Performance Analysis of Allocation Policies for InterGrid Resource Provisioning

Marcos Dias de Assunção [a,*], Rajkumar Buyya [a]

[a] *Grid Computing and Distributed Systems (GRIDS) Laboratory*
Department of Computer Science and Software Engineering
The University of Melbourne
Victoria 3010, Australia

Abstract

Several Grids have been established and used for varying science applications during the last years. Most of these Grids, however, work in isolation and with different utilisation levels. Previous work introduced an architecture and a mechanism to enable resource sharing amongst Grids. It demonstrated that there can be benefits for a Grid to offload requests or provide spare resources to another Grid, thus reducing the cost of over-provisioning. These benefits derive from the fact that resource utilisation within a Grid has fixed and operational costs such as those with electricity providers and system administrators. In this work, we address the problem of resource provisioning to Grid applications in multiple-Grid environments. The provisioning is carried out based on availability information obtained from queueing-based resource management systems deployed at the provider sites who are the participants of the Grids. We evaluate the performance of different allocation policies. In contrast to existing work on load sharing across Grids, the policies described here take into account the local load of resource providers, imprecise availability information and the monetary compensation of providers. In addition, we evaluate these policies along with mechanism that allows resource sharing amongst Grids. Experimental results obtained through simulation show that the mechanism and policies are effective in redirecting requests thus improving the applications' average weighted response time.

Key words: resource provisioning, Grid computing, interGrid resource allocation

* Corresponding author. Address: 111 Barry Street, Carlton, VIC, 3053, Australia.
Email addresses: marcosd@csse.unimelb.edu.au (Marcos Dias de Assunção), raj@csse.unimelb.edu.au (Rajkumar Buyya).

1 Introduction

Advances in Grid computing have enabled the creation of Grid-based resource sharing networks such as TeraGrid [1,2], Naregi [3], Open Science Grid [4], and PlanetLab [5]. These networks, composed of multiple resource providers, enable collaborative work and resource sharing amongst groups of individuals and organisations. These collaborations, widely known as Virtual Organisations (VOs) [6], require resources from multiple computing sites.

Although these Grids have contributed to various sciences and disciplines, they mostly work in isolation. The Grid Interoperability Now - Community Group (GIN-CG) [7] is working on providing interoperability between Grids by developing components and adapters that enable secure job submissions, data transfers and information queries. Even though GIN-CG's efforts are relevant, its members also highlight the need for common allocation and brokering of resources across Grids. [1] In addition, Iosup *et al.* [8] have identified the need for resource management across Grids. They have shown that there is a load imbalance between current Grids.

The resource utilisation within a Grid has fixed and operational costs such as those with electricity providers and system administrators. Consequently, there can be benefits for a Grid to provide spare capacity to peering Grids, possibly in return for regular payments, and to acquire resources from peering Grids to serve occasional internal peak demands. This load management approach can reduce the costs incurred by over-provisioning. We have proposed in previous work [9] a resource exchange mechanism that enables a Grid, under peak load conditions, to redirect requests to another Grid. In contrast to existing work in load management across Grids [8,7] the proposed mechanism takes into account the monetary compensation of resource providers. In this work, we address the problem of resource provisioning in environments with multiple Grids.

Emerging deadline-driven Grid applications require access to several resources and predictable Quality of Service (QoS). However, it is difficult to provision resources to these applications because of the complexity of providing guarantees about the start or completion times of applications currently in execution or waiting in the queue. The resources contributed by providers participating of a Grid are generally clusters of computers managed by queueing-based Resource Management Systems (RMSs), such as PBS [10] and Condor [11]. These RMSs generally use optimisations to the First Come First Served (FCFS) policy such as backfilling to reduce the scheduling queue fragmentation, improve job response time and maximise resource utilisation. These optimisations make

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

it difficult to predict the resource availability over a time frame as the jobs' start and completion times are dependent on resource workloads.

Grid users commonly access resources from a Grid via mediators such as brokers or gateways [12,13]. The design of gateways that provision resources to deadline-driven applications relying on information given by current RMSs may be complex and prone to scheduling decisions that are far from optimal. Furthermore, a gateway representing a Grid can have peering arrangements or contracts with other gateways through which they co-ordinate the resource provisioning. This complicates provisioning even further as now a gateway needs to not only provision resources to its users, but also provision spare capacity to other gateways. Previous work has demonstrated how information about fragments in the scheduling queue, or free time slots, can be obtained from RMSs and provided to gateways to be provisioned to Grid applications[14,15]. In this work, we utilise this information as basis for resource provisioning across Grids.

The present work specifically extends previous studies on InterGrid load management [9] and resource provisioning in multiple-site environments [15] in the following manner. First, we extend the provisioning policies in order to consider the cost of delegating resources to a gateway. Second, the mechanism for load management across Grids that previously assumed an ON/OFF approach for modelling resource providers' load, now utilises information obtained from scheduling policies using conservative backfilling and multiple resource partitions. Previous studies used of ON/OFF models [16] wherein on and off intervals represent off-peak and peak periods respectively. However, the queuing-based scheduling policies enhance the evaluation of the overall mechanism by modelling a scenario closer to reality. Third, the experiments carried out in this work measure the jobs' average weighted response time and the amount of currency spent by gateways in the contract network for each scenario. Experimental results show that the approach is effective reducing the applications' average weighted response time.

The rest of this paper is organised as follows. Section 2 presents the related work. In Section 3, we describe the InterGrid scenario. We describe the policies used by resource providers in Section 4. Section 5 discusses the resource provisioning and load sharing across Grids. We present and elaborate on the performance evaluation and experimental results in Section 6. Section 7 concludes the paper and presents future work.

2 Related Work

The proposed InterGrid architecture, mechanisms and policies are related to previous systems and techniques in several manners. Namely, we focus on

research related to:

Resource sharing networks and inter-operation efforts: Several Grids have been built over the past few years [1–4], but most of these work in isolation. Recently, two major initiatives have attempted to link resource sharing networks: the Global Environment for Network Innovations (GENI) [17] and the GIN-CG under the Open Grid Forum [7]. GENI has evolved from the PlanetLab architecture [5] to allow the federation of autonomous networks controlled by different organisations [17]. GIN-CG [7] is leveraging community efforts to address problems regarding security, standard job submission, data management and information services. GIN-CG highlights the importance of policies for interGrid brokering and resource management. The present work attempts to build on these to investigate resource management in the InterGrid.

Iosup *et al.* [8] have presented a seminal work on inter-operation of Grids. In a similar way, gateways work as site recommenders so matching requests to resources available. On the other hand, the policies presented here differ with respect to the resource exchange protocol and the consideration for economic compensation for the resources acquired from the providers.

Intermediate resource agents: Shirako [18,19] provides an architecture for resource management based on a resource leasing abstraction. Sites delegate limited power to allocate their resources by registering their offerings with brokers. Guest applications can acquire resources from brokers by leasing them for a specified time. Grit *et al.* [20] investigate the number of Virtual Machine (VM) migrations incurred when a broker and provider sites use either conflicting or synchronised policies for resource provisioning and VM placement. They show that when providers and the broker use conflicting policies, the number of migrations can be high. This paper focuses on the policies for resource exchange amongst gateways, which are similar to brokers in Shirako. However, to the best of our knowledge, resource exchange amongst Shirako brokers has not been explored yet.

Federated clusters and load sharing: Ranjan *et al.* [21] have proposed a Service Level Agreement (SLA) based coordination mechanism for Grid superscheduling. Different from that work, an InterGrid Gateway is a broker with partial information about the free time slots of resource from a group of providers and does not have full control over the contributed resources.

Balazinska *et al.* [22] have proposed a load balancing mechanism for Medusa. Medusa is a stream processing system that allows the migration of stream processing operators from overloaded resources to resources with spare capacity. We have taken inspiration from Medusa's mechanism and extended it to support the exchange of resources amongst Grids. The mechanism used for load sharing amongst gateways is derived from Medusa, but differs in terms of the negotiation protocol for exchanging resources between Grids, the resource

selection and request redirection policies.

Wang and Morris [23] provide a taxonomy on load sharing in distributed systems. Some findings include that efficiency in load sharing depends on the environment and server-initiative tends to outperform source-initiative strategies when the same amount of information about stakeholders is available. In our scenario, resources have multiple processors; also, resources are heterogeneous in the number of processors. These make the local scheduling subproblem different; in addition, resource management across Grids introduces a third subproblem: the load sharing between Grids. Surana *et al.* [24] address the load balancing in DHT-based P2P networks. 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 balance. This work, in contrast, does not perform migration of virtual servers, and focus on the redirection and assignment of resources to requests.

Resource provisioning: Singh *et al.* [14,25] have presented a provisioning model wherein Grid sites provide information on the time slots over which sets of resources will be available. The sites provide their resources to the Grid in return for payments, thus they present a cost structure consisting of fixed and variable costs over the resources provided. The provisioning model is evaluated considering the scheduling of workflow applications. The main goal is to find a subset of the aggregated resource availability, termed resource plan, such that both the allocation cost and the application makespan are minimised. They utilise a Multi-Objective Genetic Algorithm (MOGA) approach to approximate the group of resource plans that correspond to the *Pareto-optimal* set. Experiments have been carried out considering one cluster and one broker at time. Our work differs from that by Singh *et al.* in the sense that it previously investigated multiple approaches to obtain availability information and how reliable this information can be in multiple site environments [15]. In this paper we utilise this information and evaluate the load sharing across brokers (in this work termed gateways), which was not explored by Singh *et al.*

EASY backfilling and conservative backfilling: Schedulers generally use optimisations to the FCFS policy such as EASY backfilling and conservative backfilling [26], which allow a job to jump in the queue and execute earlier than jobs that arrived before it; given that enough resources are available and other waiting jobs are not delayed. Under conservative backfilling, a job can be used to backfill and execute earlier given that it does not delay any other job waiting in the queue. EASY backfilling, on the other hand, uses a job to backfill and start execution if it does not delay only the first job in the queue - also termed pivot job. The schedule generally contains the expected completion of running jobs and the start time of the pivot job only. Some schedulers 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 [27]. In such case, if the maximum number of pivots is set to 5, for example, and there are 5 jobs waiting in the queue, a 6^{th} job that just arrived is used to backfill if it does not delay any of the 5 pivot jobs. If the maximum number of pivots is set to a large number, the EASY backfilling algorithm becomes conservative backfilling. We utilise and extend policies based on these techniques for resource provisioning.

Multiple resource partition policies: Work on multiple resource partitions and priority scheduling has shown to reduce the job slowdown compared to EASY backfilling policies [28]. We build on this effort and extend it to enable other multiple partition policies. We also propose a new multiple resource partition policy based on load forecasts for resource provisioning.

3 Provisioning in InterGrid Environments

The InterGrid is an architecture and policies for enabling resource sharing across Grids. Figure 1 provides an overview of the architecture, whereas a detailed discussion of the architecture is available in previous work [29]. The resources provisioned and exchanged between Grids can be physical or virtual resources such as Virtual Machines (VMs).

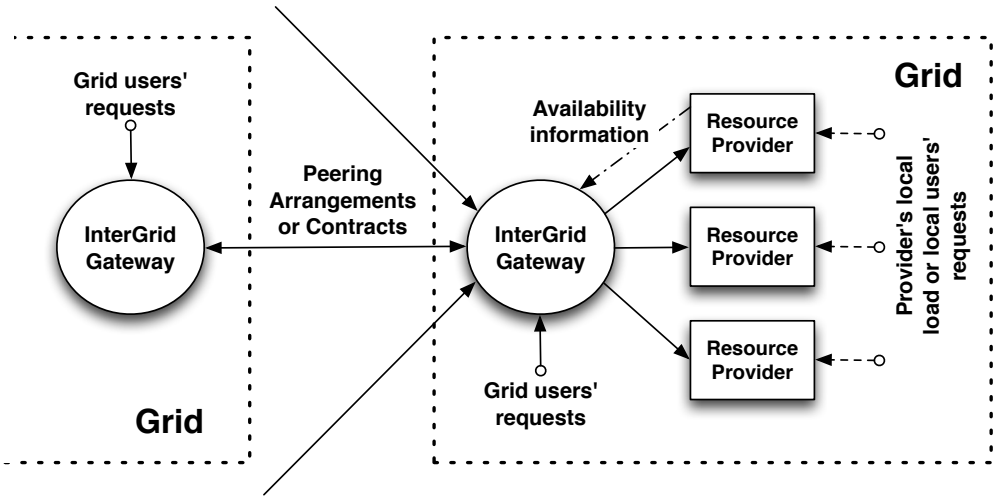

Fig. 1. The provisioning scenario with multiple Grids considered in this work.

A Resource Provider (RP) contributes a share of computational resources, storage resources, networks, application services or other type of resource to a Grid in return for regular payments. A RP has local users whose resource demands need to be satisfied, yet it delegates provisioning rights over spare resources to an InterGrid Gateway (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 time frame over which they will be available. The resources provided can be

physical or virtual resources such as Virtual Machines (VMs) and the delegation can be made through a secure protocol such as SHARP [30]. Protocols for secure delegation, however, are not in the scope of this paper; this work focus on the resource provisioning aspect. Although a Grid can have a resource management system of its own (i.e. an IntraGrid Resource Manager), for the sake of simplicity here, a RP delegates provisioning rights directly to an IGG.

A Grid has peering arrangements with other Grids, managed by IGGs and, through which they coordinate the use of resources of the InterGrid. An IGG is aware of the terms of the peering with other Grids; provides Grid selection capabilities by selecting a Grid able to provide the required resources; and replies to requests from other IGGs.

CLient applications (CLs) can implement resource management mechanisms of their own. However, we envisage that applications can have performance and environment isolation provided by Distributed Virtual Environments (DVEs), which can be created on top of the InterGrid and can span multiple Grids. DVEs can leverage virtualisation technologies [31] and provide an overlay network comprising the resources allocated from the InterGrid. When a CL requires a number of resources to deploy or execute an application, it requests the IGG based on the application demand. When the individual Grid cannot provide the required resources, the IGG selects a peering Grid based on the peering agreements and the policies in place. The CL is then given a resource ticket that will later be passed to the selected provider in return for the required resources.

4 Resource Provider Policies

We have previously investigated scheduling policies for resource providers [15] that enable an IGG to obtain resource availability information in the form of free time slots. This work considers a subset of the investigated policies, which have previously demonstrated good performance. The policies utilise an 'availability profile' similar to that described by Mu'alem and Feitelson [26]. The availability profile is a list whose entries describe the CPU availability at particular times in the future. These correspond to the completion or start of jobs and advance reservations. Jobs with the same completion time or scheduled to start at the completion of another job share entries in the profile. By scanning the availability profile and using load forecasts the resource providers inform the gateway about the free time slots; the gateway in turn can carry out provisioning decisions based on this information.

Figure 2 illustrates how a resource provider supplies the gateway with information on the free time slots. The example considers conservative backfilling. Under conservative backfilling, a job is used to backfill and start execution ear-

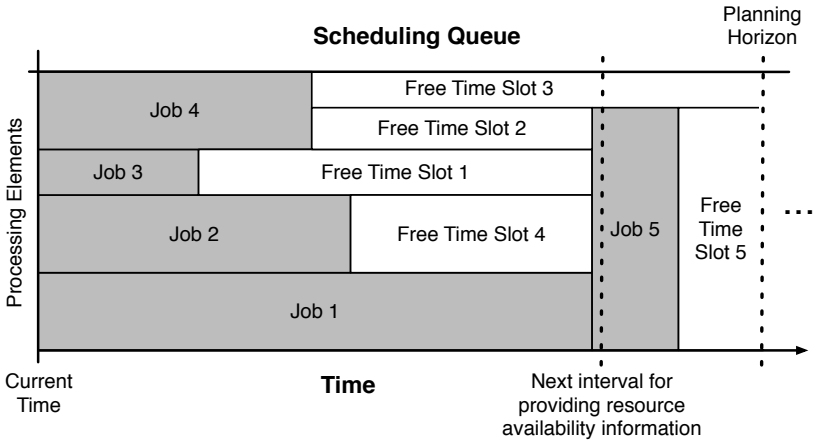

Fig. 2. Free time slots (conservative backfilling).

lier than expected, given that it does not delay any other job in the scheduling queue [26]. In order to reduce complexity, the schedule for the job is generally determined at its arrival and the availability profile is updated accordingly. Given those conditions, it is possible to obtain the free time slots by scanning the availability profile. In that case, the availability profile is scanned until a given time horizon thus creating windows of availability or free time slots; the finish time of a free time slot is either the finish time of a job in the waiting queue or the planning horizon. The horizon corresponds to the time limit up to which the availability information is evaluated. If the horizon is set to ∞, the provider will be disclosing 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.

We have also investigated other policies based on multiple resource partitions [28] and load forecasts [15]. In our implementation, the policy divides the resources available in multiple partitions and assigns jobs to these partitions according to given predicates. A partition can borrow resources from another when they are not in use by the latter and borrowing is allowed by the scheduler. We introduced a new policy in which, at time intervals, the partitions are resized by the scheduler based on a load forecast computed from information collected at previous intervals. 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 or not. If the forecast load was underestimated, the policy resizes the partitions according to the load from the last resizing period until the current time and backfills the jobs, starting with the local jobs.

Algorithm 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; the procedure *getFreeTimeSlots* triggers *reallocationEvent* to verify whether the previous forecast has turned out

to be precise or if a reallocation is required.

```
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{0, 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
```

Algorithm 1: Provider's load forecasting policy.

We use EASY backfilling with configurable maximum number of pivots, similarly to MAUI scheduler [27]. This enables the policy to be converted to conservative backfilling by setting the maximum number of pivots to a large value, here represented by ∞. From line 3 to 4 of Algorithm 1, the scheduler becomes conservative backfilling based by setting the number of pivots in each partition to ∞. It also schedules the jobs currently waiting in the queue. After that, the scheduler returns to EASY backfilling (line 5). Then, 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 amount 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 that is the case, it throws the towel and

turns the policy to conservative backfilling and informs the gateway about the availability.

The policies described in the previous work [15] did not consider the pricing of resources by providers. In this work we apply a mechanism for pricing the free time slots delegated by a resource provider to the gateway. The resource providers use a pricing function for a resource unit given by Equation 1.

$$price = cost + (cost * load) \tag{1}$$

where *cost* is the fixed cost of a unit of resource at the provider; *load* is obtained from the policy in use by the resource provider; *load* is the estimate when the policy supports forecasts or the actual load of both running and waiting jobs in the queue. A resource unit corresponds to one resource per second (i.e. a second of a CPU). Although straightforward, this pricing function has two components that capture namely the fixed cost of resources and the variable price caused by the demand.

5 Resource Provisioning and Load Sharing

For each IGG_i representing the Grid g_i, the allocation of its resources by its user communities over a unit of time represents a cost. The real-valued cost function of the participating 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 in use in Grid g_i. Therefore, the cost given by $cost_i(L)$ depends on the number of resources allocated by the requests. Although each Grid could have its own cost function, in this work, the participating Grids utilise a quadratic cost function. The use of a quadratic function allows us 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 2; where n is the number of resource providers in Grid g_i; 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. This horizon is the request deadline when calculating the marginal cost described next, or the time of the next contract update; that is, the next time when the IGGs update the prices of

units negotiated.

$$p_{cost} = \sum_{i=1}^{n} \left(cp_i \left(\frac{ru_i}{\sum_{j=1}^{n} ru_j} \right) \right) \tag{2}$$

Request redirections are based on the per request marginal cost, $mc_i : (u, L) \rightarrow \Re$ which is the increment in cost for Grid g_i for agreeing to provide resource units required by request u given its current load or allocations. If the request u requires resource units that place u_{load} load in Grid g_i, then the marginal cost of serving u is derived by Equation 3.

$$mc_i = cost_i(L + u_{load}) - cost_i(L) \tag{3}$$

Each Grid g_i has a load threshold, by crossing which it 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 hold 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 because the marginal cost of fulfilling the requests is higher than the amount that it would have to pay to the other Grid to serve them.

Figure 3 illustrates the negotiation between IGG_i and IGG_j that have a price range contract. Consider that IGG_i sends an offer to IGG_j when IGG_i's marginal cost is higher than the minimum price of the contract with IGG_j (1). IGG_j in turn sends an accept whose price is the price in the initial offer if its own marginal cost is lower than the amount that IGG_i is willing to pay (2). If IGG_j's marginal cost is higher than the amount offered by IGG_i, but smaller than the maximum amount that IGG_i would possibly pay (i.e. $price_{max} * request_{units}$), then it sends a counter-offer whose price is $mc_j/request_{units}$ (3); otherwise, the offer is rejected (4). IGG_i in turn, will accept the counter-offer if its marginal cost is still higher than the amount asked by IGG_j (5); otherwise, the counter-offer is rejected (6).

5.1 Contract Types

There are two kinds of contracts: fixed price (i.e. $PR(C_{i,j}) : [price_{max}, price_{max}]$ where $price_{max}$ is the fixed price) and price range contracts (i.e. $PR(C_{i,j}) : [price_{max} - \Delta, price_{max})$, where Δ determines the price range. In the case of

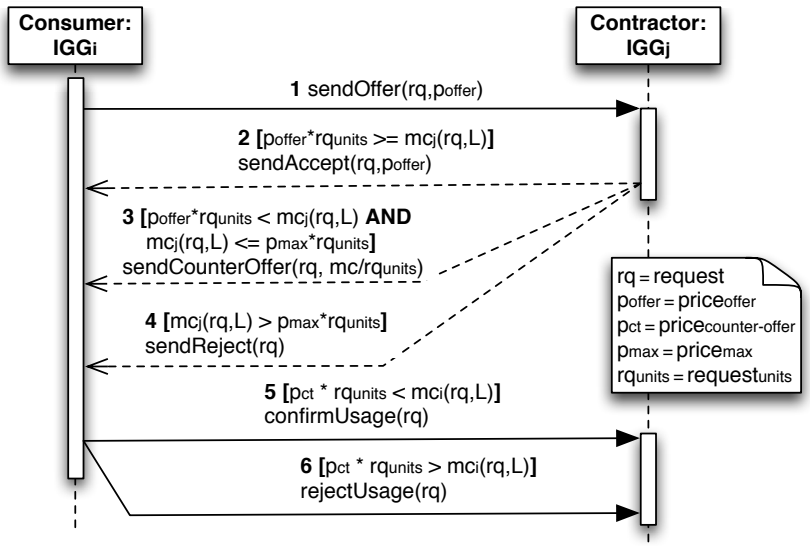

Fig. 3. Redirection negotiation.

price range contracts, participating Grids have to negotiate the final price at runtime. As discussed by Balazinska *et al.* [22], 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 marginal cost is higher than the exact price of a contract.

We define the price range for a resource unit considering 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 marginal cost due to removing k from the Grid's L is represented by δ_k, which is defined by Equation 4.

$$\delta_k(L) = \frac{mc(u, L - u_{load}) - mc(u, L - k - u_{load})}{u_{units}} \tag{4}$$

δ_k is the approximate difference in the cost function gradient 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 marginal cost exceeds $price_{max}$. In order to estimate the price range for a resource unit in the contracts in our experiments, we let L be the load threshold; u_{units} be 1 and $\Delta = \delta_k$. We evaluate different values for L and k.

5.2 Provisioning Policies

The provisioning policies define how an IGG provisions resources to its Grid users and how it offloads requests to peering Grids considering a contract network [9]. Additionally, it specifies how the IGG accepts requests from other Grids. During a given time interval or while requests previously submitted are being handled, IGG_i stores the requests in the waiting queue. After the interval (5 minutes in the experiments performed in this work), 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 peer IGG_j. The redirection occurs if the price that IGG_i would pay to peer IGG_j is lower than the request's marginal cost; otherwise, the request is served by the local Grid.

On the other hand, the peer IGG_j stores offers containing requests during a time interval (also 5 minutes in the experiments performed here). After each interval, IGG_j evaluates the offers. It sorts the offers by decreasing order of price and checks whether the price offered is higher than the requests' marginal cost. If that is the case, IGG_j accepts the request. IGG_j then creates a resource ticket for the request. If the marginal cost is higher than the amount offered but lower than the maximum amount IGG_i would be willing to pay (i.e. the amount based on the maximum price specified in the contract), IGG_j will create a counter offer whose price is the request's marginal cost. If the marginal cost is higher than the maximum price IGG_i can pay, then the offer is rejected (Figure 3).

IGGs use an earliest start time policy to select the resources to serve a request. To calculate the load in the Grid, the load caused by the request and consequently its marginal cost we use the request deadline [9]. As the requests considered in this work require a best-effort service and do not have a deadline, we create a virtual deadline given by: the latest start time based on the time slots held by the gateway plus the runtime estimate.

5.3 Storing Free Time Slots at the IGG

The resource providers issue free time slots and send them to the IGG on a periodical basis. The IGG maintains the availability information given by a provider on a modified red-black tree [32]. Each node has two references namely to its predecessor and successor nodes thus forming a linked list. This tree is analogous to the availability profile described by Mu'alem and Feitelson [26]; the nodes are ordered according to their times. That is, a free slot may lead to the creation of two nodes in the tree, namely to mark its start and finish times; free time slots can share nodes.

6 Performance Evaluation

6.1 Experimental Scenario

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 [33], Grid'5000 [34] and AuverGrid [35] comprise 5, 15 and 5 clusters respectively. For detailed information on the characteristics of the clusters we refer to Iosup *et al.* [8] and the Grid Workloads Archive website. [2] Figure 4 presents the environment simulated.

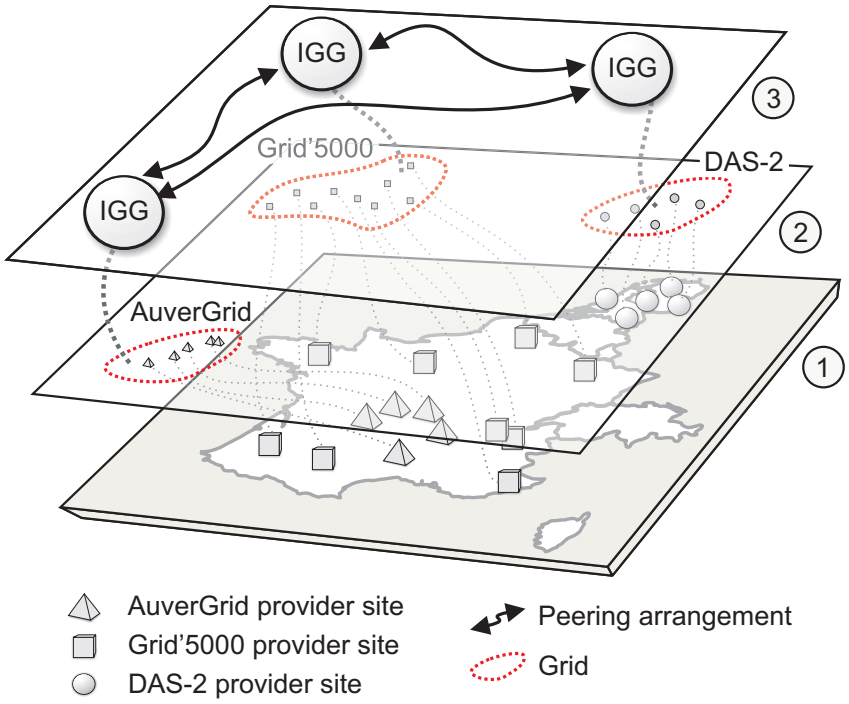

Fig. 4. Contract topology simulated.

The evaluation is performed through simulation by using a modified version of GridSim. [3] We resort to simulation because it provides a controllable environment and enables us to carry out repeatable experiments.

The workloads of the Grids are modelled using traces obtained from the Grid Workloads Archive. We divided the traces into 4-month intervals. We 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 cool-down phase in which resources start become idle by the lack

[2] More details about the modelled resources and the traces used can be obtained from the Grid Workloads Archive at http://gwa.ewi.tudelft.nl/pmwiki/

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

of job submissions. Specifically, the last simulation event is the arrival of the last job submitted in any of the utilised workloads. Additionally, we attempt to eliminate the system warm-up by disregarding the first two weeks of the experiments. For the load forecast policy, the second week is used for training.

The resource providers' local jobs are generated according to the workload model proposed by Lublin and Feitelson [36]; we refer to this model as Lublin 99. We configure the Lublin 99 model to generate type-less jobs (i.e. we do make distinctions between batch and interactive jobs); the maximum number of CPUs used by the generated jobs is set accordingly to the number of nodes in the clusters; we generate four month long workloads. We change two parameters of the Lublin 99 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 arrival rates, which we believe is representative of Grid resources. For load forecasting we use a weighted exponential moving average [37], considering a window of 25 intervals.

We perform experiments considering L in Equation 4 equals to 99% of utilisation and k set to 10% of the Grid's resources. In this case, when the fixed price ($price_{max}$) of a contract is the marginal cost of accepting 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 marginal cost at 99% and at 89% of utilisation.

6.2 Performance Metrics

The performance evaluation considers two metrics: the Average Weighted Response Time (AWRT) [38] of jobs and the percentage of the generated load redirected by the IGGs. The AWRT measures how long in average users wait to have their jobs executed. A short AWRT indicates that on average users do not wait long for their jobs to complete. 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 or not. Although we do not provide a cost minimisation policy, the experiments also measure the amount of currency

spent by IGGs in acquiring resources to serve its users.

$$AWRT_k = \frac{\sum_{j \in \tau_k} p_j \cdot m_j \cdot (c_j - s_j)}{\sum_{j \in \tau_k} p_j \cdot m_j} \tag{5}$$

The AWRT is given by Equation 5, where m_j is the number of processors required by job j, p_j is the execution time of the job, c_j is the time of completion of the job and s_j is its submission time. The resource consumption ($p_j \cdot m_j$) of each job j is used as the weight.

6.3 Policy Acronyms

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

6.4 Experimental Results

The parameters used for the experiments are summarized in Table 1. The fixed cost of a resource in Equation 1 is drawn uniformly from 0.9 to 1. The load threshold (L) and k are set to 99% and 10% respectively. The IGGs inform one another about the fixed prices or the price ranges in their contracts based on the current resource demand at intervals between 2 and 6 hours.

First Experiment: The first experiment evaluates the AWRT of both Grid and local jobs in a scenario wherein the providers send the availability information to the IGG every 12 hours. Figure 5 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 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 EASY backfilling with 'submit to the least loaded resource' (i.e. bar 1) is shown for the sake of comparison. We observe that in an overall, the AWRT is reduced by the peering of Grids under both fixed-price and price-range contracts. This occurs despite the fact that IGGs accumulate a number of requests to be handled every 5 minutes when contracts exist, in contrast to Eb+least-load in which requests are handled upon their arrival at the gateway. 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, our initial expectations were that this policy would have less impact on the providers' local jobs because they resize the free time slots given to the gateway based on load forecasts.

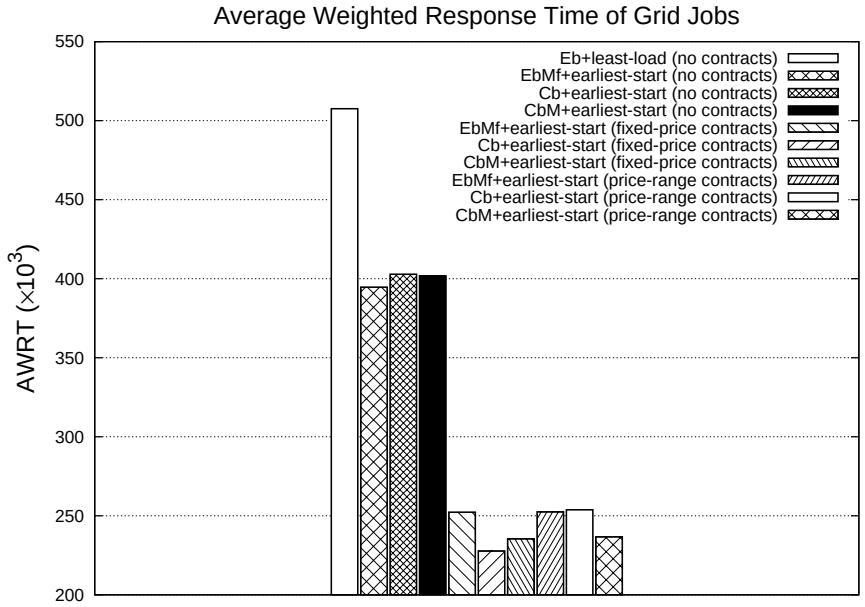

Fig. 5. Average Weighted Response Time (AWRT) of Grid jobs.

The AWRT of local jobs show the impact of peering of Grids in the providers' user applications (Figure 6). 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. Although Grids can redirect more requests under price-range contracts, this redirection does not seem to improve the jobs' AWRT. This is probably caused by the fact that IGGs handle the requests and offers every 5 minutes. With price-range contracts, IGGs redirect more requests and this delay seems to impact on the AWRT. We can improve this scenario by introducing a buy-it-now mechanism where a Grid could make an offer for immediate access to resources [39]. However, the investigation of

such a mechanism is not in the scope of this paper.

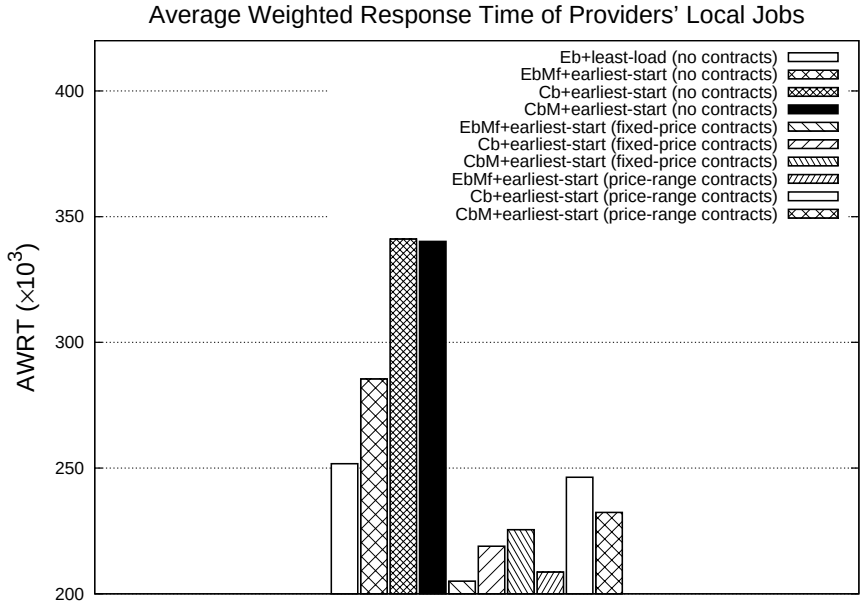

Fig. 6. Average Weighted Response Time (AWRT) of providers' jobs.

Figure 7 presents the percentage from the load from each Grid migrated to other Grids when providers send availability information every 12 hours. A previous investigation [9] 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 $k = 10\%$, Grids do not redirect more load in all the cases. For example, Figure 7 shows that when providers use conservative backfilling without multiple partitions, DAS-2 and AuverGrid in fact redirect less load. The investigation of the impact of different price ranges on provisioning under the all the policies described here is not in the scope of this paper.

Second Experiment: The second experiment performed evaluates the AWRT of Grid jobs in three situations wherein the providers send the availability information to the gateway firstly every 24 hours, secondly every 12 and finally every 6 hours. Table 2 shows the AWRT of Grid jobs per Grid under each scenario. In our previous study [9], we noticed that AuverGrid has a higher load than DAS-2 and Grid'5000. The table shows that Grids with a low utilisation do not have a decrease in the AWRT of their Grid users' applications. In fact, we can notice that in price range contracts, the AWRT is worsened. In contrast, AuverGrid has a substantial reduction in the AWRT of its Grid jobs. We can conclude that in terms of improving the AWRT, the peering of Grids with very different utilisation levels may not benefit the under-utilised Grids. However, as presented next, the Grids have economic benefits. In addition, the mechanism achieves its goal of redirecting requests from a Grid with high utilisation to others with lower utilisation levels as shown in Figure 7.

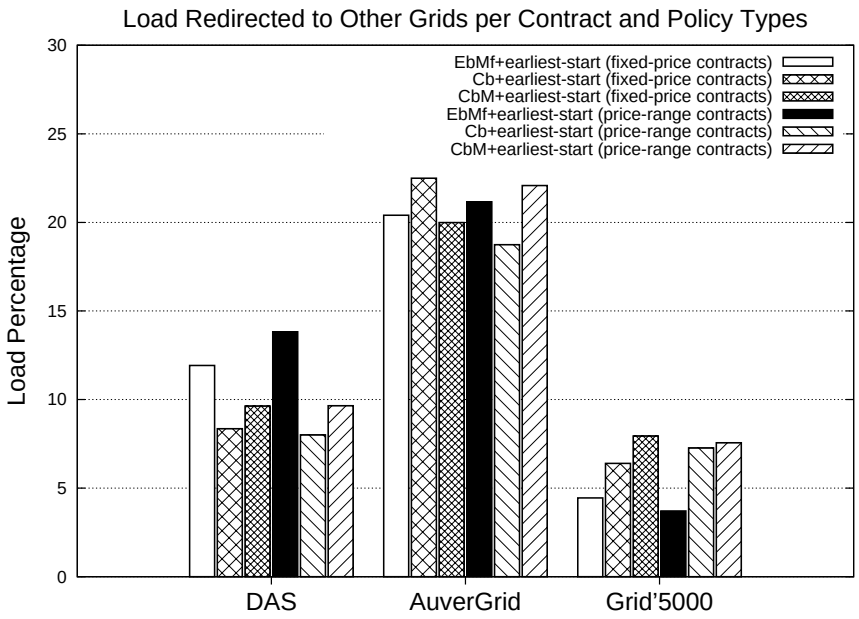

Fig. 7. Percentage of load generated by each Grid that was redirected to other Grids.

The experiments show that load management across Grids through resource exchange considering the economic compensation of resource providers is possible. The amount of load migrated shows that Grids balance their load and redirect requests thus minimising the costs with resource usage. The allocation policies allow gateways to make decisions on resources provided to peering Grids.

During the second experiment, we also evaluated the number of currency units spent by each IGG to acquire resources either from its providers or from other IGGs when contracts exist. Table 3 summarises the results. It also presents the amount of currency received by each IGG from other IGGs. The amount received by one IGG does not include the quantity paid by its users to execute their applications. Although the load redirection mechanism redirects requests based on their marginal cost, it does not provide a cost minimisation approach. That is, the marginal cost is calculated once a schedule for the job is found or a ticket is created. To schedule the job, the gateway uses an earliest start approach without searching for the cheapest resource. Despite this limitation, the results presented in Table 3 corroborate those from Table 2. AuverGrid has a reduction in the amount of currency spent acquiring resources when contracts are established. For example, when providers send the availability information every 24 hours, the amounts spent by AuverGrid for EbMf, Cb and CbM are 388175, 389507 and 385897 respectively. If Grid establish fixed-price contracts, AuverGrid's expenditure with resources for EbMf, Cb and CbM decrease, respectively, to 369901, 366215 and 363983. DAS-2 and Grid'5000 on the other hand, tend to spend more acquiring resources than if peering did not exist. However, they have an increase in the amount earned from other Grids.

As example, DAS-2 spends an amount of 28005, but receives 72102 from other Grids. Even though as of writing we are investigating further the advantages of peering for each Grid, we can conclude that Grids benefit from peering even when they pay a higher price for resources because the amount earned from other Grids generally contribute in reducing the overall expenditure with resources.

7 Conclusions and Future Work

This paper presents the performance evaluation of policies for resource provisioning across Grids and it demonstrates how Grids can redirect requests to other Grids during periods of peak demand. We have presented simulation results that demonstrate that the mechanism and policies are effective to redirect requests across Grids leading to a reduction in the Average Weighted Response Time (AWRT) of Grid applications. The interGrid load sharing mechanism provides economic incentives to Grids to redirect or accept requests from other Grids. Although underloaded Grids had the AWRT of their user applications increased, they had economic benefits given by the amount of currency received from other Grids. The experiments demonstrate that, despite the imprecise resource availability information given by providers, the load management across Grids through resource exchange is possible while accounting for the economic compensation of resource providers. [4]

We plan to investigate how IGGs can co-ordinate resource provisioning via shared spaces implemented atop Distributed Hash Tables (DHT) or other P2P systems. We are extending the mechanism by providing means for Grids to redirect requests across several Grids (i.e. it will support transitive relationships between the Grids in the contract network). Future investigations also include more sophisticated resource provisioning policies for the gateways, specially for handling advance reservation requests, more sophisticated load forecasting techniques and the impact of varying price-range contracts on provisioning.

Acknowledgements

We thank Marco Netto, Sungjin Choi, Mukaddim Pathan and Alexandre di Costanzo from the University of Melbourne for sharing their thoughts on the topic. We are grateful to Dr. Franck Cappello, Dr. Olivier Richard, Dr. Emmanuel Medernach and the Grid Workloads Archive group for making the

[4] We are currently implementing the proposed architecture. Software requirement specifications and other details are available at: http://www.gridbus.org/intergrid/

Grid workload traces available. This work is supported by DEST and ARC Project grants. Marcos' PhD research is partially supported by National ICT Australia (NICTA).

References

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

[2] T. Dunning, R. Nandkumar, International cyberinfrastructure: Activities around the globe, Cyberinfrastructure Technology Watch Quarterly 2 (1).
URL `http://www.ctwatch.org/quarterly/articles/2006/02`

[3] K. Miura, Overview of Japanese science Grid project NAREGI, Progress in Informatics (2006) 67–75.

[4] Open Science Grid, http://www.opensciencegrid.org (2005).

[5] L. Peterson, S. Muir, T. Roscoe, A. Klingaman, PlanetLab architecture: An overview, Tech. Rep. PDN-06-031, PlanetLab Consortium, Princeton, USA (May 2006).

[6] I. Foster, C. Kesselman, S. Tuecke, The anatomy of the Grid: Enabling scalable virtual organizations, International Journal of Supercomputer Applications 15 (3) (2001) 200–222.

[7] Grid Interoperability Now Community Group (GIN-CG), http://forge.ogf.org/sf/projects/gin (2006).
URL `http://forge.ogf.org/sf/projects/gin`

[8] A. Iosup, D. H. J. Epema, T. Tannenbaum, M. Farrellee, M. Livny, Inter-operating Grids through delegated matchmaking, in: 2007 ACM/IEEE Conference on Supercomputing (SC 2007), Reno, USA, 2007.

[9] M. D. de Assunção, R. Buyya, A resource exchange mechanism for InterGrid load management, Technical report, Grid Computing and Distributed Systems (GRIDS) Laboratory, The University of Melbourne, Australia (April 2008).

[10] OpenPBS: The portable batch system software, Veridian Systems, Inc., Mountain View, CA (2005).
URL `http://www.openpbs.org/scheduler.html`

[11] The Condor Project homepage, http://www.cs.wisc.edu/condor/ (2005).

[12] R. Buyya, D. Abramson, J. Giddy, 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), Beijing, China, 2000, pp. 283–289.

[13] S. Venugopal, R. Buyya, L. Winton, A grid service broker for scheduling e-science applications on global data grids: Research articles, Concurrency and Computation: Practice and Experience (CCPE) 18 (6) (2006) 685–699.

[14] G. Singh, C. Kesselman, E. Deelman, 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), ACM Press, Monterey, California, USA, 2007, pp. 117–126.

[15] M. D. de Assunção, R. Buyya, Performance analysis of multiple site resource provisioning: Effects of the precision of availability information, Technical report, Grid Computing and Distributed Systems (GRIDS) Laboratory, The University of Melbourne, Australia (May 2008).

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

[17] L. Peterson, J. Wroclawski, Overview of the GENI architecture, GENI Design Document GDD-06-11, GENI: Global Environment for Network Innovations (January 2007).
URL `http://www.geni.net/GDD/GDD-06-11.pdf`

[18] D. Irwin, J. Chase, L. Grit, A. Yumerefendi, D. Becker, K. G. Yocum, Sharing networked resources with brokered leases, in: USENIX Annual Technical Conference, Boston, Massachusetts, 2006, pp. 199–212.

[19] L. Ramakrishnan, D. Irwin, L. Grit, A. Yumerefendi, A. Iamnitchi, J. Chase, Toward a doctrine of containment: Grid hosting with adaptive resource control, in: 2006 ACM/IEEE Conference on Supercomputing (SC 2006), ACM Press, New York, NY, USA, 2006, p. 101.

[20] L. Grit, D. Inwin, A. Yumerefendi, J. Chase, Virtual machine hosting for networked clusters: Building the foundations for 'autonomic' orchestration, in: 1st International Workshop on Virtualization Technology in Distributed Computing (VTDC 2006), Tampa, Florida, 2006.

[21] R. Ranjan, A. Harwood, R. Buyya, SLA-based coordinated superscheduling scheme for computational Grids, in: IEEE International Conference on Cluster Computing (Cluster 2006), Barcelona, Spain, 2006, pp. 1–8.

[22] M. Balazinska, H. Balakrishnan, M. Stonebraker, Contract-based load management in federated distributed systems, in: 1st Symposium on Networked Systems Design and Implementation (NSDI), USENIX, San Francisco, CA, 2004, pp. 197–210.

[23] Y.-T. Wang, R. J. T. Morris, Load sharing in distributed systems, IEEE Transactions on Computers C-34 (3) (1985) 204–217.

[24] S. Surana, B. Godfrey, K. Lakshminarayanan, R. Karp, I. Stoica, Load balancing in dynamic structured peer-to-peer systems, Performance Evaluation 63 (3) (2006) 217–240.

[25] G. Singh, C. Kesselman, E. Deelman, Application-level resource provisioning on the grid, in: 2nd IEEE International Conference on e-Science and Grid Computing (e-Science 2006), Amsterdam, The Netherlands, 2006, pp. 83–83.

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

[27] D. B. Jackson, Q. Snell, M. J. Clement, Core algorithms of the Maui scheduler, in: 7th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '01), LNCS, Springer-Verlag, London, UK, 2001, pp. 87–102.

[28] B. G. Lawson, E. Smirni, 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, Springer-Verlag, London, UK, 2002, pp. 72–87.

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

[30] Y. Fu, J. Chase, B. Chun, S. Schwab, A. Vahdat, SHARP: An architecture for secure resource peering, in: 19th ACM Symposium on Operating Systems Principles (SOSP 2003), New York, NY, USA, 2003, pp. 133–148.

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

[32] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, Introduction to Algorithms, 2nd Edition, MIT Press / McGraw-Hill, Cambridge, Massachusetts, 2001.

[33] The distributed ASCI supercomputer 2 (DAS-2), Dutch University Backbone (2006).

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

[35] Conseil Régional Auvergne, AuverGrid, http://www.auvergrid.fr (2007).

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

[37] J. E. Hanke, A. G. Reitsch, Business Forecasting, 5th Edition, Prentice-Hall, Inc., Englewood Cliffs, USA, 1995.

[38] C. Grimme, J. Lepping, A. Papaspyrou, Prospects of collaboration between compute providers by means of job interchange, in: Job Scheduling Strategies for Parallel Processing, Vol. 4942 of Lecture Notes in Computer Science, Springer, Berlin / Heidelberg, 2008, pp. 132–151.

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

Tables Referred in the Manuscript

Table 1. Description of the parameters used in the experiments.

Parameter	**Description**
Number of Grids	3
Contract Topology	all-to-all (see Figure 4)
Cost of a resource unit	0.90–1.00
Load threshold (%)	99
Value of k (%)	10
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

Table 2. The 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	27691	32653	32865	30311	39518	34236	30079	35575	36467
AuverGrid	853264	1009034	1009370	453125	432218	463038	520081	465852	387086
Grid´5000	173757	180541	182793	176162	179872	183429	175015	180846	185185
	Providers sending availability information every 12 hours								
DAS	25858	28876	28969	28082	34059	33243	28096	33287	33915
AuverGrid	979386	1002692	999848	556294	470293	497227	551942	549930	497493
Grid´5000	178681	177033	176717	172410	178709	175760	177335	178220	178522
	Providers sending availability information every 6 hours								
DAS	25518	27044	27152	29022	29374	33869	42009	28889	29762
AuverGrid	980650	996009	997119	571826	524826	449851	544159	533275	535906
Grid´5000	180664	172878	175510	186064	171690	174887	174007	175910	174124

Table 3. Number of currency units[a] received by an IGG from other IGGs (i.e. received)[b] and spent by the IGG acquiring resources from its providers or from other IGGs when contracts are enabled (i.e. spent).

	Providers sending availability information every 24 hours														
	No contracts			Fixed-price contracts						Price-range contracts					
	EbMf	Cb	CbM	EbMf		Cb		CbM		EbMf		Cb		CbM	
Grid	spent	spent	spent	received	spent	received	spent	received	spent	received	spent	received	spent	received	spent
DAS	30810	26828	26614	79475	32259	78335	28835	72102	28005	70054	32138	81492	28248	90771	28646
AuverGrid	388175	389507	385897	114	369901	81	366215	2	363983	12	377959	99	367183	132	363506
Grid´5000	181275	157093	159900	25970	181671	23566	156577	22864	157800	25132	181651	21680	157186	26143	157660
	Providers sending availability information every 12 hours														
DAS	31120	27040	26662	70361	32303	73879	29042	64597	28846	66980	32476	58682	28808	66404	29045
AuverGrid	378825	388784	387525	109	378385	70	372724	2	373540	250	377643	0	375728	75	370316
Grid´5000	181462	160970	159327	12892	179938	13761	160540	16642	163184	17419	181947	14718	160057	20196	161285
	Providers sending availability information every 6 hours														
DAS	30812	27491	27496	68031	32195	65795	29150	72158	30325	77961	32444	62108	28993	65491	29402
AuverGrid	389592	390215	388963	231	377703	170	374411	431	376042	631	377856	205	374673	371	375395
Grid´5000	179335	164163	163665	16803	180770	18185	162988	11979	164263	14233	179944	18731	163626	13711	163153

[a] To minimise the space used by the table, the amounts spent and received where divided by 10,000.

[b] Please note that 'received' is the amount received by one IGG from others IGG and does not include the amount paid by the Grid users to execute their applications.

Captions of the Figures Used in the Manuscript

1 The provisioning scenario with multiple Grids considered in this work.

2 Free time slots (conservative backfilling).

3 Redirection negotiation.

4 Contract topology simulated.

5 Average Weighted Response Time (AWRT) of Grid jobs.

6 Average Weighted Response Time (AWRT) of providers' jobs.

7 Percentage of load generated by each Grid that was redirected to other Grids.