ng, C., Parkes, D. C., Vahdat, A., and Snoeren, A. (2007). Practical market-based resource allocation. Technical report CS2007-0901, CSE, University of California San Diego.

[11] AuYoung, A., Grit, L., Wiener, J., and Wilkes, J. (2006). Service contracts and aggregate utility functions. In *15th IEEE International Symposium on High Performance Distributed Computing (HPDC 2006)*, pages 119–131, Paris, France.

[12] Baake, P. and Wichmann, T. (1999). On the economics of Internet peering. *NETNOMICS*, 1(1):89–105.

[13] Badasyan, N. and Chakrabarti, S. (2005). Private peering, transit and traffic diversion. *NETNOMICS*, 7(2):115–124.

[14] Balazinska, M., Balakrishnan, H., and Stonebraker, M. (2004). Contract-based load management in federated distributed systems. In *1st Symposium on Networked Systems Design and Implementation (NSDI)*, pages 197–210, San Francisco, USA. USENIX Association.

[15] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. (2003). Xen and the art of virtualization. In *19th ACM Symposium on Operating Systems Principles (SOSP '03)*, pages 164–177, New York, USA. ACM Press.

[16] Barmouta, A. and Buyya, R. (2003). GridBank: a Grid Accounting Services Architecture (GASA) for distributed systems sharing and integration. In *Workshop on Internet Computing and E-Commerce (held with IPDPS 2003)*, Nice, France.

[17] Bartley, W. W., Klein, P. G., and Caldwell, B., editors (1989). *The Collected Works of Friedrich A. Von Hayek*. University of Chicago Press.

[18] Berman, F. and Wolski, R. (1997). The AppLeS project: A status report. In *8th NEC Research Symposium*, Berlin, Germany.

[19] Berners-Lee, T. and Fischetti, M. (1999). *Weaving the Web: the past, present and future of the World Wide Web by its inventor*. Orion Business, London.

[20] Boghosian, B., Coveney, P., Dong, S., Finn, L., Jha, S., Karniadakis, G. E., and Karonis, N. T. (2006). Nektar, SPICE and Vortonics: Using federated Grids for large scale scientific applications. In *IEEE Workshop on Challenges of Large Applications in Distributed Environments (CLADE)*, Paris, France. IEEE Computing Society.

[21] Bolze, R., Cappello, F., Caron, E., Daydé, M., Desprez, F., Jeannot, E., Jégou, Y., Lantéri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Primet, P., Quetier, B., Richard, O., Talbi, E.-G., and Iréa, T. (2006). Grid'5000: a large scale and highly reconfigurable experimental Grid testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494.

[22] Brugnoli, M., Willmott, S., Heymann, E., Hurley, P., and Senar, M. A. (2007). Applying internet random early detection strategies to scheduling in Grid environments. In *14th International Conference on High Performance Computing (HiPC 2007)*, volume 4873 of *LNCS*, pages 587–598, Berlin/Heidelberg, Germany. Springer.

[23] Brune, M., Gehring, J., Keller, A., and Reinefeld, A. (1999). Managing clusters of geographically distributed high-performance computers. *Concurrency: Practice and Experience*, 11(15):887–911.

[24] Brunelle, J., Hurst, P., Huth, J., Kang, L., Ng, C., Parkes, D., Seltzer, M., Shank, J., and Youssef, S. (2006). EGG: An extensible and economics-inspired open Grid computing platforms. In *3rd International Workshop on Grid Economics and Business Models (GECON 2006)*, pages 140–150, Singapore.

[25] Bulhões, P. T., Byun, C., Castrapel, R., and Hassaine, O. (2004). N1 Grid engine 6 features and capabilities. White paper, Sun Microsystems, Phoenix, USA.

[26] Burchard, L.-O. (2005). Analysis of data structures for admission control of advance reservation requests. *IEEE Transactions on Knowledge and Data Engineering*, 17(3):413–424.

[27] Butt, A. R., Zhang, R., and Hu, Y. C. (2003). A self-organizing flock of condors. In *2003 ACM/IEEE Conference on Supercomputing (SC 2003)*, page 42, Washington, DC, USA. IEEE Computer Society.

[28] Buyya, R. (2002). *Economic-based Distributed Resource Management and Scheduling for Grid Computing*. PhD thesis, Monash University, Melbourne, Australia.

[29] Buyya, R., Abramson, D., and Giddy, J. (2000a). An economy driven resource management architecture for global computational power Grids. In *7th International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000)*, Las Vegas, USA. CSREA Press.

[30] Buyya, R., Abramson, D., and Giddy, J. (2000b). Nimrod/G: An architecture for a resource management and scheduling system in a global computational Grid. In *4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000)*, pages 283–289, Beijing, China.

[31] Buyya, R. and Murshed, M. (2002). GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. *Concurrency and Computation: Practice and Experience (CCPE)*, 14(13–15):1175–1220.

[32] 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 *Fifth IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05)*, volume 2, pages 776–783, Washington, DC, USA. IEEE Computer Society.

[33] Caromel, D., Delbé, C., di Costanzo, A., and Leyton, M. (2006). ProActive: an integrated platform for programming and running applications on Grids and P2P systems. *Computational Methods in Science and Technology*, 12(1):69–77.

[34] Caromel, D., di Costanzo, A., and Mathieu, C. (2007). Peer-to-peer for computational Grids: Mixing clusters and desktop machines. *Parallel Computing*, 33(4–5):275–288.

[35] Casavant, T. L. and Kuhl, J. G. (1988). A taxonomy of scheduling in general-purpose distributed computing systems. *IEEE Transactions on Software Engineering*, 14(2):141–154.

[36] Catlett, C., Beckman, P., Skow, D., and Foster, I. (2006). Creating and operating national-scale cyberinfrastructure services. *Cyberinfrastructure Technology Watch Quarterly*, 2(2):2–10.

[37] Chapin, S. J., Katramatos, D., Karpovich, J. F., and Grimshaw, A. S. (1999). The Legion resource management system. In *Job Scheduling Strategies for Parallel Processing (IPPS/SPDP '99/JSSPP '99)*, pages 162–178, London, UK. Springer-Verlag.

[38] Chase, J. S., Irwin, D. E., Grit, L. E., Moore, J. D., and Sprenkle, S. E. (2003). Dynamic virtual clusters in a Grid site manager. In *12th IEEE International Symposium on High Performance Distributed Computing (HPDC 2003)*, page 90, Washington, DC, USA. IEEE Computer Society.

[39] Clark, D. D., Wroclawski, J., Sollins, K. R., and Braden, R. (2005). Tussle in cyberspace: Defining tomorrow's Internet. *IEEE/ACM Transactions on Networking*, 13(3):462–475.

[40] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction to Algorithms*. MIT Press / McGraw-Hill, Cambridge, Massachusetts, 2nd edition.

[41] Czajkowski, K., Foster, I., and Kesselman, C. (1999). Resource co-allocation in computational Grids. In *8th International Symposium on High Performance Distributed Computing (HPDC'99)*, pages 219–228, Redondo Beach, USA.

[42] Dang, V. D. (2004). *Coalition Formation and Operation in Virtual Organisations*. PhD thesis, Faculty of Engineering, Science and Mathematics, School of Electronics and Computer Science, University of Southampton, Southampton, UK.

[43] DAS (2006). The Distributed ASCI Supercomputer 2 (DAS-2). Dutch University Backbone.

[44] Dash, R. K., Jennings, N. R., and Parkes, D. C. (2003). Computational-mechanism design: A call to arms. *IEEE Intelligent Systems*, 18(6):40–47.

[45] de Assunção, M. D. and Buyya, R. (2008). Performance analysis of multiple site resource provisioning: Effects of the precision of availability information. In *International Conference on High Performance Computing (HiPC 2008)*, volume 5374 of *LNCS*, pages 157–168, Berlin/Heidelberg. Springer.

[46] de Assunção, M. D., Buyya, R., and Venugopal, S. (2008). InterGrid: A case for internetworking islands of Grids. *Concurrency and Computation: Practice and Experience (CCPE)*, 20(8):997–1024.

[47] de Assunção, M. D., Nadiminti, K., Venugopal, S., Ma, T., and Buyya, R. (2005). An integration of global and enterprise Grid computing: Gridbus Broker and Xgrid perspective. In *International Conference on Grid and Cooperative Computing (GCC 2005)*, volume 3795, Berlin/Heidelberg. Springer.

[48] Deelman, E., Singh, G., Livny, M., Berriman, B., and Good, J. (2008). The cost of doing science on the Cloud: The Montage example. In *2008 ACM/IEEE Conference on Supercomputing (SC 2008)*, pages 1–12, Piscataway, USA. IEEE.

[49] Dimitrakos, T., Golby, D., and Kearley, P. (2004). Towards a trust and contract management framework for dynamic virtual organisations. In *eChallenges*, Vienna, Austria.

[50] Dixon, C., Bragin, T., Krishnamurthy, A., and Anderson, T. (2006). Tit-for-tat distributed resource allocation. In *ACM SIGCOMM 2006 Conference*, Pisa, Italy. ACM.

[51] Dumitrescu, C. and Foster, I. (2004). Usage policy-based CPU sharing in virtual organizations. In *5th IEEE/ACM International Workshop on Grid Computing (Grid 2004)*, pages 53–60, Washington, DC, USA. IEEE Computer Society.

[52] Dumitrescu, C. and Foster, I. (2005). GRUBER: A Grid resource usage SLA broker. In Cunha, J. C. and Medeiros, P. D., editors, *Euro-Par 2005*, volume 3648 of *LNCS*, pages 465–474, Berlin/Heidelberg. 3648.

[53] Dumitrescu, C., Raicu, I., and Foster, I. (2005a). DI-GRUBER: A distributed approach to Grid resource brokering. In *2005 ACM/IEEE Conference on Supercomputing (SC 2005)*, page 38, Washington, DC, USA. IEEE Computer Society.

[54] Dumitrescu, C., Wilde, M., and Foster, I. (2005b). A model for usage policy-based resource allocation in Grids. In *6th IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 191–200, Washington, DC, USA. IEEE Computer Society.

[55] Dunning, T. and Nandkumar, R. (2006). International cyberinfrastructure: Activities around the globe. *Cyberinfrastructure Technology Watch Quarterly*, 2(1).

[56] Eerola, P., Kónya, B., Smirnova, O., Ekelöf, T., Ellert, M., Hansen, J. R., Nielsen, J. L., Wäänänen, A., Konstantinov, A., and Ould-Saada, F. (2003). Building a production Grid in Scandanavia. *IEEE Internet Computing*, 7(4):27–35.

[57] EGEE (2005). gLite – lightweight middleware for Grid computing. http://glite.web.cern.ch/glite.

[58] Ellert, M., Grønager, M., Konstantinov, A., Kónya, B., Lindemann, J., Livenson, I., Nielsen, J. L., Niinimäki, M., Smirnova, O., and Wäänänen, A. (2007). Advanced Resource Connector middleware for lightweight computational Grids. *Future Generation Computer Systems*, 23(2):219–240.

[59] Elmroth, E. and Gardfjäll, P. (2005). Design and evaluation of a decentralized system for Grid-wide fairshare scheduling. In *1st IEEE International Conference on e-Science and Grid Computing*, pages 221–229, Melbourne, Australia. IEEE Computer Society.

[60] Elmroth, E. and Tordsson, J. (2005). An interoperable, standards-based Grid resource broker and job submission service. In *1st International Conference on e-Science and Grid Computing (e-Science 2005)*, pages 212–220, Melbourne, Australia.

[61] Elmroth, E. and Tordsson, J. (2008). Grid resource brokering algorithms enabling advance reservations and resource selection based on performance predictions. *Future Generation Computer Systems*, 24(6):585–593.

[62] 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 (CCPE)*. (To appear).

[63] Emeneker, W., Jackson, D., Butikofer, J., and Stanzione, D. (2006). Dynamic virtual clustering with Xen and Moab. In *Frontiers of High Performance Computing and Networking with ISPA 2006*, volume 4331 of *LNCS*, pages 440–451, Berlin/Heidelberg. Springer.

[64] Emmott, S. and Rison, S. (2006). Towards 2020 science report. Technical report, Microsoft Research, Cambridge, USA.

[65] Enabling Grids for E-sciencE (2005). EGEE project. http://www.eu-egee.org/.

[66] England, D. and Weissman, J. B. (2004). Costs and benefits of load sharing in the computational Grid. In *10th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '04)*, volume 3277 of *LNCS*, pages 160–175, New York, USA. Springer Berling Heidelberg.

[67] Epema, D. H. J., Livny, M., van Dantzig, R., Evers, X., and Pruyne, J. (1996). A worldwide flock of condors: Load sharing among workstation clusters. *Future Generation Computer Systems*, 12(1):53–65.

[68] Erdil, D. C. and Lewis, M. J. (2007). Grid resource scheduling with gossiping protocols. In *Seventh IEEE International Conference on Peer-to-Peer Computing (P2P 2007)*, pages 193–202, Washington, DC, USA. IEEE Computer Society.

[69] Ernemann, C., Hamscher, V., and Yahyapour, R. (2002). Economic scheduling in Grid computing. In *8th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 2002)*, pages 128–152, London, UK. Springer-Verlag.

[70] Eymann, T., Ardaiz, O., Catalano, M., Chacin, P., Chao, I., Freitag, F., Gallegati, M., Giulioni, G., Joita, L., Navarro, L., Neumann, D. G., Rana, O., Reinicke, M., Schiaffino, R. C., Schnizler, B., Streitberger, W., Veit, D., and Zini, F. (2005). Catallaxy-based Grid markets. *International Journal on Multiagent and Grid Systems, Special Issue on Smart Grid Technologies & Market Models*, 1(4):297–307.

[71] Farooq, U., Majumdar, S., and Parsons, E. W. (2005). Impact of laxity on scheduling with advance reservations in Grids. In *13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 319–322.

[72] Feigenbaum, J., Karger, D. R., Mirrokni, V. S., and Sami, R. (2005). Subjective-cost policy routing. In Deng, X. and Ye, Y., editors, *First International Workshop on Internet and Network Economics (WINE 2005)*, volume 3828 of *LNCS*, pages 174–183, Hong Kong. Springer.

[73] Feigenbaum, J., Sami, R., and Shenker, S. (2004). Mechanism design for policy routing. In *PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 11–20, New York, USA. ACM.

[74] Feitelson, D. G., Rudolph, L., Schwiegelshohn, U., Sevcik, K. C., and Wong, P. (1997). Theory and practice in parallel job scheduling. In *Job Scheduling Strategies for Parallel Processing (IPPS '97)*, pages 1–34, London, UK. Springer-Verlag.

[75] Field, J. and Varela, C. A. (2005). Transactors: A programming model for maintaining globally consistent distributed state in unreliable environments. In *ACM Conference on Principles of Programming Languages (POPL 2005)*, pages 195–208, Long Beach, USA.

[76] Fontán, J., Vázquez, T., Gonzalez, L., Montero, R. S., and Llorente, I. M. (2008). OpenNEbula: The open source virtual machine manager for cluster computing. In *Open Source Grid and Cluster Software Conference – Book of Abstracts*, San Francisco, USA.

[77] Foster, I., Freeman, T., Keahey, K., Scheftner, D., Sotomayor, B., and Zhang, X. (2006). Virtual clusters for Grid communities. In *6th IEEE International Symposium on Cluster Computing and the Grid (CCGRID 2006)*, pages 513–520, Washington, DC, USA. IEEE Computer Society.

[78] Foster, I., Gieraltowski, J., Gose, S., Maltsev, N., May, E., Rodriguez, A., Sulakhe, D., Vaniachine, A., Shank, J., Youssef, S., Adams, D., Baker, R., Deng, W., Smith, J., Yu, D., Legrand, I., Singh, S., Steenberg, C., Xia, Y., Afaq, A., Berman, E., Annis, J., Bauerdick, L., Ernst, M., Fisk, I., Giacchetti, L., Graham, G., Heavey, A., Kaiser, J., Kuropatkin, N., Pordes, R., Sekhri, V., Weigand, J., Wu, Y., Baker, K., Sorrillo, L., Huth, J., Allen, M., Grundhoefer, L., Hicks, J., Luehring, F., Peck, S., Quick, R., Simms, S., Fekete, G., VandenBerg, J., Cho, K., Kwon, K., Son, D., Park, H., Canon, S., Jackson, K., Konerding, D., Lee, J., Olson, D., Sakrejda, I., Tierney, B., Green, M., Miller, R., Letts, J., Martin, T., Bury, D., Dumitrescu, C., Engh, D., Gardner, R., Mambelli, M., Smirnov, Y., Voeckler, J., Wilde, M., Zhao, Y., Zhao, X., Avery, P., Cavanaugh, R., Kim, B., Prescott, C., Rodriguez, J., Zahn, A., McKee, S., Jordan, C., Prewett, J., Thomas, T., Severini, H., Clifford, B., Deelman, E., Flon, L., Kesselman, C., Mehta, G., Olomu, N., Vahi, K., De, K., McGuigan, P., Sosebee, M., Bradley, D., Couvares, P., Smet, A. D., Kireyev, C., Paulson, E., Roy, A., Koranda, S., Moe, B., Brown, B., and Sheldon, P. (2004). The Grid2003 production Grid: Principles and practice. In *13th IEEE International Symposium on High Performance Distributed Computing (HPDC 2004)*, pages 236–245.

[79] Foster, I. and Kesselman, C. (1997). Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128.

[80] Foster, I. and Kesselman, C. (1999). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[81] 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 *7th International Workshop on Quality of Service (IWQoS'99)*, pages 27–36, London, UK.

[82] Foster, I., Kesselman, C., and Tuecke, S. (2001). The anatomy of the Grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3):200–222.

[83] Freeman, P. A., Crawford, D. L., Kim, S., and noz, J. L. M. (2005). Cyberinfrastructure for science and engineering: promises and challenges. *Proceedings of the IEEE*, 93(3):682–691.

[84] Frey, J., Tannenbaum, T., Livny, M., Foster, I. T., and Tuecke, S. (2001). Condor-G: A computation management agent for multi-institutional Grids. In *10th IEEE International Symposium on High Performance Distributed Computing (HPDC 2001)*, pages 55–63, San Francisco, USA. IEEE Computer Society.

[85] Friedman, T. (2008). OneLab experiences in development and federation. In *3rd International Conference on Future Internet and Technologies (CFI08)*, Seoul, Korea.

[86] Fu, Y., Chase, J., Chun, B., Schwab, S., and Vahdat, A. (2003). SHARP: An architecture for secure resource peering. In *19th ACM Symposium on Operating Systems Principles (SOSP 2003)*, pages 133–148, New York, USA. ACM Press.

[87] Garbacki, P. and Naik, V. K. (2007). Efficient resource virtualization and sharing strategies for heterogeneous Grid environments. In *10th IFIP/IEEE International Symposium on Integrated Network Management (IM 2007)*, pages 40–49, Munich, Germany.

[88] Graupner, S., Kotov, V., Andrzejak, A., and Trinks, H. (2002). Control architecture for service Grids in a federation of utility data centers. Technical report HPL-2002-235, HP Laboratories Palo Alto, Palo Alto, USA.

[89] Grid Interoperability Now Community Group (GIN-CG) (2006). http://forge.ogf.org/sf/projects/gin.

[90] Grimme, C., Lepping, J., and Papaspyrou, A. (2008). Prospects of collaboration between compute providers by means of job interchange. In *Job Scheduling Strategies for Parallel Processing*, volume 4942 of *LNCS*, pages 132–151, Berlin/Heidelberg, Germany. Springer.

[91] Grit, L. E. (2007). *Extensible Resource Management for Networked Virtual Computing*. PhD thesis, Department of Computer Science, Duke University, Durham, NC, USA. Adviser: Jeffrey S. Chase.

[92] 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.

[93] Hanke, J. E. and Reitsch, A. G. (1995). *Business Forecasting*. Prentice-Hall, Inc., Englewood Cliffs, USA, 5th edition.

[94] Hawkinson, J. and Bates, T. (1996). Guidelines for creation, selection, and registration of an autonomous system (as). IETF, RFC 1930.

[95] Hazlewood, V., Bean, R., and Yoshimoto, K. (2001). SNUPI: A Grid accounting and performance system employing portal services and RDBMS back-end. In *Linux Clusters: the HPC Revolution*.

[96] Hey, T. and Trefethen, A. E. (2002). The UK e-science core programme and the Grid. *Future Generation Computer Systems*, 18(8):1017–1031.

[97] Huang, R., Casanova, H., and Chien, A. A. (2006). Using virtual Grids to simplify application scheduling. In *20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*, Rhodes Island, Greece. IEEE.

[98] Huedo, E., Montero, R. S., and Llorente, I. M. (2004). A framework for adaptive execution in Grids. *Software Practice and Experience*, 34(7):631–651.

[99] Huedo, E., Montero, R. S., and Llorente, I. M. (2009). A recursive architecture for hierarchical Grid resource management. *Future Generation Computer Systems*, 25(4):401–405.

[100] Huston, G. (1999). Interconnection, peering and settlements - part I. *The Internet Protocol Journal*, 2(1):2–16.

[101] Iosup, A., Epema, D. H. J., Tannenbaum, T., Farrellee, M., and Livny, M. (2007). Inter-operating Grids through delegated matchmaking. In *2007 ACM/IEEE Conference on Supercomputing (SC 2007)*, pages 1–12, New York, USA. ACM Press.

[102] Irwin, D., Chase, J., Grit, L., Yumerefendi, A., Becker, D., and Yocum, K. G. (2006). Sharing networked resources with brokered leases. In *USENIX Annual Technical Conference*, pages 199–212, Berkeley, USA. USENIX Association.

[103] Islam, M., Balaji, P., Sadayappan, P., and Panda, D. K. (2003). QoPS: A QoS based scheme for parallel job scheduling. In *9th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '03)*, volume 2862 of *LNCS*, pages 252–268, Seattle, USA. Springer.

[104] Jackson, D. B., Snell, Q., and Clement, M. J. (2001). Core algorithms of the Maui scheduler. In *7th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '01)*, LNCS, pages 87–102, London, UK. Springer-Verlag.

[105] Jackson, S. (2004). Gold allocation manager. www.clusterresources.com/products/gold-allocation-manager.php.

[106] Jacq, N., Salzemann, J., Jacq, F., Legré, Y., Medernach, E., Montagnat, J., Maaß, A., Reichstadt, M., Schwichtenberg, H., Sridhar, M., Kasam, V., Zimmermann, M., Hofmann, M., and Breton, V. (2008). Grid-enabled virtual screening against malaria. *Journal of Grid Computing*, 6(1):29–43.

[107] Katzy, B., Zhang, C., and Löh, H. (2005). *Virtual Organizations: Systems and Practices*, chapter Reference Models for Virtual Organisations, pages 45–58. Springer Science+Business Media, Inc., New York, USA.

[108] Keahey, K., Foster, I., Freeman, T., and Zhang, X. (2006). Virtual workspaces: Achieving quality of service and quality of life in the Grid. *Scientific Programming*, 13(4):265–275.

[109] Kertész, A., Farkas, Z., Kacsuk, P., and Kiss, T. (2008). *Grid Enabled Remote Instrumentation*, chapter Grid Interoperability by Multiple Broker Utilization and Meta-Brokering, pages 303–312. Signals and Communication Technology. Springer US, New York, USA.

[110] Kim, K. H. and Buyya, R. (2007). Fair resource sharing in hierarchical virtual organizations for global Grids. In *8th IEEE/ACM International Conference on Grid Computing (Grid 2007)*, pages 50–57, Austin, USA. IEEE.

[111] Klein, N. (2008). *The Shock Doctrine: The Rise of Disaster Capitalism*. Penguin Group (Australia), Maryborough, Australia.

[112] Kurose, J. F. and Ross., K. W. (2002). *Computer Networking: A Top-Down Approach Featuring the Internet*. ISBN: 0201976994. Addison Wesley Professional, Boston, 2nd edition.

[113] Lai, K., Rasmusson, L., Adar, E., Sorkin, S., Zhang, L., and Huberman, B. A. (2004). Tycoon: an implementation of a distributed market-based resource allocation system. Technical report, HP Labs, Palo Alto, USA.

[114] Lawson, B. G. and Smirni, E. (2002). Multiple-queue backfilling scheduling with priorities and reservations for parallel systems. In *8th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '02)*, LNCS, pages 72–87, London, UK. Springer-Verlag.

[115] Legrand, I., Newman, H., Voicu, R., Cirstoiu, C., Grigoras, C., Toarta, M., and Dobre., C. (2004). Monalisa: An agent based, dynamic service system to monitor, control and optimize Grid based applications. In *Computing in High Energy and Nuclear Physics (CHEP)*, Interlaken, Switzerland.

[116] Lifka, D. A. (1995). The ANL/IBM SP scheduling system. In *Workshop on Job Scheduling Strategies for Parallel Processing (IPPS '95)*, pages 295–303, London, UK. Springer-Verlag.

[117] Litzkow, M. J., Livny, M., and Mutka, M. W. (1988). Condor – a hunter of idle workstations. In *8th International Conference of Distributed Computing Systems*, pages 104–111, San Jose, USA. Computer Society.

[118] 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.

[119] Mach, R., Lepro-Metz, R., and Jackson, S. (2006). Usage record – format recommendation. Draft, Open Grid Forum (OGF).

[120] MacLaren, J. (2003). Advance reservations: State of the art (version 4). Draft, Global Grid Forum.

[121] Margo, M. W., Yoshimoto, K., Kovatch, P., and Andrews, P. (2000). Impact of reservations on production job scheduling. In *Job Scheduling Strategies for Parallel Processing (JSSPP 2007)*, volume 4942 of *LNCS*, pages 116–131, Berlin/Heidelberg, Germany. Springer.

[122] McLean, B. and Elkind, P. (2004). *The Smartest Guys in the Room: The Amazing Rise and Scandalous Fall of Enron*, chapter Gaming California, pages 264–283. Portfolio, New York, USA.

[123] Metz, C. (2001). Interconnecting ISP networks. *IEEE Internet Computing*, 5(2):74–80.

[124] Miura, K. (2006). Overview of Japanese science Grid project NAREGI. *Progress in Informatics*, pages 67–75.

[125] Mohamed, H. and Epema, D. (2007). KOALA: A co-allocating Grid scheduler. *Concurrency and Computation: Practice and Experience (CCPE)*, 20(16):1851–1876.

[126] Montero, A. J. R., Huedo, E., Montero, R. S., and Llorente, I. M. (2007). Management of virtual machines on Globus Grids using GridWay. In *4th Workshop on High Performance Grid Computing (held with IPDPS 2007)*, pages 1–7, Long Beach, USA. IEEE Computer Society.

[127] Montero, R. S., Huedo, E., and Llorente, I. M. (2008). Dynamic deployment of custom execution environments in Grids. In *2nd International Conference on Advanced Engineering Computing and Applications in Sciences (ADVCOMP '08)*, pages 33–38, Valencia, Spain. IEEE Computer Society.

[128] 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.

[129] Nakai, J. and Van Der Wijngaart, R. F. (2003). Applicability of markets to global scheduling in grids: Critical examination of general equilibrium theory and market folklore. Technical report NAS-03-004, NASA Advanced Supercomputing (NAS) Division.

[130] Netto, M. A. S., Bubendorfer, K., and Buyya, R. (2007). SLA-Based advance reservations with flexible and adaptive time QoS parameters. In *5th international conference on Service-Oriented Computing (ICSOC 2007)*, pages 119–131, Berlin, Heidelberg, Germany. Springer.

[131] Norman, T. J., Preece, A., Chalmers, S., Jennings, N. R., Luck, M., Dang, V. D., Nguyen, T. D., Deora, V., Shao, J., Gray, W. A., and Fiddian, N. J. (2004). Agent-based formation of virtual organisations. *Knowledge-Based Systems*, 17:103–111.

[132] Norton, W. B. (2000). Internet service providers and peering. In *19th North American Network Operators Group Meeting (NANOG 19)*, Albuquerque, USA.

[133] Nurmi, D., Wolski, R., Crzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., and Zagorodnov, D. (2008). Eucalyptus: A technical report on an elastic utility computing architecture linking your programs to useful systems. Technical report 2008-10, Department of Computer Science, University of California, Santa Barbara, California, USA.

[134] O'Callaghan, J. (2006). A national Grid infrastructure for Australian researchers. *Cyberinfrastructure Technology Watch Quarterly*, 2(1).

[135] Open Science Grid (2008). A blueprint for the Open Science Grid. http://www.opensciencegrid.org/documents/.

[136] Orgerie, A.-C., Lefèvre, L., and Gelas, J.-P. (2008). Save watts in your Grid: Green strategies for energy-aware framework in large scale distributed systems. In *14th IEEE International Conference on Parallel and Distributed Systems (ICPADS'08)*, pages 171–178, Melbourne, Australia.

[137] Padala, P., Shin, K. G., Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A., and Salem, K. (2007). Adaptive control of virtualized resources in utility computing environments. In *2007 Conference on EuroSys (EuroSys 2007)*, pages 289–302, Lisbon, Portugal. ACM Press.

[138] Palankar, M. R., Iamnitchi, A., Ripeanu, M., and Garfinkel, S. (2008). Amazon s3 for science Grids: a viable solution? In *International Workshop on Data-aware Distributed Computing (DADC '08) in conjunction with HPDC 2008*, pages 55–64, New York, USA. ACM.

[139] Park, H., Lee, P., Lee, J. R., Kim, S., Kwak, J., Cho, K. W., Lim, S.-B., and Lee, J. (2006). Construction and utilization of the cyberinfrastructure in Korea. *Cyberinfrastructure Technology Watch Quarterly*, 2(1).

[140] Patel, J., Teacy, L. W. T., Jennings, N. R., Luck, M., Chalmers, S., Oren, N., Norman, T. J., Preece, A., Gray, P. M. D., Shercliff, G., Stockreisser, P. J., Shao, J., Gray, A. W., Fiddian, N. J., and Thompson, S. (2005). Agent-based virtual organisations for the Grids. *International Journal of Multi-Agent and Grid Systems*, 1(4):237–249.

[141] Peterson, L., Muir, S., Roscoe, T., and Klingaman, A. (2006). PlanetLab architecture: An overview. Technical Report PDN-06-031, PlanetLab Consortium, Princeton, USA.

[142] Peterson, L. and Wroclawski, J. (2007). Overview of the GENI architecture. GENI Design Document GDD-06-11, GENI: Global Environment for Network Innovations.

[143] Platform Computing (2003). Open source metascheduling for virtual organizations with the community scheduler framework (CSF). Technical report, Platform Computing, Ontario, Canada.

[144] Pordes, R., Petravick, D., Kramer, B., Olson, D., Livny, M., Roy, A., Avery, P., Blackburn, K., Wenaus, T., Würthwein, F., Foster, I., Gardner, R., Wilde, M., Blatecky, A., McGee, J., and Quick, R. (2007). The Open Science Grid. *Journal of Physics: Conference Series*, 78.

[145] Ram, N. M. and Ramakrishnan, S. (2006). GARUDA: India's national Grid computing initiative. *Cyberinfrastructure Technology Watch Quarterly*, 2(1).

[146] Ramakrishnan, L., Irwin, D., Grit, L., Yumerefendi, A., Iamnitchi, A., and Chase, J. (2006). Toward a doctrine of containment: Grid hosting with adaptive resource control. In *2006 ACM/IEEE Conference on Supercomputing (SC 2006)*, page 101, New York, NY, USA. ACM Press.

[147] Ranjan, R., Buyya, R., and Harwood, A. (2005). A case for cooperative and incentive-based coupling of distributed clusters. In *7th IEEE International Conference on Cluster Computing*, Boston, USA. IEEE Computer Society.

[148] Ranjan, R., Harwood, A., and Buyya, R. (2006). SLA-based coordinated superscheduling scheme for computational Grids. In *IEEE International Conference on Cluster Computing (Cluster 2006)*, pages 1–8, Barcelona, Spain. IEEE.

[149] Ranjan, R., Rahman, M., and Buyya, R. (2008). A decentralized and cooperative workflow scheduling algorithm. In *8th IEEE International Symposium on Cluster Computing and the Grid (CCGRID 2008)*, 1–8, Lyon, France. IEEE Computer Society.

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

[151] Ricci, R., Oppenheimer, D., Lepreau, J., and Vahdat, A. (2006). Lessons from resource allocators for large-scale multiuser testbeds. *SIGOPS Operating Systems Review*, 40(1):25–32.

[152] Riedel, M., Laure, E., Field, L., Casey, J., Litmaath, M., Baud, J. P., Catlett, C., Skow, D., Navarro, J., Soddemann, T., Zheng, C., Papadopoulos, P., Katz, M., Smirnova, O., Konya, B., Arzberger, P., Wurthwein, F., Rana, A., Martin, T., Welch, V., Rimovsky, T., Newhouse, S., Tanaka, Y., Tanimura, Y., Ikegami, T., Abramson, D., Enticott, C., Pordes, R., Sharma, N., Timm, S., Moont, G., Aggarwal, M., Colling, D., van der Aa, O., Sim, A., Natarajan, V., Shoshani, A., Gu, J., Galang, G., Zappi, R., Magnoni, L., Ciaschini, V., Cowles, B., Wang, S., Saeki, Y., Sato, H., Matsuoka, S., Uthayopas, P., Sriprayoonsakul, S., Koeroo, O., Viljoen, M., Pearlman, L., Pickles, S., Moloney, G., Lauret, J., Marsteller, J., Sheldon, P., Pathak, S., Witt, S. D., Mencak, J., and Phatanapherom, S. (2007). Interoperation scenarios of production e-science infrastructures. In *OGF Workshop on eScience Highlights*, Bangalore, India.

[153] Röblitz, T. and Rzadca, K. (2006). On the placement of reservations into job schedules. In *Euro-Par 2006 Parallel Processing*, volume 4128 of *LNCS*, pages 198–210. Springer Berlin/Heidelberg.

[154] Röblitz, T., Schintke, F., and Wendler, J. (2004). Elastic Grid reservations with user-defined optimization policies. In *Workshop on Adaptive Grid Middleware (AGridM 2004)*, Antibes Juan-les-Pins, France.

[155] Rowstron, A. and Druschel, P. (2001). Pastry: scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, volume 2218 of *LNCS*, pages 329–350, Berlin/Heidelberg, Germany. Springer.

[156] Ruth, P., Jiang, X., Xu, D., and Goasguen, S. (2005a). Virtual distributed environments in a shared infrastructure. *IEEE Computer*, 38(5):63–69.

[157] Ruth, P., McGachey, P., and Xu, D. (2005b). VioCluster: Virtualization for dynamic computational domain. In *IEEE International on Cluster Computing (Cluster 2005)*, pages 1–10, Burlington, USA. IEEE.

[158] Ruth, P., Rhee, J., Xu, D., Kennell, R., and Goasguen, S. (2006). Autonomic live adaptation of virtual computational environments in a multi-domain infrastructure. In *3rd IEEE International Conference on Autonomic Computing (ICAC 2006)*, pages 5–14, Dublin, Ireland. IEEE.

[159] Sairamesh, J., Stanbridge, P., Ausio, J., Keser, C., and Karabulut, Y. (2005). Business models for virtual organization management and interoperability. Deliverable document 01945 prepared for TrustCom and the European Commission.

[160] Sandholm, T., Gardfjäll, P., Elmroth, E., Johnsson, L., and Mulmo, O. (2004). An OGSA-based accounting system for allocation enforcement across HPC centers. In *2nd international Conference on Service Oriented Computing (ICSOC 2004)*, pages 279–288, New York, USA. ACM Press.

[161] Schwiegelshohn, U. and Yahyapour, R. (1999). Resource allocation and scheduling in metasystems. In *7th International Conference on High-Performance Computing and Networking (HPCN Europe '99)*, pages 851–860, London, UK. Springer-Verlag.

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

[163] Shneidman, J., Ng, C., Parkes, D. C., AuYoung, A., Vahdat, A. C. S. A., and Chun, B. (2005). Why markets could (but don't currently) solve resource allocation problems in systems. In *10th Conference on Hot Topics in Operating Systems (HOTOS'05)*, pages 7–7, Berkeley, CA, USA. USENIX Association.

[164] Shoykhet, A., Lange, J., and Dinda, P. (2004). Virtuoso: A system for virtual machine marketplaces. Technical report NWU-CS-04-39, Electrical Engineering and Computer Science Department, Northwestern University, Evanston/Chicago, IL.

[165] Siddiqui, M., Villazón, A., and Fahringer, T. (2006). Grid capacity planning with negotiation-based advance reservation for optimized QoS. In *2006 ACM/IEEE Conference on Supercomputing (SC 2006)*, pages 21–21, New York, USA. ACM.

[166] Singh, G., Kesselman, C., and Deelman, E. (2006). Application-level resource provisioning on the Grid. In *2nd IEEE International Conference on e-Science and Grid Computing (e-Science 2006)*, pages 83–83, Amsterdam, The Netherlands.

[167] Singh, G., Kesselman, C., and Deelman, E. (2007a). Adaptive pricing for resource reservations in shared environments. In *8th IEEE/ACM International Conference on Grid Computing (Grid 2007)*, pages 74–80, Austin, USA. ACM/IEEE.

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

[169] Smith, W., Foster, I., and Taylor, V. (2000). Scheduling with advanced reservations. In *14th International Parallel and Distributed Processing Symposium (IPDPS 2000)*, pages 127–132, Cancun, Mexico.

[170] Snell, Q., Clement, M., Jackson, D., and Gregory, C. (2000). The performance impact of advance reservation meta-scheduling. In *Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP 2000)*, volume 1911 of *LNCS*, pages 137–153. Springer.

[171] Sotomayor, B., Keahey, K., and Foster, I. (2008). Combining batch execution and leasing using virtual machines. In *17th International Symposium on High performance Distributed Computing (HPDC 2008)*, pages 87–96, New York, USA. ACM.

[172] Srinivasan, S., Kettimuthu, R., Subramani, V., and Sadayappan, P. (2002). Selective reservation strategies for backfill job scheduling. In *8th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '02)*, volume 2537 of *LNCS*, pages 55–71, London, UK. Springer Berlin Heidelberg.

[173] Sulistio, A., Cibej, U., Prasad, S., and Buyya, R. (2009). GarQ: An efficient scheduling data structure for advance reservations of Grid resources. *International Journal of Parallel, Emergent and Distributed Systems*, 24(1):1–19.

[174] Sulistio, A., Kim, K. H., and Buyya, R. (2007). On incorporating an on-line strip packing algorithm into elastic Grid reservation-based systems. In *13th International Conference on Parallel and Distributed Systems (ICPADS 2007)*, volume 1, pages 1–8, Hsinchu, Taiwan.

[175] Sulistio, A., Kim, K. H., and Buyya, R. (2008). Managing cancellations and no-shows of reservations with overbooking to increase resource revenue. In *8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2008)*, pages 267–276, Lyon, France. IEEE Computer Society.

[176] Sundararaj, A. I. (2006). *Automatic, Run-time and Dynamic Adaptation of Distributed Applications Executing in Virtual Environments*. PhD thesis and technical report, Department of Electrical Engineering and Computer Science, Northwestern University, Evanston/Chicago, IL, USA.

[177] Sundararaj, A. I., Gupta, A., and Dinda, P. A. (2005). Increasing application performance in virtual environments through run-time inference and adaptation. In *14th IEEE International Symposium on High Performance Distributed Computing (HPDC 2005)*, pages 47–58, Research Triangle Park, NC.

[178] Surana, S., Godfrey, B., Lakshminarayanan, K., Karp, R., and Stoica, I. (2006). Load balancing in dynamic structured peer-to-peer systems. *Performance Evaluation*, 63(3):217–240.

[179] Svirskas, A., Arevas, A., Wilson, M., and Matthews, B. (2005). Secure and trusted virtual organization management. *ERCIM News*, (63).

[180] Talby, D. and Feitelson, D. G. (1999). Supporting priorities and improving utilization of the IBM SP scheduler using slack-based backfilling. In *13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing*.

[181] Tatezono, M., Maruyama, N., and Matsuoka, S. (2006). Making wide-area, multisite MPI feasible using Xen VM. In *Workshop on Frontiers of High Performance Computing and Networking (held with ISPA 2006)*, volume 4331 of *LNCS*, pages 387–396, Berlin/Heidelberg. Springer.

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

[183] Urgaonkar, B. and Roscoe, P. S. T. (2002). Resource overbooking and application profiling in shared hosting platforms. In *5th Symposium on Operating Systems Design and Implementation*, pages 239–254, Boston, Massachusetts.

[184] Varela, C. A. and Agha, G. (2001). Programming dynamically reconfigurable open systems with SALSA. *ACM SIGPLAN Notices, OOPSLA'2001 Intriguing Technology Track Proceedings*, 36(12):20–34.

[185] Vázquez-Poletti, J. L., Huedo, E., Montero, R. S., and Llorente, I. M. (2007). A comparison between two Grid scheduling philosophies: EGEE WMS and GridWay. *Multiagent and Grid Systems*, 3(4):429–439.

[186] Venugopal, S., Buyya, R., and Winton, L. (2006). A Grid service broker for scheduling e-science applications on global data Grids: Research articles. *Concurrency and Computation: Practice and Experience (CCPE)*, 18(6):685–699.

[187] Venugopal, S., Nadiminti, K., Gibbins, H., and Buyya, R. (2008). Designing a resource broker for heterogeneous Grids. *Software: Practice and Experience*, 38(8):793–825.

[188] Veridian Systems Inc. (2005). OpenPBS: The Portable Batch System software. http://www.openpbs.org/scheduler.html.

[189] Wäldrich, O., Wieder, P., and Ziegler, W. (2006). A meta-scheduling service for co-allocating arbitrary types of resources. In *Parallel Processing and Applied Mathematics (PPAM 2005)*, volume 3911 of *LNCS*, pages 782–791, Berlin/Heidelberg, Germany.

[190] Wang, Y., Scardaci, D., Yan, B., and Huang, Y. (2007). Interconnect EGEE and CNGRID e-infrastructures through interoperability between gLite and GOS middlewares. In *International Grid Interoperability and Interoperation Workshop (IGIIW 2007) with e-Science 2007*, pages 553–560, Bangalore, India. IEEE Computer Society.

[191] Wang, Y.-T. and Morris, R. J. T. (1985). Load sharing in distributed systems. *IEEE Transactions on Computers*, C-34(3):204–217.

[192] Wasson, G. and Humphrey, M. (2003). Policy and enforcement in virtual organizations. In *4th International Workshop on Grid Computing*, pages 125–132, Washington, DC, USA. IEEE Computer Society.

[193] Weiss, A. (2007). Computing in the Clouds. *netWorker*, 11(4):16–25.

[194] Weiss, M. B. and Shin, S. J. (2004). Internet interconnection economic model and its analysis: Peering and settlement. *NETNOMICS*, 6(1):43–57.

[195] Wesner, S., Dimitrakos, T., and Jeffrey, K. (2004). Akogrimo - the Grid goes mobile. *ERCIM News*, (59):32–33.

[196] Wieczorek, M., Siddiqui, M., Villazon, A., Prodan, R., and Fahringer, T. (2006). Applying advance reservation to increase predictability of workflow execution on the Grid. In *2nd IEEE International Conference on e-Science and Grid Computing (E-Science 2006)*, page 82, Washington, DC, USA. IEEE Computer Society.

[197] Wolski, R., Plank, J. S., Brevik, J., and Bryan, T. (2001). Analyzing market-based resource allocation strategies for the computational Grids. *The International Journal of High Performance Computing Applications*, 15(3):258–281.

[198] Yu, J. and Buyya, R. (2006). A taxonomy of workflow management systems for Grid computing. *Journal of Grid Computing*, 3(3–4):171–200.