A Resource Exchange Mechanism for Peak Load Management in InterGrid Environments

Marcos Dias de Assunção and Rajkumar Buyya
Grid Computing and **D**istributed **S**ystems (GRIDS) Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
{marcosd, raj}@csse.unimelb.edu.au

Abstract

The potential of the Grid computing paradigm has led to the creation of several regional and national Grid environments. These environments are, however, like "Grid islands" with no resource sharing between them. We have presented an architecture to enable internetworking of Grids for resource sharing based on peering arrangements between Grids. This architecture and related mechanisms and policies are termed as the InterGrid. In this paper, we address the problem of resource exchange in InterGrid environments. Enabling resource exchange in InterGrids is a challenging task: a Grid can benefit from internetworking with other Grids but needs to respect its local user communities. The paper describes a peak load management mechanism and related policies for resource exchange between Grids using a resource share model. We demonstrate that gateways with provisioning rights over the resources of a Grid can interact with one another by using the proposed mechanism, enabling resource exchange while satisfying the requests of local user communities.

1. Introduction

The emergence of Grid computing has led to the creation of several Grid-based resource sharing networks (e.g. TeraGrid [6, 14], Naregi [13] and Open Science Grid [1]) currently utilised by multiple scientific communities for varying purposes. Resource sharing in these networks is generally performed through collaborations among groups of individuals, organisations and resources formed to tackle a common problem or achieve a common goal; these collaborations are widely known as Virtual Organisations (VOs) [8]. Although appealing, these efforts have resulted in disjointed communities or what we term "Grid islands", with no resource sharing between them.

Our previous work introduces an architecture based on gateways that mediate the resource exchange between Grids and allow participants to allocate resources from different Grids in a seamless manner [7]. This architecture and related mechanisms and policies are termed as the InterGrid. The InterGrid is inspired in the way that Internet Service Providers (ISPs) establish agreements

with one another in the Internet. The Internet is composed of competing ISPs that agree to allow traffic into one another's networks providing their customers with connectivity to the entire Internet. These agreements between ISPs are commonly termed as peering and transit arrangements [12]. We have advocated that internetworking of Grids through peering arrangements that enable Grids to exchange resources with one another is important for the evolution of Grid computing.

Additionally, the Grid Interoperability Now - Community Group (GIN-CG) [2] has been working on providing interoperability between Grids by developing components and adapters that enable secure job submission, data transfers and information queries. Although GIN-CG's efforts are relevant, its members also highlight the need for common allocation and brokering of resources between Grids. However, these challenges are not yet addressed in present research[1]. Therefore, we have been investigating mechanisms and policies for resource allocation and exchange in the InterGrid, which can be applicable to scenarios such as the GIN Grids.

Resource exchange across Grids is a challenging task because of the autonomy regarding capacity planning and provisioning of resources to user communities within each Grid. There is contention for resources and dynamicity regarding the shares supplied by resource providers within each participating Grid. Moreover, there can be benefits for a Grid to provide spare capacity to peering Grids, possibly in return to regular payments, and to acquire resources from peering Grids to serve occasional peaks in the demand of its user communities. This approach can reduce the costs incurred by over-provisioning. Thus, the main problems of resource exchange in the InterGrid are how a Grid can (i) meet the demand of local user communities by providing the resources required; (ii) coordinate with peering Grids in acquiring additional resources when its users demand; and (iii) provide spare resources to other Grids in return to payments when user communities in the other Grids require them.

The main contributions of this paper are (a) to propose a mechanism for Grid internetworking based on a contract network between the peering Grids that enables a Grid to offload requests to another Grid; (b) provide policies for inter-Grid resource exchange to enable the redirection of resource requests to a peering Grid and the acceptance of resource requests from other Grids.

2. Related Work

The two major related initiatives that have recently emerged for enabling internetworking of resource sharing networks are the Global Environment for Network Innovations (GENI) [16] and GIN-CG under the Open Grid Forum [2].

The PlanetLab architecture [15] has been evolving to allow the federation of autonomous PlanetLabs controlled by different organisations [16]. PlanetLab currently provides a global infrastructure and mechanisms that allow the creation of slices; on top of these slices, varying distributed applications can run. The interest in federation, however, will eventually lead to the creation of smaller autonomous PlanetLabs. The use of virtualisation technology and federation of autonomous utility infrastructures is also a scenario considered by GENI. The mechanisms and policies that we are investigating can enable the interaction between dispersed PlanetLab's with disparate slice authorities.

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

Several Grid infrastructures have been created over the last years [1, 6, 13, 14]. Additionally, GIN-CG [2] has been working on leveraging community efforts to address problems regarding security, standard job submission, data management and information services. GIN-CG's members highlight the importance of policies for inter-Grid brokering and resource management. We consider these efforts of utmost importance and aim to build on these to investigate resource management in InterGrids. We follow a slightly different resource model and consider that Grids are aggregates of resource providers that can be autonomous in nature. Enabling resource exchange amongst these aggregates is more challenging than resource allocation within an individual aggregate. Although in its infancy, our work intends to develop mechanisms that can bridge this gap.

Shirako [11, 17] offers an architecture for resource management based on the abstraction of resource leasing. Sites delegate limited power to allocate their resources by registering their resource offerings with brokers. Guest applications can acquire resources from brokers by leasing them for a specified time. Our work shares some concepts with Shirako such as the delegation of provisioning rights. We focus on the policies for resource exchange amongst InterGrid Gateways that can also be suitable to the interaction amongst brokers in systems like Shirako. This has not been explored in depth in Shirako thus far.

Grit *et al.* [10] investigate the number of VM migrations incurred when a broker and provider sites use either conflicting or synchronised policies for resource provisioning and VM placement. In their allocation model, sites delegate rights to provision host servers to a broker. The broker determines how to allocate resource slivers from its logical inventory to serve requests. Sites have their policies to map VMs to physical resources. Clients may want to resize their slivers due to load changes. It is shown that when providers and the broker use conflicting policies, the number of migrations can be high (i.e. the broker assumes that VMs can be migrated or that providers assume that slivers can grow; and providers try to minimise the number of resources used). We are currently researching the interaction and contracts between sites and gateways, the provisioning of resources to applications by sites in the presence of intermediate entities like IGGs and further the provision by the IGGs themselves under imprecise information provided by participating sites.

The creation of execution environments has been considered by others. For example, there are initiatives aiming at the creation and management of execution environments [18]. Such technology can be used to enable the creation of DVEs spanning multiple Grids and the peering arrangements between Grids. We aim to enable VOs to evolve to multiple Grids in a transparent way through the peering arrangements between IGGs.

Balazinska *et al.* [3] propose 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 in the above mechanism and extend it to support the exchange of slots amongst Grids in the proposed architecture.

3. Resource Exchange in the InterGrid

This section provides an overview of the InterGrid architecture and its main components followed by a description of the resource exchange scenario.

3.1. Overview of the InterGrid

Figure 2 provides an overview of the InterGrid architecture for enabling internetworking of Grids. A detailed discussion has already been presented elsewhere [7]. The abstraction of containers is used for resource allocation across Grids. This way, the resources exchanged between Grids can be physical or virtual resources such as Virtual Machines (VMs). We use the term "slot" to denote a VM or a physical resource on which the services and applications required by users can be deployed.

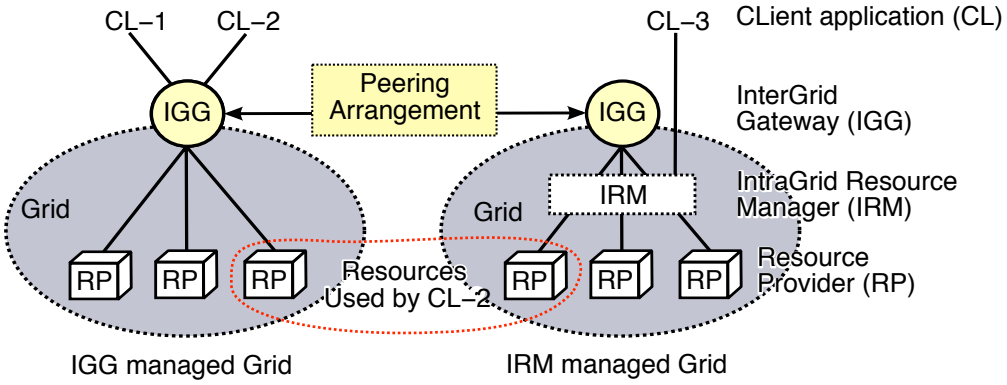

Figure 1. The main components of the InterGrid architecture.

Resource Providers (RPs) contribute a share of computational resource, storage resource, networks, application services or other types of resources to a Grid in return for regular payments. The RP advertises its resources (i.e. slots) in the registry provided by the IntraGrid Resource Manager (IRM). The advertisement is made through a "slot assertion", which is a delegation of the provisioning rights over a set of slots or resource shares. This delegation can use a secure protocol such as SHARP [9]. The IRM can manage different shares of resources that have been allocated by the RPs to the Grid. The IRM is not an essential part of the architecture as it may represent an existing resource manager deployed in an individual Grid. The IRM can assign provisioning rights to the InterGrid Gateway (IGG) over part of the resources provided by RPs. We also consider the case where RPs can assign provisioning rights directly to the IGG.

A Grid can have peering arrangements with other Grids through which they coordinate the use of resources of the InterGrid. These arrangements are managed by IGGs. An IGG is aware of the terms of the peering with other Grids; provides Grid selection capabilities by selecting a suitable 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 [4] and provide an overlay network comprising the resources allocated from the InterGrid. When a CL shows interest in obtaining a number of slots to deploy or execute an application, it sends "slot requests" to the IRM (i.e. when the Grid has its own IRM) or to the IGG based on the demands of the application. The IRM can provide all or part of the required slots based on its provisioning policies. If the individual Grid cannot provide the required slots, then the IRM forwards part of the slot requests to the IGG. The IGG selects a peering Grid from which the slots can be allocated based on the peering agree-

ments and the policies in place. Once the slots have been acquired, the CL is given a permission to use them.

3.2. The Resource Exchange Scenario

The InterGrid is defined by Equation (1). The InterGrid comprises a set *G* of participating Grids. Each Grid $g_i \in G$ has a set CL_i of clients running applications that require resources and a set of resource providers RP_i that provide shares of resources to Grid g_i. Grid g_i is at a given time composed of a set of resources, which is the union of the resources contributed by the each resource provider *rp* $\in RP_i$. CL and RP represent the set of clients and providers in the InterGrid scenario respectively.

$$g_i \in G : g_i \Rightarrow \bigcup_{cl_j}^{cl_n} \textbf{ and } \bigcup_{rp_j}^{rp_n} ;\ rp_j \in RP_i \subset RP \textbf{ and } cl_j \in CL_i \subset CL \tag{1}$$

Here, the current resource allocation at Grid g_i is represented by A_i. K represents the types of resources that Grids can acquire from one another. Clients issue resource requests according to their demands requiring resources whose types are specified in *K*. A resource request can be fragmented or continuous. If the request is continuous, then it has to be served with resources from a single resource provider. When a request *r* can be fragmented, then the resources used to serve *r* can be provided by multiple resource providers. In this work, we consider that requests cannot be fragmented. We also consider that a request for additional resources corresponds to the arrival of a new request, whereas the release of resources by a client represents the termination of a request.

The peering agreements between Grids define, for example, (i) how resource requests are redirected from one Grid to another; (ii) how to balance resource requests; and (iii) the resource types for resource exchange between Grids. Therefore, the goal of a participating Grid g_i is to (i) serve its clients or user communities by providing allocations A_i that assign resources that satisfy their QoS requirements; (ii) offer spare resource to peering Grids under some compensation; and (iii) acquire resource from other Grids to satisfy its clients under conditions of peak load.

3.3. Assertions, Requests and Permissions

Each resource provider rp_j has a Resource Manager (RM) responsible for the resource provisioning and allocation at the RP. RPs agree to provide their resources under the terms stated in contracts such as Service Level Agreements (SLAs). The resources provided to a Grid under these agreements are published as slot assertions. These assertions and a description of the policies utilised by the RPs are kept by the IRM. The IRM can delegate part or the full set of slot assertions to the IGG.

A slot assertion for rp_j, sa_j, includes information about the slots available at RP *j*, their characteristics and when they will be available. Therefore, $sa_j = (S_j, t_j, \Delta t_j)$ where S_j is a slot set containing the list of slots available at rp_j; t_j represents the start time when the slots will be available; and Δt_j is the amount of time over which the slots will be available.

A slot set can contain information regarding the number of CPUs, the CPUs' architecture and speed, amount of memory and disk space. Here, we use the term *type* to indicate a virtual or physical resource with a given configuration. Thus, a slot set S_j is represented by $S_j = (s_j, x_j)$, where s_j represents the number of slots that compose the set; x_j is the type of the slots in the set.

The requests received by an IRM or IGG contain a description of the required slots. A slot request made by client cl_j is defined as $sr_j = (g_j, S_j, t_j, \Delta t_j, rt_j)$, where g_j is the slot request group (i.e. the request belongs to a group of requests identified by g_j); S_j is the required slot set; t_j is the start time when the client expects to use the slots; Δt_j is the amount of time for which the slot set will be required; rt_j is the type of request. There can be *RT* request types, so rt_j = *(1 or 2 or ... or RT)*. For example, a request can be flexible regarding the time, meaning that the resources can be provided at a time different from that originally stipulated. Here, we consider two types of requests, namely immediate start and advance reservations.

4. The Peak Load Management Mechanism

In this section we describe the mechanism to balance the load imposed by slot requests in the InterGrid. The mechanism is derived from Medusa [3], but differs in terms of the resource selection and request redirection policies.

The offloading of allocation requests is enabled between Grids that have negotiated contracts, at within the contracted price range. The offloading occurs when a Grid forwards requests to another because the cost of fulfilling the requests is higher than the amount that it would have to pay to the other Grid to serve them. 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 (2), where A_i corresponds to current allocations of Grid g_i and SS_i to the number of slots in Grid g_i, as described beforehand.

$$\forall A_i, \text{cost}_i(A_i, SS_i) \rightarrow \Re \tag{2}$$

Therefore, the cost given by $cost_i(A_i, SS_i)$ depends on the number of slots allocated by the requests and the slot sets available at the Grid. Although each Grid could have its own cost function, in this work, the participating Grids utilise a quadractic cost function. Grid g_i has a load threshold, by crossing which Grid g_i considers itself overloaded. Request movements are based on the per slot marginal cost, $mc_i : (u, A_i, SS_i) \rightarrow \Re$ which is the increment in the cost for Grid g_i in accepting to provide the slots required by request u given its current allocations A_i and slots SS_i. If the request u requires one slot of a type x specified in *K* for a time unit t, so the marginal cost for one slot of that type is given by $m_x = mc_x(u, A_{i,x}, SS_{i,x})$, where $A_{i,x}$ is the current allocation of slots of type x and $SS_{i,x}$ is the total number of slots of type x available.

4.1. Contract Types

A contract $C_{i,j}$ between IGG_i and IGG_j has a price range $PR_x(C_{i,j}) : [min_x(C_{i,j}), max_x(C_{i,j})]$, which is the price paid by IGG_i for a slot of type x allocated from Grid IGG_j over a time unit t. IGG_i can have several contracts with other Grids. The set of contracts of IGG_i is represented by CS_i. C_i is the number of contracts of IGG_i. During periods of peak load, IGG_i can redirect requests to IGG_j if both have a contract. Based on the current load levels, they agree on a final price $P_x(C_{i,j})$ for a slot of type x within $PR_x(C_{i,j})$. The number of slots allocated by Grid g_i from Grid g_j is represented by $\Upsilon_{i,j}$. IGG_i pays the amount equivalent to $(P_x(C_{i,j}) * \Upsilon_{i,j} * \Delta t)$ to IGG_j, where Δt corresponds to the number of time units over which Grid g_i utilises resources from Grid g_j.

We support two kinds of contracts, namely fixed price (i.e. $PR_x(C_{i,j}) : [fp_x, fp_x]$ where fp is the fixed price) and price range contracts (i.e. $PR_x(C_{i,j}) : [fp_x - \Delta_x, fp_x]$). In the case of price range contracts, participating Grids have to negotiate the final price at runtime. As discussed by Balazinska *et al.* [3], a load management mechanism based on fixed price contracts may present disadvantages in some cases. For example, it lacks on flexibility when a Grid needs to offload requests, because it would offload them only if the price of a contract in its contract set is exactly lower than its marginal cost. Moreover, a Grid will accept a request only if the price of the request is higher than the Grid's marginal cost.

Similarly to Balazinska *et al.* [3], we define the price range for a slot of type *x* considering the decrease of *k* slots from the allocations *A*. Let *u* be a request that requires 1 slot of type *x*; the decrease in the per-slot marginal cost due to removing *k* slots from the Grid's *A* is represented by δk, which is defined by (3).

$$\delta_k(A) = mc(u, A - u) - mc(u, A - (k + 1)u) \tag{3}$$

δk is the approximate difference in the cost function gradient evaluated at the load level including and excluding the *k* slots of type *x*. Given a contract with fixed price *fp*, A_x is the maximum set of requests that an IGG can accept before its per slot marginal cost exceeds fp. Therefore, A_x satisfies: $mc(u, A_x - u) \leq size(u) * fp$ and $mc(u, A_x) > size(u) * fp$. In order to estimate the price range for a slot in the contracts in our experiments, we let A_x be the allocations *A* at a given utilisation rate and *u* be a request of size 1 and $\Delta_x = \delta_k$. We evaluate different values for A_x and *k*.

Considering that two Grids g_i and g_j that have a contract specifying a price range. Consider that g_i sends an offer to g_j when g_i's marginal cost is higher than the minimum price of the contract with g_j. Grid g_j in turn sends a counter-offer if its marginal cost is smaller than the maximum price specified in the contract. Grid g_j's counter-offer contains the price of the offer sent by g_i if g_j's marginal cost is smaller than the offered price. The price in the counter-offer is g_j's marginal cost if the latter is between the range specified in the contract. Grid g_i accepts the counter-offer if the price given by g_j is smaller than its marginal cost.

4.2. Request Balancing Algorithms

In this section we describe a set of algorithms for the proposed peak load management mechanism. These algorithms define how an IGG offloads slot requests to peering Grids considering a contract network and how it accepts requests from peering Grids. Algorithm 1 describes how an IGG_i allocates slots to the requests made by its user communities (i.e. resource provisioning) and how it offloads the requests to another Grid.

During a given number of time units Ω, or while requests previously submitted are being treated (i.e. requests are being either served or moved), IGG_i stores the requests received in the list OR_i (lines 2 to 6). After Ω, IGG_i orders the contracts in ascending order of price and for each contract $C_{i,j}$. IGG_i evaluates whether there are requests that can be redirected to the peering IGG_j. From line 11 to 13, IGG_i verifies if its marginal cost is greater than the price that it would pay to the IGG_j for serving the request. If the marginal cost is greater, then the IGG_i adds the request to the *requestset*. Otherwise, if there are resources available, the request is treated locally (lines 14 to 17). From line 20 to 32, IGG_i creates an offer, sends it to IGG_j and waits for a reply. If the IGG_j

accepts the offer, then the IGG_i informs the CLs about the acceptance. From line 34 to 42 the IGG_i performs a last iteration to allocate slots to requests that have not being accepted by other Grids.

Algorithm 2 specifies the steps that the IGG_j follows to accept or reject offers sent by peering Grids. IGG_j inserts the offers in a list during a time interval (lines 2 to 6). After the interval lapsed, IGG_j sorts the offers by decreasing order of price. In addition, IGG_j maintains a list of offers with which it has agreed or to which it will send a counter-offer (i.e. *potentialset*). For each request in an offer, IGG_j verifies if the marginal cost of accepting the request is smaller than the amount that it is going to receive to serve the offer, given its current load and the *potentialset*. After that, if the marginal cost is smaller, IGG_j adds the request to the set of requests that will be accepted. If the set of requests accepted is not empty, then IGG_j sends a response and updates its *potentialset* (lines 22 to 25).

We use a best-fit policy for resource selection by an IGG. Algorithm 3 shows the best-fit algorithm for *select(request r)*. In this algorithm, IGG_j orders the slot assertions in increasing order of slots available and tries to fit the request in the smallest slot assertion able to provide the slots required. Moreover, a Grid may provide a large share of its resources to peering Grids, which may decrease the request acceptance rate of its local user communities. To minimise this impact, we introduce a Peering Threshold (PT) that defines the number of resources that an IGG can provide to its peering IGGs. When an IGG selects resources to peering Grids, it verifies if the resulting allocations will not violate PT.

5. Performance Evaluation

5.1. Experimental Scenario

Simulation tool and cost functions: The evaluation of the proposed mechanism is performed through simulation by using GridSim Toolkit 4.0 [5]. We extend GridSim to support the functionality of slots and IGGs. We consider that IGGs use a quadratic cost function given by $\alpha(A_x)+\beta(A_x)^2$ where α is the average cost of a slot of type *x* for a time unit (i.e. it is the price paid by the IGG to resource providers), A_x is the load (i.e. the number of slots of type *x* allocated) and β is a small constant that indicates how steep the cost curve is as the Grid approaches full utilisation.

Workload and requests: The workload of clients is modelled using the San Diego Supercomputer Center (SDSC) SP2 log[2]. The log is divided into 15-day intervals one interval is assigned to each client. Requests that require less than 5 CPUs and whose duration is smaller than 2 hours long have been removed from the original log, because we believe that these do not reflect the current requests for initialising VMs on remote sites. We scale the trace according to the maximum capacity of resource providers of the Grid to which each client belongs so the resource requests issued by a client can be served by an individual resource provider. The experiments are performed with 10 different sets of workloads. For each set, 10 executions are carried out and the best and worst results removed. The results presented are averages of the experiments with 10 sets of workloads.

There are two types of requests, namely Immediate Start Requests (ISRs) and Advance Reservations (ARs). When an IGG receives an ISR, it verifies whether it has free slots over the duration of the ISR starting from the current time. ARs on the other hand, are requests for slots for a given

[2]We use the version 3.1 of the SDSC SP2 log, available at: http://www.cs.huji.ac.il/labs/parallel/workload/

Algorithm 1 Provisioning and request redirection.

loop
  for Ω time units or **while** ($movement = true$) **do**
    for each request *r* received **do**
      $OR_i \leftarrow OR_i \cup \{r\}$
    end for
  end for
  sort CS_i on $P(C_{i,j})$ ascending
  for each contract $C_{i,j} \in CS_i$ **do**
    $requestset \leftarrow \emptyset$
    for each request $u \in OR_i$ **do**
      if $mc(u, A_i) > size(u) * P(C_{i,j})$ **then**
        $requestset \leftarrow requestset \cup \{u\}$
      else
        $selected \leftarrow select(u)$ /* find slots in SS_i; see Algorithm 3 */
        $send(CL_i, accept, u, selected)$
        $A_i \leftarrow A_i \cup \{selected\}$
        $OR_i \leftarrow OR_i - \{u\}$
      end if
    end for
    if $requestset \neq \emptyset$ **then**
      offer $\leftarrow P(C_{i,j}), requestset$
      $(resp, acceptset, assigned) \leftarrow send(IGG_j$,*offer*$)$
      $movement \leftarrow true$ /* Accept set contains the resources given by IGG_j*/
      if $resp = accept$ and $acceptset \neq \emptyset$ **then**
        $confirm(IGG_j, P(C_{i,j}), acceptset)$
      end if
      for each request accepted $a \in accepset$ **do**
        $selected \leftarrow assigned\_slots(assigned, a)$
        $send(CL_i, accept, a, selected)$
        $OR_i \leftarrow OR_i - \{a\}$
      end for
    end if
  end for
  if $OR_i \neq \emptyset$ **then**
    for each request $u \in OR_i$ **do**
      $selected \leftarrow select(u)$ **/* find slots in SS_i */**
      if $selected \neq \emptyset$ **then**
        $send(CL_i, accept, u, selected)$
      else
        $send(CL_i, reject, u)$
      end if
    end for
  end if
end loop

Algorithm 2 Offer evaluation.

loop
  for Ω time units or **while** $(movement = true)$ **do**
    for each offer o received **do**
      $offers_j \leftarrow OR_i \cup \{r\}$
    end for
  end for
  sort $offers_j$ on price descending
  $potentialset \leftarrow \emptyset$
  for each offer $o_i \in offers_j$ **do**
    $acceptset \leftarrow \emptyset$
    $assigned \leftarrow \emptyset$
    for each request $u \in o_i$ **do**
      $load \leftarrow A_j \cup potentialset \cup acceptset$
      if $mc(u, load) < size(u) * P(o_i)$ **then**
        $selected \leftarrow select(u)$ /* find slots in SS_j; see Algorithm 3 */
        if $selected \neq \emptyset$ **then**
          $acceptset \leftarrow acceptset \cup \{u\}$
          $assigned \leftarrow assigned \cup selected$
        end if
      end if
    end for
    if $acceptset \neq \emptyset$ **then**
      $potentialset \leftarrow potentialset \cup acceptset$ /* Update the set of potentially accepted requests */
      $resp \leftarrow (accept, acceptset, assigned)$
      $movement \leftarrow true$
    else
      $resp \leftarrow (reject, \emptyset, \emptyset)$
    end if
  end for
end loop

time in the future. ARs can be made up to 12 hours before their start time.

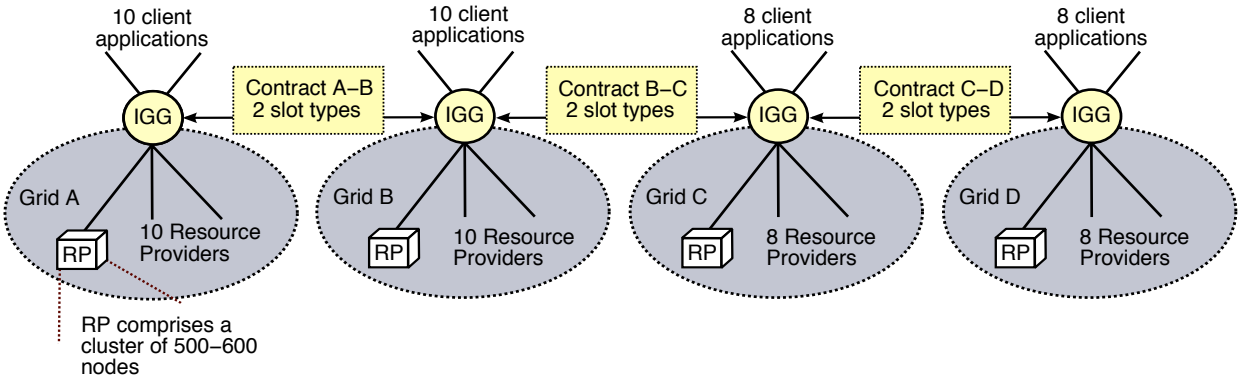

Figure 2. InterGrid environment simulated.

Algorithm 3 Selection of slots from SS_i.

```
1: reqslots ← get the slots required by r
2: type ← get the type of slots of reqslots
3: st_r ← get the start time of r
4: Δt_r ← duration of request r
5: SS_type ← the slot assertions of type type in SS_i
6: selected ← ∅
7: found ← ∅
8: sort SS_type on capacity and duration ascending
9: for each assertion sa_j ∈ SS_type or while (found = false) do
10:    st_j ← get the start time of sa_j
11:    Δt_j ← get the duration of sa_j
12:    if st_j ≤ st_r and Δt_j ≥ Δr then
13:       avail_j ← n. of slots available in sa_j from st_r to Δt_r
14:       if reqslots ≤ avail_i then
15:          selected ← select reqslots from sa_j
16:          /* lock slots in sa_j until the confirmation of use */
17:          found ← true
18:       end if
19:    end if
20: end for
21: return selected
```

Evaluated scenario: Figure 2 presents the InterGrid environment simulated while Table 1 summarises the parameters. Grids exchange two types of slots. We specify that about 50% of the resource providers per Grid are dedicated. The resource providers that are not dedicated issue slot assertions with a duration uniformly distributed between 3 and 5 days. The time between two slot assertions issued by a resource provider is uniformly distributed between 0 and 6 hours. PT is 30%, which means that an IGG will not provide more than 30% of its resources to its peers at any time. We calculate the price for slots in the contracts between the Grids by assigning different values to *A* in equation (3). These values in fact correspond to A_x explained in Section 4.1. We perform experiments considering *A* equals to 99%, 95%, 90% and 85% of utilisation and with different values for *k* (i.e. 1%, 5%, 10% and 20% of the Grids' resources). For example, when *A*=99% and *k*=1%, the fixed price (fp) of a contract is equals the marginal cost of accepting a request of size 1 when the Grid is at 99% of utilisation. The price range contract has a maximum price of fp and a minimum price given by fp minus the difference between the marginal cost at 99% of utilisation and the the marginal cost at 98% of utilisation.

Performance metrics: We select two metrics, namely increase in request acceptance rate and percentage of the generated load that is redirected by the IGGs. The redirected load demonstrates the performance of the mechanism in terms of migration of peak loads; the increase in acceptance rate, on the other hand, demonstrates whether the IGG compromises local users by peering with other IGGs. We also compute the increase in utilisation for comparison against the migration of load.

Table 1. Evaluated scenario and parameters.

Parameter Description	Value
Number of types of slots exchanged	2
Number of dedicated RPs per Grid	50%
Mean duration of slot assertions	3-5 days
Mean time between slot assertions	0-6 hours
Peering Threshold (PT)	30%
Number of Grids	4
Number of RPs per Grid	8-10
Number of slots per RP	500-600
Number of clients (CL) per Grid	8-10
Proportion of AR requests	20% of the requests of each client
Contracts between Grids (A↔B represents a contract between Grids A and B)	(A↔B) (B↔C) (C↔D)

5.2. Experimental Results

The global increase in request acceptance rate under the different types of contracts is shown in Figure 3 (a-c). It represents the acceptance rate of all the requests made by users of all the Grids. The charts demonstrate that, overall, Grids benefit from peering with one another by increasing the request acceptance rate. The charts in Figure 3 (d-f) present the percentage from the total load that has been exchanged by the IGGs. We have noticed that IGGs can accept requests using price range contracts without heavily compromising the request acceptance rate when they maintain PT below 30%. When Grids define the maximum price for a slot as the marginal cost at 99% of utilisation (i.e. A_x equals to 99%), the overall acceptance rate is improved in almost all the cases. The acceptance is better when the contracts define a price range, which allows Grids to redirect more load.

When IGGs set the maximum price equals to the marginal cost at a low A_x and the price range is large, the increase in acceptance rate and the amount of load exchanged decrease. The reason for this behaviour is that overloaded IGGs tend to be more conservative in accepting load from other IGGs, even though they try to migrate load more easily. In other words, an IGG_i that needs to offload, will redirect requests willing to pay a price less or equals to the maximum price of a contract. If the marginal cost of the IGG considering to accept the load is slightly above the maximum price, it will not accept the load. In our experiments, this behaviour is reflected in the acceptance rate.

Table 2 presents the increase in the acceptance rate of the requests originated in each Grid and the amount of load migrated by the IGGs to other Grids. Grid A and D migrate smaller amounts of load as they have only one contract each. Grid C transfers the most of load to other Grids, so decreasing its resource utilisation. Even though Grids A and B do not have a substantial increase in the acceptance rate of the requests originated by their users, they increase their resource utilisation, without compromising the acceptance rate of the requests originated in their Grids. This leads to an increase in profit as they can receive a payment from the other Grids for the resources provided; however, we do not measure the profits of each Grid here. Overall, the algorithms achieve their

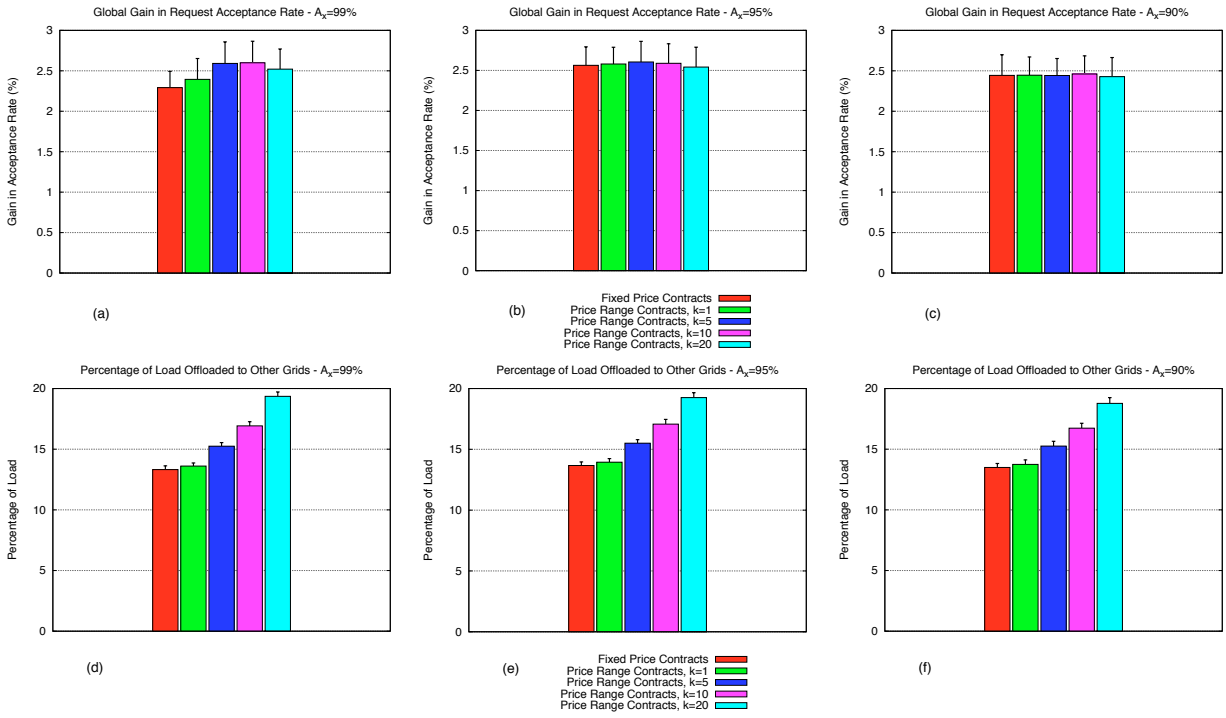

Figure 3. Increase in acceptance rate, and load exchanged amongst the Grids (*PT=30%*).

goal, which is redirect requests driven by the Grids marginal cost.

6. Conclusions and Future Work

This paper describes a mechanism and related policies that enable organisations or Grids that employ virtualisation technology to redirect requests to other Grids during periods of peak demands. The mechanism is based on contracts that specify price ranges for virtual resources exchanged between Grids. We present empirical results that demonstrate that with simple resource selection policies, the mechanism enables the migration of requests across Grids, leading to an overall increase in the request acceptance rate and redirection of requests under peak demands. We have seen that some Grids can increase the resource utilisation, without compromising the acceptance rate of requests originated at the local Grid.

Future investigations include more sophisticated resource selection policies, specially for handling advance reservation requests. IGGs with over-provisioned peering policies (i.e. that are willing to provide a large share of its resources to peering Grids) need to ensure that they are able to meet the needs its local user communities. Moreover, the proposed mechanism will be improved by providing means for Grids to redirect requests across several Grids (i.e. we are extending the mechanism to support transitive relationships between the Grids in the contract network).

Currently, we are investigating the provisioning of resources by resource providers under the presence of intermediate entities such as IGGs. Moreover, we want to work on the problem faced by gateways to provision resources to users of a Grid and to other gateways based on imprecise provisioning decisions made by providers. Additionally, in these scenarios gateways may act to maximise their utility, which is given by the difference between the price paid to the resource

Table 2. Acceptance rate and load redirected per Grid.

		A_x 99%					A_x 95%				
		Fixed	k				Fixed	k			
Metric	**Grid**	Price	1	5	10	20	Price	1	5	10	20
Increase in request acceptance rate (%)	A	0.6040	0.5718	0.4038	0.2167	0.1377	0.6198	0.7132	0.5831	0.3836	0.3660
	B	0.9213	1.0484	1.8319	2.0646	2.2168	1.7897	1.9031	2.0887	2.1332	2.3840
	C	5.5093	5.5878	5.5462	5.6052	5.4440	5.3075	5.2357	5.2984	5.3459	5.2227
	D	2.3564	2.5343	2.7384	2.7656	2.5186	2.6819	2.7081	2.6918	2.7878	2.3648
% of the generated load redirected to other Grids	A	7.76	7.87	8.89	9.94	11.49	7.75	7.91	9.02	9.98	11.55
	B	13.79	14.38	17.00	19.43	22.85	14.80	15.22	17.60	19.78	23.02
	C	23.16	23.53	25.42	27.20	30.20	23.04	23.43	25.11	27.07	29.67
	D	10.37	10.49	11.63	12.89	14.99	10.74	10.81	12.09	13.24	14.72
Increase in resource utilisation (%)	A	2.9069	3.0632	3.3575	3.7115	4.1442	3.1645	3.1666	3.4478	3.7535	4.2066
	B	5.2801	5.2719	5.1885	4.9187	4.2210	5.1720	5.0620	4.8918	4.5012	3.8351
	C	-0.5753	-0.4947	-0.4074	-0.2988	-0.3426	-0.0476	0.0052	-0.1006	-0.0340	0.0104
	D	1.0847	1.0947	1.2161	1.2891	1.4032	1.2886	1.3444	1.3973	1.3973	1.5410

		A_x 90%					A_x 85%				
		Fixed	k				Fixed	k			
Metric	**Grid**	Price	1	5	10	20	Price	1	5	10	20
Increase in request acceptance rate (%)	A	0.7026	0.5766	0.5735	0.5174	0.3935	0.6855	0.6554	0.4427	0.4795	0.3392
	B	1.8427	1.9550	1.9560	2.2031	2.3241	1.7866	1.8149	1.9576	2.1828	2.1508
	C	4.9483	4.9482	4.9569	4.8747	4.8630	4.4843	4.5853	4.4561	4.4585	4.5667
	D	2.5636	2.4953	2.4841	2.4781	2.4016	2.3509	2.3925	2.3581	2.3419	2.1160
% of the generated load redirected to other Grids	A	7.55	7.73	8.75	9.82	11.08	7.03	7.16	8.21	9.19	10.52
	B	14.81	15.33	17.41	19.56	22.48	14.22	14.76	16.75	18.76	21.41
	C	22.39	22.57	24.43	26.31	28.67	20.81	21.17	23.22	24.96	27.10
	D	10.81	10.94	11.98	13.15	14.57	10.38	10.55	11.58	12.51	13.95
Increase in resource utilisation (%)	A	3.1466	3.2162	3.4901	3.7339	4.0733	2.9848	3.0715	3.3186	3.4722	3.8211
	B	4.6328	4.5691	4.3622	4.1091	3.4430	4.0787	4.0399	3.8411	3.5232	2.9968
	C	0.3576	0.3292	0.1594	0.2535	0.4811	0.6417	0.5962	0.6974	0.6375	0.8768
	D	1.4644	1.4705	1.5006	1.4532	1.6752	1.4329	1.3388	1.5613	1.6283	1.7090

providers and the amount received from users when provisioning resources. Therefore, we are working on provisioning of resources in these multi-level federated infrastructures[3].

Acknowledgements

We thank Marco A. S. Netto, Kyong Hoon Kim, Suraj Pandey, Sungjing Choi and Vanessa Teague from the University of Melbourne for sharing their thoughts on the topic. This work is supported by Department of Education, Science and Training (DEST) and Australian Research Council (ARC) Project grants. Marcos' PhD research is partially supported by National ICT Australia (NICTA).

References

[1] Open Science Grid. http://www.opensciencegrid.org, 2005.

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

[3]In addition, an initial prototype of our infrastructure is under development. Preliminary software requirement specifications are available at: http://www.gridbus.org/intergrid/

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

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

[5] R. Buyya and M. Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience (CPE)*, 14(13–15):1175–1220, November-December 2002.

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

[7] M. D. de Assunção and R. Buyya. InterGrid: A case for internetworking islands of Grids. *Concurrency and Computation: Practice and Experience (CPE)*, Online, July 2007.

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

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

[10] L. Grit, D. Inwin, A. Yumerefendi, and 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, November 2006.

[11] D. Irwin, J. Chase, L. Grit, A. Yumerefendi, D. Becker, and K. G. Yocum. Sharing networked resources with brokered leases. In *USENIX Annual Technical Conference*, pages 199–212, Boston, Massachusetts, June 2006.

[12] C. Metz. Interconnecting ISP networks. *IEEE Internet Computing*, 5(2):74–80, 2001.

[13] K. Miura. Overview of japanese science grid project naregi. *Progress in Informatics*, pages 67–75, 2006.

[14] R. Nandkumar. International cyberinfrastructure: Activities around the globe. *Cyberinfrastructure Technology Watch Quarterly*, 2(1), February 2006.

[15] L. Peterson, S. Muir, T. Roscoe, and A. Klingaman. Planetlab architecture: An overview. Technical Report PDN-06-031, PlanetLab Consortium, Princeton, USA, May 2006.

[16] L. Peterson and J. Wroclawski. Overview of the GENI architecture. GENI Design Document GDD-06-11, GENI: Global Environment for Network Innovations, January 2007.

[17] L. Ramakrishnan, D. Irwin, L. Grit, A. Yumerefendi, A. Iamnitchi, and J. Chase. 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, 2006. ACM Press.

[18] P. Ruth, X. Jiang, D. Xu, and S. Goasguen. Virtual distributed environments in a shared infrastructure. *IEEE Computer*, 38(5):63–69, May 2005.