OurGrid: An approach to easily assemble grids with equitable resource sharing

Nazareno Andrade[1] Walfredo Cirne[1] Francisco Brasileiro[1]
Paulo Roisenberg[2]

[1]Universidade Federal de Campina Grande
http://dsc.ufcg.edu.br/ourgrid
{nazareno,walfredo,fubica}@dsc.ufcg.edu.br

[2]Hewllet Packard Brazil
paulo.roisenberg@hp.com

Abstract

Available grid technologies like the Globus Toolkit [18] make possible for one to run a parallel application on resources distributed across several administrative domains. Most grid computing users, however, don't have access to more than a handful of resources onto which they can use this technologies. This happens mainly because gaining access to resources still depends on personal negotiations between the user and each resource owner of resources. To address this problem, we are developing the OurGrid resources sharing system, a peer-to-peer network of sites that share resources equitably in order to form a grid to which they all have access. The resources are shared accordingly to a network of favors *model, in which each peer prioritizes those who have credit in their past history of interactions. The emergent behavior in the system is that peers that contribute more to the community are prioritized when they request resources. We expect, with OurGrid, to solve the access gaining problem for users of* bag-of-tasks *applications (those parallel applications whose tasks are independent).*

1 Introduction

To use grid computing, a user must assemble a grid. A user must not only have the technologies to use grid computing, but also, she must have access to resources on which she can use these technologies. For example, to use resources through the Globus Toolkit [18], she must have access, i.e., permission to use resources on which Globus is installed.

Today, the access gaining to grid resources is done via personal requests from the user to each resource's owner. To run her application on the workstations of some laboratories in a university, a user must convince the system administrator of each laboratory to give her access to their system's workstations. When the resources the user wishes to use cross institutional boundaries, the situation gets more complicated, as possibly different institutional policies come in. Thus, is very difficult for one to gain access to more than a handful of resources onto which she can use grid computing technologies to run her applications.

As resource owners must provide access to their resources to allow users to form grids, there must be interest in providing resources to grid users. Also, as several grid users may demand the same resource simultaneously, there must be mechanisms for dealing with conflicting requests for resources, arbitrating them. As this problem can be seen as an offer and demand problem, approaches to achieve this have been based on *grid economy* [2, 9, 29, 6], which means using, in grids, economic models from real markets.

Although models that mimic real markets have the mechanisms to solve the problems of a grid's offer and demand, they rely on a yet-not-available infrastructure of electronic monetary transactions. To make possible for users to securely verify what they have consumed and pay for it, there must be mature and well deployed technologies for electronic currency and banking. As these technologies are not widely deployed yet, the ac-

tual use of the economic mechanisms and architectures in real settings is postponed until the technologies are mature and the infrastructure needed to use them is available.

Nevertheless, presently there exists demand for grids to be used in production. Aiming to provide, in short term, an infrastructure that addresses this demand for an expressive set of users, we are developing OurGrid. The OurGrid design is based on a model of resource sharing that provides equity with a minimum of guaranties needed. With it, we aim to provide an easy to install, open and extensible platform, suitable for running a useful set of grid applications for users willing to share their resources in order to obtain access to the grid.

Namely, the type of application for which OurGrid intends to provide resources to are those parallel applications whose task are loosely coupled known as *bag-of-tasks* (BoT) applications [26]. BoT are those parallel applications composed of a set of independent tasks that need no communication among them during execution. Many applications in areas such as computational biology [27], simulations, parameter sweep [3] and computer imaging [25, 24] fit into this definition and are useful to large communities of users.

Additionally, from the research perspective, there exists demand for understanding grid usage requirements and patterns in real settings. With a system such as OurGrid in production for real users, we will be able to gather valuable information about the needs and habits of grid users. This allows us to both provide better guidance to future efforts in more general solutions and to collect important data about grids' usage, like workloads, for example.

The remaining of this paper is structured in the following way. In Section 2 we go into further details about the grid assembling problem, discussing related works and presenting our approach. We discuss how BoT applications are suitable for running with resources provided by OurGrid in Section 3. Section 4 describes the design of OurGrid and the network of favors model. An evaluation of the system is discussed in Section 5. In Section 6 we expose the future steps planned in the OurGrid development. Finally, we make our concluding remarks in Section 7.

2 Assembling a grid

In a traditional system, like a LAN or a parallel supercomputer, a user obtains access to resources by negotiating with the resources' owner the right to access them. Once access is granted, the system's administrator configures to the user a set of permissions and priorities. Although this procedure is still used also in grid computing, due to grid's inherent wide distribution, spawning across many administrative boundaries, this approach is not suitable.

Grid computing aims to deal with large, heterogeneous and dynamic users and resources sets [17]. Moreover, if we are to build large scale grids, we must be able to form them with mutually untrusted and even unknown parts. In this scenario, however, it is very difficult to an ordinary user to obtain access to more than a small set of services whose owners are known. As grid computing aims to provide access to large quantities of resources widely distributed, giving the users the possibility of accessing only small quantities of resources means neglecting the potential of grid computing.

The problem of assembling a grid also raises some issues from the resource providers' perspective. Suppose a very simple scenario where just two institutions, A and B, want to create a grid joining their resources. Both of them are interested in having access to as many processors as possible. Also, both of them shall want some fairness in the sharing. Probably both of them will want to assure that they will not only give access to their resources to the other institution's users, but also that its users will access the other institution's resources, maybe in equal proportions. Existing solutions in grid computing allow these two institutions to define some policies in their resource sharing, creating static constraints and guarantees to the users of the grid [15, 28, 12]. However, if a third institution C joins the grid, new agreements must be negotiated between the institutions and configured on each of them. We can easily see that these mechanisms are neither scalable nor flexible enough to the large scale grids scenarios.

2.1 Related work

Although grid computing is a very active area of research, until recently, research efforts in the dynamic access gaining to resources did not exist. We attribute this mainly to the recentness of grid computing, that has made necessary to postpone the question of access gaining until the technologies needed to use grids matured.

Past efforts have been spent in defining mechanisms that support static access policies and constraints to allow the building of metacomputing infrastructures across different administrative domains like in the Condor system [28] and in the Computational Co-op [12].

Since 1984 the Condor system has used different mechanisms for allowing a Condor user to access resources across institutional boundaries. After trying

to use institutional level agreements [16], Condor was changed to a user-to-institution level [28], to provide flexibility, as requested by its users. Recently, it was perceived also that interoperability with grid middlewares was also needed, and a new architecture for accessing grid resources was developed [19]. Although it has not dealt with dynamic access gaining, the Condor project has made valuable contributions to understanding the needs of users in accessing and using the grid.

The Computational Co-op defined a mechanism for gathering sites in a grid using cooperatives as a metaphor. This mechanism allows all sites to control how much of their resources are being used by the grid and provides guarantees on how much resource from the grid it can use. This is done through a proportional-share ticket-based scheduler. The tickets are used by users to access both local and grid resources, obtaining priorities as they spend the tickets. However, both the need of negotiations between the owners of the sites to define the division of the grid tickets and the impossibility of tickets transfers or consumption makes the Co-op not flexible enough to environments as dynamic as grids. Moreover, just as e-cash, it depends on good cryptography infrastructure to make sure that tickets are not forged.

Recent effort related to access gaining in grid computing is the research on grid economy. Namely, the Grid Architecture for Computational Economy (GRACE) [8], the Nimrod/G system [2] and the Compute Power Market [9] are related to our work. The GRACE is an abstract architecture that supports different economic models for negotiating access to grid resources. Nimrod/G is a grid broker for the execution of parameter sweep applications that implements GRACE concepts, allowing a grid client to negotiate access to resources paying for it. The Compute Power Market aims to provide access to resources in a decentralized manner, through a peer-to-peer network, letting users pay in cash for using grid resources. An important point to note in these approaches is that for allowing negotiations between service consumers and providers using secure global currencies as proposed by Nimrod/G and Compute Power Market, an infrastructure for the secure negotiation, payments and banking must be deployed. The level of maturity of the basis technologies — as, for example, secure and well deployed electronic money — makes necessary to postpone the use of economic-based approaches in real systems.

2.2 OurGrid approach

The central point of OurGrid is the utilization of assumptions that, although more restrictive to the system's usefulness, are easier to satisfy than those of existing systems based on grid economy. Our assumptions about the environment in which the system will operate are that (i) there are at least two peers in the system willing to share their resources in order to obtain access to more resources and (ii) the applications that will be executed using OurGrid need no quality of service (QoS) guarantees. With these assumptions, we aim to build a resource sharing network that promotes equity in the resources sharing. By equity we mean that participants in the network which have donated more resources are prioritized when they ask for resources.

With the assumption that there will be at least two resource providers in the system, we ensure that there will exist participants in the system which own resources whose access can be exchanged. This makes possible the use of an exchange based economic model, instead of the more commonly used price based models [2, 9].

By assuming that there are no requirements for QoS guarantees, we put aside negotiations, once providers need not to negotiate a product whose characteristics won't be guaranteed. Without negotiations, it becomes unnecessary that participants even agree on values for the resources allocated and consumed. This simplifies the process, once consumers don't have to verify that an agreed value was really consumed and providers don't have to assure that resources are provided as agreed.

Actually, in this way we are building the simplest form of an exchanged based economic model. As there's no negotiation, every participant does favors expecting to be reciprocated, and, in conflicting situations, prioritizes those who have done favors to it in the past. The more a participant offers, the more it expects to be rewarded. There are no negotiations or agreements, however. Each participant accounts its favors only to himself, and cannot expect to profit from them in other way than getting other participants to make him favors.

As there is no cost in donating idle cycles — as they will be forever lost if not consumed instantaneously —, a participant in the model can only gain from donating them. As, by our first assumption, there exists at least one other participant which is sharing her idle resources, donating implies in eventually benefiting from accessing extra resources.

As we shall see, from the local behavior of all partic-

ipants, the emergent behavior of the system promotes equity in the arbitration of conflicting requests for the shared resources in the system. An important point is that the absence of QoS guarantees makes impossible to *guarantee* equity in the resource sharing. The system can't guarantee that a user will access enough resources to compensate the amount she donated to the community, because it can't guarantee that there will ever be available resources for the time needed. As such, we propose a system that aims not to guarantee, but to *promote* the resource sharing equity. Promoting equity means trying, via a best-effort strategy, to achieve equity.

The proposed assumptions about the system ease the development and deployment of OurGrid, restricting, in turn, its utility. The necessity of the participants to own resources excludes users that don't own any resources, but are willing to pay (e.g. in cash) to use the grid. Also, the absence of QoS guarantees makes impossible the advance reservation of resources and, consequently, preclude mechanisms that provide synchrony to the execution of parallel applications that need communication between tasks.

We believe, however, that even with this restrictions, OurGrid will still be very useful. OurGrid delivers services that are suitable to the bag-of-tasks class of applications. As stated before, these applications are relevant to many areas of research, being interesting to many users.

3 Bags-of-tasks applications

Due to the independence of their tasks, BoT applications are especially suited for execution on the grid, where both failures and slow communication channels are expected to be more frequent than in conventional platforms for the execution of parallel applications. Moreover, we argue that this class of applications can be successfully executed without the need of QoS guarantees, as in the OurGrid scenario.

A BoT application can perform well with no QoS guarantees as it (i) does not need any synchronization between tasks, (ii) has no dependencies between tasks and (iii) can tolerate faults caused by resources unavailability with very simple strategies. Example of such strategies to achieve fault tolerance in the OurGrid scenario are replication of tasks in multiple resources or the simple re-submition of tasks that failed to execute [22, 25]. As such, this class of application can cope very well with resources that neither are dedicated nor can have their availability guaranteed, as failure in the execution of individual tasks does not impact on the execution of the other tasks.

Besides performing well with our assumptions, another characteristic of BoT applications that matches our approach is their users work cycle. Experience says that, once the application is developed, users usually carry out the following cycle: (a) plan details of the computation, (b) run the application, (c) examine the results, (d) restart cycle. Planning the details of the computation means spending the time needed to decide the parameters to run the application. Often, a significant amount of time is also needed to process and understand the results produced by a large scale computation.

As such, during the period in which a user is running her BoT application, she wants as much resources as possible, but during the other phases of her working cycle, she leaves her resources idle. These idle resources can be provided to other users to grant, in return, access to other users' resources, when needed. An example of this dynamic for two BoT application users is illustrated in Figure 1. In this Figure, the users use resources both local and obtained from the grid — which, in this case, are only the other user's idle resources — whenever they need to run their BoT applications. Note that whenever a user needs her own resources, it has priority over the foreign user.

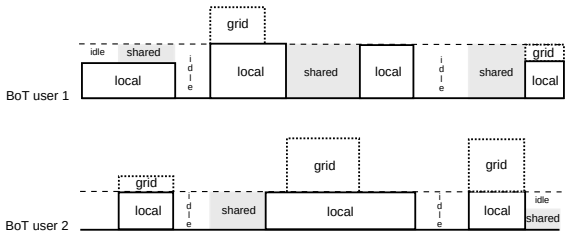

Figure 1. Idle resource sharing between two BoT users

Another point to note is that, as resources are heterogeneous, a user might not own the resources she needs to run an application that poses constraints on the resources it needs. For example, a user can own machines running both Linux and Solaris. If she wants to run an application that can run only on Solaris, she won't be able to use all of her resources. As such, it is possible for a user to share part of her resources while consuming other part, maybe in addition to resources from other users.

In this way, we believe that expecting BoT applications users to share some of their resources in order to gain access to more resources is very plausible. As stated before, this kind of exchange can be carried out without any impact to the resources owners,

because there are exchanged only resources that otherwise would be idle. In return, they get extra resources when needed to run their applications.

4 OurGrid

Based on the discussed approach we intend to develop OurGrid to work as a peer-to-peer network of resources owned by a community of grid users. By adding resources to the peer-to-peer network and sharing them with the community, a user gains access to all the available resources on it. All the resources are shared respecting each provider's policies and OurGrid strives to promote equity in this sharing.

A user accesses the grid through the services provided by a peer, which maintains communication with other peers and uses the community services (e.g. application-level routing and discovery) to access them, acting as a grid broker to its users. A peer P will be accessed by *native* and *foreign* users. Native users are those who access the OurGrid resources through P, while foreign users have access to P's resources via other peers.

A peer is both a consumer and a provider of resources. When a peer P is making a favor in response to a request from a peer Q, P is acting as a provider of resources to Q, while Q is acting as a consumer of P's resources.

OurGrid network architecture is shown in Figure 2. Clients are software used by the users to access the community resources. A client is at least an application scheduler, possibly with extra functionalities. Examples of such clients are MyGrid [14, 13], APST [10], Nimrod/G [3] and AppLeS [7].

We plan to provide access, through OurGrid, to different resource types. In Figure 2, for example, the resources of type A could be clusters of workstations accessed via Globus GRAM, the type B resources could be parallel supercomputers and type C resources could be workstations running MyGrid's UserAgent [14].

Although resources of any granularity (i.e. workstations, clusters, entire institutions, etc.) can be encapsulated in an OurGrid peer, we propose them to manage access to whole sites instead of to individual resources. As resources are often grouped in sites, using this granularity in the system will give us some advantages: (i) the number of peers in the system diminishes considerably, improving the performance of searches; (ii) the system's topology becomes closer to its network infrastructure topology, alleviating traffic problems found in other peer-to-peer systems [23]; and (iii) the system becomes closer to the real ownership distribution of the resources, as they are, usually grouped in sites, each with its proper set of users and owners.

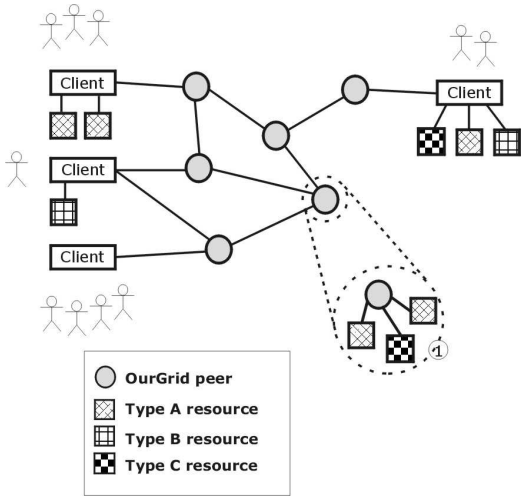

Figure 2. OurGrid network architecture

Finally, an OurGrid community can be part of a larger set of resources that a user has access to, and users can be native users of more than one peer, either in the same or in different communities.

In the rest of this section, we describe in details the key aspects of OurGrid design. In Subsection 4.1 we present the model accordingly to which the resources are shared, the *network-of-favors*. Subsection 4.2 depicts the protocol used to gain access to the resources of an OurGrid community.

4.1 The network of favors

All resources in the OurGrid network are shared in a network of favors. In this network of favors, allocating a resource to a requesting consumer is a favor. As such, it is *expected* that the consumer becomes in debt with the owner of the consumed resources. The model is based on the expectation that its participants will reciprocate favors to those consumers they are in debt with, when solicited. If a participant is not perceived to be acting in this way, it is gradually less prioritized, as its debt grows.

Every peer in the system keeps track of a local *balance* for each known peer, based on their past interactions. This balance is used to prioritize peers with more credit when arbitrating conflicting requests. For a peer p, all consumption of p's resources by another peer p' is debited from the balance for p' in p and all resources provided by p' to p is credited in the balance p maintains for p'.

With all known peers' balances, each participant can

maintain a ranking of all known participants. This ranking is updated on each provided or consumed favor. The quantification of each favor's value is done locally an independently — as negotiations and agreements aren't used —, serving only to the decisions of future resource allocations of the local peer. As the peers in the system ask each other favors, they gradually discover which participants are able to reciprocate their favors, and prioritize them, based on their debt or credit.

As a consequence, while a participant prioritizes those who cooperate with him in satisfactory ways, it marginalizes the peers who, for any reason, do not reciprocate the favors satisfactorily. The non-reciprocation can happen for many reasons, like, for example: failures on services or on the communication network; the absence of the desired service in the peer; or the utilization of the desired service by other users at the moment of the request. Free-rider [4] peers may even choose not to reciprocate favors. In all of this cases, the non-reciprocation of the favors gradually diminishes the probability of the peer to access the grid's resources.

Note that our mechanism of prioritizing intends to solve only conflicting situations. It is expected that, if a resource is available and idle, any user can access it. In this way, an ordinary user can, potentially, access all the resources in the grid. Thus, users that contribute very little or don't contribute can still access the resources of the system, but only if no other peer that has more credit requests them. The use of idle and not requested resources by peers that don't contribute (i.e., free-riders) actually maximizes the resource utilization, and does not harm the peers who have contributed with their resources.

Another interesting point is that our system, as conceived, is totally decentralized and composed of autonomous entities. Each peer depends only on its local knowledge and decisions to be a part of the system. This characteristic greatly improves the adaptability and robustness of the system, that doesn't depend on coordinated actions or global views [5].

4.2 The OurGrid resource sharing protocol

To communicate with the community, gain access to, consume and provide resources, all peers use the OurGrid resource sharing protocol. Note that the protocol concerns only the resource sharing in the peer-to-peer network. We consider that the system uses lower-level protocols to other necessary services, such as peers discovery and broadcasting of messages. An example of a platform that provides these protocols is the JXTA [1] project.

The three participants in the OurGrid resource sharing protocol are clients, consumers and a providers. A client is a program that manages to access the grid resources and to run the application tasks on them. OurGrid will be one of such resources, transparently offering computational resources to the client. As such, a client may (i) access both OurGrid peers and other resources directly, such as Globus GRAM [15] or a Condor Pool [28]; and (ii) access several OurGrid peers from different resource sharing communities. We consider that the client encompasses the application scheduler and any other domain-specific module needed to schedule the application efficiently.

A consumer is the part of a peer which receives requests from a user's client to find resources. The consumer is used first to request resources to providers that are able and willing to do favors to it, and, after obtaining them, to execute tasks on the resources. Providers are the part of the peers which manages the resources shared in the community and provides them to consumers.

As illustrated in Figure 3, every peer in the community has both a consumer and a provider modules. When a consumer receives a request for resources from a local user's client, it broadcasts to the peer-to-peer network the desired resources' characteristics in a *ConsumerQuery* message. The resources' characteristics are the minimum constraints needed to execute the tasks this *ConsumerQuery* message is referring to. It is responsibility of the client to discover this characteristics, probably asking this information to the user. Note that as it is broadcasted, the *ConsumerQuery* message also reaches the provider that belongs to the same peer the consumer does.

All providers whose resources match the requested characteristics and are available (accordingly to their local policies) reply to the requester with a *ProviderWorkRequest* message. The set of replies received up to a given moment defines the grid that has been made available for the client request by the OurGrid community. Note that this set is dynamic, as replies can arrive later, when the resources needed to satisfy the request became available at more providers.

With the set of available resources, it is possible for the consumer peer to ask for its client to schedule tasks onto them. This is done sending a *ConsumerScheduleRequest* message containing all known available providers. The application scheduling step is kept out of the OurGrid scope to allow the user to select among existing scheduling algorithms [11, 22] the one that optimizes her application accordingly to her knowledge about the characteristics of the applica-

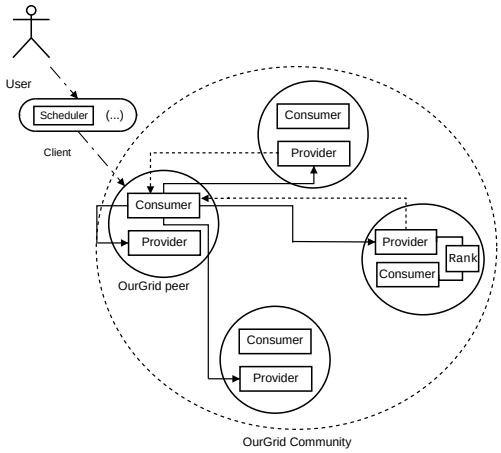

Figure 3. Consumer and provider interaction.

tion.

Once the client has scheduled any number of tasks to one or more of the providers who sent $ProviderWorkRequest$ messages, it sends a $ClientSchedule$ message to the consumer to which it requested the resources. As each peer represents a site, owning a set of resources, the $ClientSchedule$ message can contain either a list of ordered pairs $(task, provider)$ or a list of tuples $(task, provider, processor)$. It is up to the client deciding how to format its $ClientSchedule$ message. All tasks are sent through the consumer and not directly from the client to the provider, to allow the consumer to account its resource consumption.

To each provider P_n in the $ClientSchedule$ message, the consumer then sends a $ConsumerFavor$ message containing the tasks to be executed in P_n with all the data needed to run it. If the peer who received the $ConsumerFavor$ message finishes its tasks successfully, it then sends back a $ProviderFavorReport$ message to the corresponding consumer. After concluding each task execution, the provider also updates its local rank of known peers, *subtracting* the accounting it made of the task execution cost from the consumer peer's balance. The consumer peer, on receiving the $ProviderFavorReport$, also updates its local rank, but *adding* the accounting it made of the reported tasks execution cost to the provider balance. Note that the consumer may either trust the accounting sent by the provider or make its own autonomous accounting.

While a provider owns available resources that match the request's constraints and is willing to do favors, it keeps asking the consumer for tasks. A provider may decide not to continue making favors to a consumer in order to prioritize another requester who is upper in its ranking. The provider also decides to stop requesting tasks if it receives a message from the consumer informing that there are no tasks left to schedule or if it receives no response from a task request. Note that, after the first broadcast, the flow of requests is from the providers to the consumer. As the $ProviderWorkRequest$ messages are the signal of availability, we alleviate the consumer from the task of managing the state of its current providers.

In Figure 4, a sequence diagram for an interaction between a consumer and two providers is shown. The provider *provider*1 makes a favor to *consumer*, but *provider*2 either is unable or has decided not to provide any resources to *consumer*.

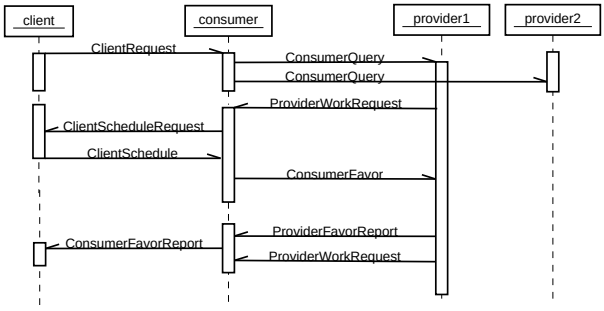

Figure 4. Sequence diagram for a consumer and two providers interaction

Although an OurGrid network will be an open system, potentially comprised by different algorithms and implementations for its peers, we present in the following sections examples of expected correct behaviors for both the peer's provider and consumer. The algorithms intend to exemplify and make clearer how should a peer behavior to obtain access to the community's shared resources.

4.2.1 Provider algorithm

A typical provider runs three threads: the receiver, the allocator and the executor. The receiver and the allocator execute continuously, both of them access, add, remove and alter elements of the list of received requests and of known peers. The executor is instantiated by the allocator to take care of individual tasks execution and accounting.

The receiver thread keeps checking for received requests. For each of the requests received, it verifies if the request can be fulfilled with the owned resources. It does so by verifying if the provider owns resources to satisfy the request's requirements, no matter if they are available or not, accordingly to the peer's sharing

policies. If the consumer request can be satisfied, the receiver adds it to a list of received requests. There are two such lists, one for requests issued by local users and another one for those issued by foreign users. This allows us to prioritize local users requests in the scheduling of tasks to the local resources. The allocator thread algorithm is shown in Algorithm 1.

While executing, the allocator thread continuously tries to satisfy the received requests with the available resources. It tries to find first a request from a local user which can be fulfilled, and if there's none, it tries the same with the community requests received.

The function *getLocalRequestRanked()*, in line labeled 2, returns the request with the specified position in the priority ranking, accordingly to a local set of policies. The policies can differ from peer to peer, but examples of local prioritizing policies would be FIFO or to prioritize the users who consumed less in their past histories. The function *getCommunityRequestRanked()*, in line labeled 5, does the same thing, but for the community requests. It must be based on the known peers balances, that serve as a ranking to prioritize these requests.

On lines labeled 3 and 6, the allocator verifies if the resources necessary specified to fulfill the request passed as a parameter are available, according to the local policies of availability. If some request was chosen to be answered in this iteration of the main loop, the allocator decides which resources will be allocated to this request (line labeled 7) and sends a message asking for tasks to execute.

If it receives tasks to execute, it then schedule the received tasks on the resources allocated to that request. This is done on the *execute()* function, which creates a provider executor thread for each task that will be executed. The executor first sets up the environment in which the task will execute. To set up the environment means to prepare any necessary characteristics specified by the local policies of security. For example, it could mean to create a directory with restricted permissions or with restricted size in which the task will execute.

After the task is effectively executed and its results have been collected and sent back, the executor must update the credit of the consumer peer. The quantifying function may differ from peer to peer. A simple example of how it could be done is to sum up all the CPU time used by the task and multiply it by the CPU speed, in MIPS. Once the peer has estimated the value of the favor it just made, it updates the *knownPeersBalances* list, decreasing the respective consumer balance.

For simplicity, we are considering here that our provider's allocator scheduling is non-preemptive. However, it is reasonable to expect that, to avoid impacting on interactive user of the shared resources, a provider may suspend or even kill tasks from foreign users.

4.2.2 Consumer algorithm

As we did with the provider, this section will discuss a simple yet functional version of a consumer algorithm. The consumer runs three threads: the requester, the listener and the remote executor. The requester responsibility is to broadcast client requests it receives as *ConsumerQuery* messages.

After the *ConsumerQuery* message for a given *ClientRequest* message has been sent, the consumer listener thread starts waiting for its responses. It receives all the ProviderWorkRequest messages sent to the peer, informing that the resources are available to the Client as they arrive.

Each instance of the remote executor thread, as illustrated in Algorithm 2, is responsible for sending a set of tasks to a provider, waiting for the responses and updating the balance of this provider in the local peer. The quantification is shown on line labeled 1, and may differ from peer to peer. Examples of how it can be performed may vary from simply using the accounting sent by the provider to more sophisticated mechanisms, such as sending a micro-benchmark to test the resource performance, collect the CPU time consumed and then calculating the favor cost as a function of both. Yet another possibility is to estimate the task size, maybe asking the user this information, and then assigning a cost based on this size to each task execution.

The provider's balance is updated on line labeled 2. Note that the usage is added to the provider's balance, while in the provider's executor it was deducted.

5 Evaluation

In this section we show some preliminary results from simulations and analytical evaluation of the OurGrid system. Note that, due to its decentralized and autonomous nature, characterizing the behavior of an OurGrid community is quite challenging. Therefore, at this initial moment, we base our analysis on a simplified version of OurGrid, called OurGame. OurGame was designed to capture the key features of OurGrid, namely the system-wide behavior of the network of favors and the contention for finite resources. The simplification consists of grouping resource consumption into turns. In a turn, each peer is either a provider or a consumer. If a peer is a consumer, it tries to consume

```
   Data      : communityRequests, localRequests, knownPeersCredits, localPriorityPolicies
   while true do
       chosen = null ;
       /* local users' requests are prioritized over the community's */;
1      if localRequests.length > 0 then
           rank = 1 ;
           repeat
2              actual = getLocalRequestRanked( localRequests, localPriorityPolicies, rank++ );
3              if isResourceToSatisfyAvailable( actual ) then
                   chosen = actual ;
               end
           until ( chosen != null ) || ( rank > localRequests.length) ;
       end
       /* if there's no local user's request which can be satisfied */;
4      if ( chosen == null ) && ( communityRequests.lenght > 0 ) then
           rank = 1 ;
           repeat
5              actual = getCommunityRequestRanked( communityRequests, knownPeersBalances, rank++ );
6              if isResourceToSatisfyAvailable( actual ) then
                   chosen = actual ;
7                  resourcesToAllocate = getResourcesToAllocateTo( chosen );
               end
           until ( chosen != null ) || ( rank > communityRequests.length) ;
       end
       /* actually allocate resource to the chosen task */;
8      if chosen != null then
           send( chosen.SrcPeerID, ProviderWorkRequest ) ;
           receivedMessage = receiveConsumerFavorMessage( timeout ) ;
           if receivedMessage != null then
               receivedTasks = getTasks( receivedMessage );
               foreach task in receivedTasks do
10                 execute( tasks, resourcesToAllocate );
               end
           else
11             if isRequestLocal( chosen ) then
                   localRequests.remove( chosen ) ;
               else
12                 communityRequest.remove( chosen ) ;
               end
           end
       end
   end
```

Algorithm 1: Provider's allocator thread algorithm

all available resources. If a peer is a provider, it tries to allocate all resources it owns to the current turn consumers. In short, OurGame is a repeated game that captures the key features of OurGrid and allows us to shed some light over its system-wide behavior.

5.1 OurGame

Our system in this model is comprised of a community of n peers represented by a set $P = \{p_1, p_2, ..., p_n\}$. Each peer p_k owns a number r_k of resources. All resources are identical, but the amounts in each peer may be different. Each peer can be in one of two states: *provider* or *consumer*. When it is in the provider state, it is able to provide all its local resources, while in the consumer state it sends a request for resources to the community. We consider that when a peer is in the consumer state it consumes all its local resources and, as such, it cannot provide resources to the community. All requests sent by the consumers are equal, requesting as much resources as can be provided.

A peer p_k is a tuple $\{id, r, state, ranking, \rho, allocationStrategy\}$. The id field represents this peer identification, to be used by other peers to control its favor balance. As stated before, r represents p_k's amount of resources, *state* represents the peer's actual state, assuming the *provider* or *consumer* values. The *ranking* is a list of pairs $(peer\_id, balance)$, representing the known peers ranking. In this pair, *peer_id* represents a known peer and *balance* the credit or debit associated with this peer. To all unknown peers, we consider $balance = 0$. The ρ field is the probability of p_k of being a provider in a given turn of the game.

The *allocationStrategy* element of the tuple defines the peer's resource allocation behavior. As instances of the *allocationStrategy*, we have implemented *AllForOneAllocationStrategy*

```
   Data      : provider, scheduledTasks, knowPeersCredits
   send( provider, ConsumerFavor );
   unansweredTasks = scheduledTasks;
   while ( unansweredTasks.lenght > 0 ) && ( timeOutHasExpired( ) == false ) do
   |   results = waitProviderFavorReport( providerID );
   |   answeredTasks = results.getTasks( );
   |   removeReportedTasks( answeredTasks, unansweredTasks );
   |   foreach task in answeredTasks do
   |   |   if isProviderLocal ( provider ) == false then
1  |   |   |   usage = quantifyUsage( results );
   |   |   |   previousBalance = getPeerBalance( knownPeerBalance, provider );
2  |   |   |   updatePeerBalance( knownPeersBalance, provider, ( previousBalance + usage ) );
   |   |   end
   |   end
   end
```

Algorithm 2: Consumer's remote executor thread algorithm

and *ProportionallyForAllAllocationStrategy*. The former allocates all of the provider's resources to the consumer that has the greatest balance value (ties are broken randomly). The *ProportionallyForAllAllocationStrategy* allocates the peer's resources proportionally to all requesting peers with positive balance values. If there are no peers with positive balance values, it allocates to all with zero balance values and if there are no requesting peer with a non-negative balance value, it allocates proportionally to all requesting peers.

In this model the time line is divided in turns. The first action of all peers in every turn is to, accordingly to its ρ, choose its state during the turn, either consumer or provider. Next, all peers who are currently in consumer state send a request to the community. All requests arrive in all peers instantaneously, asking for as many resources as the peer owns. As our objective is studying how the system deals with conflicting requests, all consumers always ask for the maximum set of resources.

On receiving a request, each provider chooses, based on its *allocationStrategy* which resources to allocate to which consumers, allocating always all of its resources. All allocations last for the current turn only. In the end of the turn, each peer updates its *ranking* with perfect information about the resources it provided or consumed in this turn.

5.2 Scenarios

To verify the system behavior, we varied the following parameters:

- Number of peers: We have simulated communities with 10, 100 and 1000 peers;
- Peers strategies: Regarding the *allocationStrategy*, we have simulated the following scenarios: 100% of the peers using *AllForOneAllocationStrategy*, 100% using *ProportionallyForAllAllocationStrategy* and the following combinations between the two strategies in the peers: (25%, 75%), (50%, 50%) and (75%, 25%).
- Peer probability of being a provider in a turn (ρ): We have simulated with all peers having a probability of 0.25, 0.50 and 0.75 to be in the provider state. Also, we have simulated an heterogeneous scenario in which each peer has a probability of being a provider given by a uniform distribution in the interval [0.00..0.99]. We have not considered peers with probability 1.00 of being a provider because we believe that the desire of consuming is the primary motivation for a site to join an OurGrid community, and a peer would not joint it to be always a provider.
- Amount of resources owned by a peer: All peers own an amount of resources in a uniform distribution in the interval [10..50]. We considered this to be the size of a typical laboratory that will be encapsulated in an OurGrid peer.

All combinations of those parameters gave us 60 simulation scenarios. We have implemented the model and this scenarios using the SimJava [21] simulation toolkit[1].

5.3 Metrics

Since participation in an OurGrid community is voluntary, we have designed OurGrid (i) to promote equity (i.e., if the demand is greater than the offer of resources, the resources obtained from the grid should be equivalent to resources donated to the grid), and (ii)

[1] SimJava is available at `http://www.dcs.ed.ac.uk/home/simjava/`

to prioritize the peers that helped the community the most (in the sense that they have donated more than they have consumed). We gauge equity using *Favor Ratio* (FR) and the prioritization using *Resource Gain* (RG).

The Favor Ratio FR_k of a peer p_k after a given turn is defined by the ratio of the accumulated amount of resources gained from the grid (note that this excludes the local resources consumed) by the accumulated amount of resources it has donated to the grid. More precisely, for a peer p_k which, during t turns, gained g_k resources from the grid and donated d_k to the grid, $FR_k = g_k/d_k$. As such, FR_k represents a relation between the amount of resources a peer gained and how much resources it has donated. If $FR_k = 1$, peer p_k has received from the grid an amount of resources equal to that it donated. That is, $FR_k = 1$ denotes equity.

The Resource Gain RG_k of a peer p_k after a given turn is obtained dividing the accumulated amount of resources used by it (both local and from the grid) by the accumulated amount of local resources it has used. As such, let l_k be all the local resources a peer p_k consumed during t turns and g_k the total amount of resources it obtained from the grid during the same t turns, $RG_k = (l_k + g_k)/l_k$. RG_k measures the "speed-up" delivered by the grid, i.e. how much grid resources helped a peer in comparison to its local resources.

Note that RG_k represents the resources obtained by a peer when it requested resources, because whenever a peer asks for grid resources, it is also consuming its local resources. Thus, we can interpret RG_k as a quantification of how much that peer was prioritized by the community.

Thus, to verify the equity in the system-wide behavior, we expect to observe that, in situations of resource contention, $FR_k = 1$ for all peers p_k. We want also to verify if the peers which donated more to the community are really being prioritized. To gauge this, we will use RG_k, which we expect to be greater for the peers with the greatest differences between what they have donated and what they have consumed from the community.

Due to the considerations of this model, we can easily draw a relation between RG_k and FR_k. Consider a peer p_k, let r_k be the amount of resources it owns, t the number of turns executed, l_k its local resources consumed, d_k the amount of resources it donated to the community, i_k the resources that went idle because there were no consumers in some turns in which p_k was in the provider state and ρ_k the probability of the peer p_k being a provider in a given turn. Let us also denote R_k as the total amount of resources that a peer had available during t turns. As such:

$$\begin{cases} R_k = t.r_k \\ R_k = l_k + d_k + i_k \\ l_k = (1 - \rho_k).R_k \end{cases} \quad (1)$$

From (1) and the definitions of FR_k and RG_k, we can derive that:

$$RG_k = 1 + \frac{\rho_k.FR_k}{(1-\rho_k)} - \frac{i_k.FR_k}{(1-\rho_k).t.r_k} \quad (2)$$

Another relation that is useful is obtained from the fact that the total amount of resources available in the system is the sum of all resources obtained from the grid, local consumed and left idle for all peers. As all resources donated are consumed or left idle, no resources are lost nor created, we can state a resource conservation law as follows:

$$\sum_k R_k = \sum_k g_k + \sum_k l_k + \sum_k i_k \quad (3)$$

5.4 Results discussion

With the OurGame model, the scenarios in which we instantiated the model and the metrics to measure its characteristics presented, we shall now show the results we have obtained so far. We will divide the discussion between the scenarios in which all peers have the same providing probability ρ and those in which ρ_k is given by a uniform distribution in [0..0.99]. In each of them we will then examine how the number of peers, the providing probabilities ρ_k and the *allocationStrategy* they were using impacted their RG_k and FR_k values, and, consequently, on the network of favors behavior and on the OurGrid resource contention.

5.4.1 Results for communities in which all peers have equal providing probabilities

For all scenarios in which ρ_k was equal to all peers, both FR_k and RG_k converged. FR_k always converged to 1, but RG_k converged to different values, depending on the scenarios' parameters.

With all peers competing with the same appetite for resources, each peer gains back the same amount of resources it has donated to the community, that explains the convergence of FR_k. Figure 5 shows this happening, despite variance on the amount of resources owned by each peer. The three lines in the Figure are a peer with the greatest r_k, a peer with a mean value of r_k and a peer with the smaller r_k in the scenario.

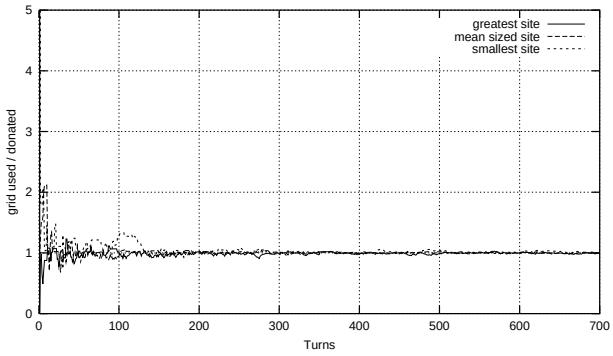

Figure 5. FR for peers with different resource quantities in a 10-peer community using ProportionallyForAllAllocationStrategy with $\rho = 0.25$

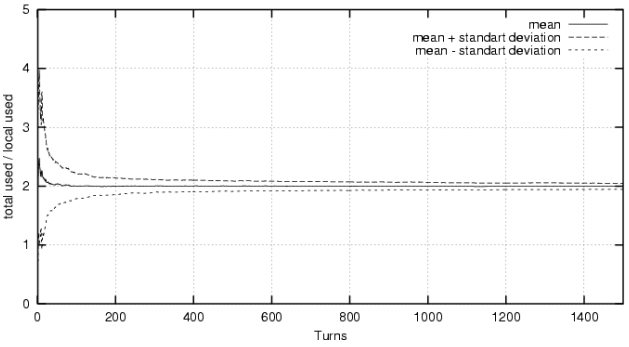

Figure 6. RG in a 100-peer community using ProportionallyForAllAllocationStrategy and with $\rho = 0.50$

Regarding RG_k, with $FR_k = 1$, from equation (2), we obtain that $RG_k = 1 + \frac{\rho_k}{(1-\rho_k)} - \frac{i}{(1-\rho_k).t.r_k}$. To facilitate our understanding, we divide the analysis of RG_k behavior in two situations: the scenarios in which there are no idle resources (i.e., $i_k = 0$) and the scenarios in which there are idle resources (i.e., $i_k > 0$).

Analytically analyzing the scenarios, we observe that in a given scenario, $i > 0$ happens if there is no consumer in some round. The probability of all peers $p_1, ..., p_n$ to be in the provider state is $AP = \prod_{1 \leq k \leq n} \rho_k$. Thus, the number of turns in which all resources in the community were idle in a given scenario is $IT = t.AP$. For all scenarios but the ones with 10 peers and 0.50 or 0.75 providing probabilities, $IT = 0$.

In the scenarios where $i_k = 0$, as $FR_k = 1$ we find, from equation (2) that $RG_k = 1 + \rho_k/(1 - \rho_k)$. As $0 \leq \rho_k < 1$, this means that $RG_k \propto \rho_k$. As such, the peers with greater ρ_k are the peers which have the greater difference between what they donated and consumed, the fact of $RG_k \propto \rho_k$ shows that the more a peer contribute to the community, the more it is prioritized. As, in this scenarios, all peers have the same ρ, however, they all have the same RG_k. For example, in a community in which all peers have $\rho = 0.50$, we found RG_k from all peers converged to $RG = 1 + 0.5/(1 - 0.5) = 2$. As can be seen in Figure 6, for a 100-peer community.

In the scenarios in which $i > 0$, we also found $FQ_k = 1$ and RG_k also converged. However, we observed two differences in the metrics behavior: (i) FR_k took a greater number of turns to converge and (ii) RG_k converged to a value smaller than in the scenarios where $i = 0$. The former behavior difference happened because each peer took a longer time to rank the other peers, as there happened turns with no resource consumption. The latter behavior difference is explained by equation (3). As, for a given peer p_k, l_k is fixed, the idle resources i_k actually are resources *not consumed*. This means that g_k and, consequently, RG_k decreases as i_k increases. In short, as the total amount of resources consumed by the peers is less than the total amount of resources that were made available (i.e., both donated and idle), their RG_k is smaller than in the scenarios where all resources made available were consumed.

Finally, regarding the strategy the peers used to allocate their resources, we found that varying the strategy used by the peers did not affect significantly the metrics behavior. The number of peers in the community, on the other hand, naturally affects the number of turns needed to both metrics converge. The number of turns needed to the metrics to converge is bigger as the size of the community grows.

5.4.2 Results for communities in which peers have different providing probabilities

After observing the effects of each of the simulation parameters in a community that had the same probability for consuming resources, we now discuss how the variation on this probability affects our defined metrics.

First, in the simulations of the 10-peer communities, we found that FR_k did not converge. Figure 7 shows FR_k for three peers of a community with this number of peers and in which the providing chance ρ_k of each peer p_k is given by a uniform distribution in the interval [0.00..0.99]. As can be seen, the peer which donated less to the community — as its providing chance is smaller that of all other peers' in Figure 7 —, ob-

tained the greater FR. This is easily explained if we take a look also in the values of RG_k for these peers. The RG_k behavior for the same three peers is shown in Figure 8. Note that, for the peer with the greatest ρ, RG_k explodes the scale in its first request, after some turns providing, what gives us the almost vertical solid line in the graph. Figure 8 shows how a peer is prioritized as it donates more resources to the community. Consequently, the peer which provided more resources is the peer with the greatest RG. Whenever it asks for resources, it manages to get access to more resources than a peer that has provided less to the community.

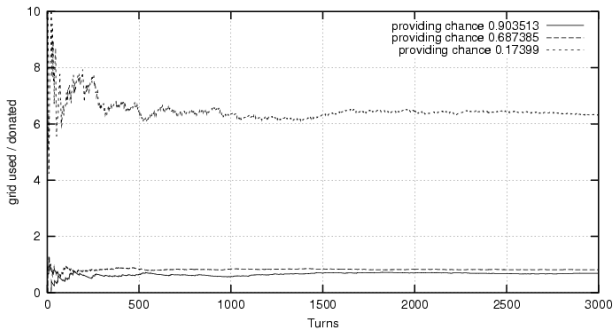

Figure 7. FR for three peers in a 10-peer community with different providing probabilities using ProportionallyForAllAllocationStrategy

The peer with the lesser providing chance obtained more resources from the grid, and thus got a greater FR_k because it actually requested more resources and donated just a little bit. As the providing probabilities are static, the peers with greatest probabilities provided more and didn't asked resources often enough to make their FR_k raise. Thus, FR_k did not converge because there was not enough competition in these scenarios, and there were turns in which only peers which contributed with small amounts of resources to the community requested resources. Note that without enough competition for the resources we cannot observe the fairness of the system. Nevertheless, by observing RG_k, we still can observe how the prioritization was done, when the peers which contributed more to the community did asked for resources.

An interesting behavior we have observed is that with the growth of the community size, FR_k once again converges to 1. It happens for the 100-peer and the 1000-peers communities, in our simulations. The histogram[2] of FR_k in a 100-peer community in the turn 3000 is show in Figure 9.

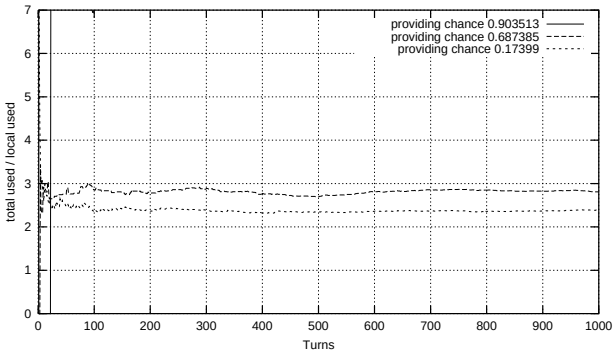

Figure 8. RG for three peers in a 10-peer community with different providing probabilities using ProportionallyForAllAllocationStrategy

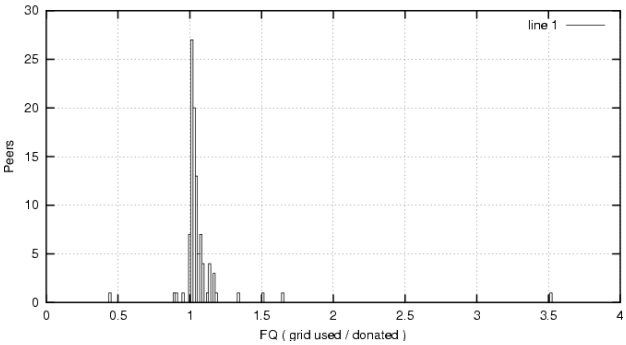

Figure 9. FR histogram in a 100-peer community with different allocation strategies on turn 3000

The convergence of FR_k happens due to the greatest concurrence present in greater communities. As there are more peers, there are less turns in which only peers with small ρ_i request the resources of the community. As such, less peers manage to obtain FR_k as high as happened in the 10 peers scenarios. This may still happen, if there are only peers that donate very little in a sufficiently large number of turns. Nevertheless, this is not prejudicial to our objectives, as these resources could not be allocated to a peer that contributed significantly to the community.

[2]We opted to show a histogram due to the great number of peers in the simulation.

With $FR_k = 1$ and $i = 0$, we again find that $RG_k \propto \rho_k$. This shows that peers which contributed more, that is, which have the highest ρ_k, were more prioritized.

We shall remark that again, our two allocation strategies did not show impact in the simulations results. As such, in the long run, peers that allocate all of their resources to the highest ranked peer perform as well as peers that allocate their resources proportionally to the balances of the requesters.

6 Future directions

The next steps in the OurGrid development are (i) simulating real grid users workloads on the peers; (ii) studying the impact of malicious peers in the system; and (iii) the actual implementation of OurGrid. Now we have evaluated the key characteristics of our network of favors, simulating more realistic scenarios is needed to understand the impact of the grid environment in the model presented in this work. Peer's maliciousness is important mostly in two aspects in OurGrid: a peer consumer shall want to assure that a provider executed a task correctly and there must not be possible to exploit the community using unfair accounting.

More specifically in the malicious peers problem, to deal with the need of the consumer to assure correct task execution in unreliable providers, we plan to study both (a) replication in order to discover providers a consumer can trust and (b) the insertion of application specific verification, like the techniques described in [20]. To cope with the objective of making the community tolerant to peers using unfair accounting, marginalizing them, we aim to study the use of (a) autonomous accounting and (b) replication to determine if a consumer shall trust unknown providers.

We plan to start OurGrid implementation as an extension of MyGrid[3] [14, 13] former work done at UFCG. OurGrid will be able to serve as a MyGrid resource in the user's grid, and will initially obtain access to resources through the already existent MyGrid's Grid Machine Interface. The Grid Machine Interface is an abstraction that provides access to different kinds of grid resources (Globus GRAM, MyGrid's UserAgent, Unix machines via ssh, etc.) and will allow OurGrid to interoperate with existing grid middleware. Interoperability is important to both take advantage of existing infrastructure and to ease the OurGrid adoption by the community of users.

[3]MyGrid is open-source and it is available at `http://dsc.ufcg.edu.br/mygrid/`

7 Conclusions

We have presented the design of OurGrid, a system that aims to allow users of BoT applications to easily obtain access and use computational resources, dynamically forming an on-demand, large-scale, grid. Also, opting for simplicity in the services it delivers, OurGrid will be able to be deployed immediately, both satisfying a current need in the BoT users community and helping researchers in better understanding how grids are really used in production, a knowledge that will help to guide future research directions.

OurGrid is based on a *network of favors*, in which a site donates its idle resources as a favor, expecting to be prioritized when it asks for favors from the community. Our design aims to provide this prioritization in a completely decentralized manner. The decentralization is crucial to keep our system simple and not dependent on centralized services that might be hard to deploy, scale and trust.

Our preliminary results on the analysis through simulation of this design to solve the conflict for resources in a decentralized community shows us that this approach is promising. We expect to evolve the present design into a solution that, due to its simplicity, will be able to satisfy a need from real grid users *today*.

8 Acknowledgments

We would like to thank Hewlett Packard, CNPq and CAPES for the financial support. It was crucial to the progress of this work. We would also like to thank Elizeu Santos-Neto, Lauro Costa and Jacques Sauvé for the insightful discussions that much contributed to our work.

References

[1] Project JXTA. http://www.jxta.org/.

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

[3] Abramson, D., Giddy, J., and Kotler, L. High performance parametric modeling with Nimrod/G: Killer application for the global grid? In *Proceedings of the IPDPS'2000* (2000), IEEE CS Press, pp. 520–528.

[4] Adar, E., and Huberman, B. A. Free riding on gnutella. *First Monday 5*, 10 (2000). http://www.firstmonday.dk/.

[5] Babaoglu, O., and Marzullo, K. *Distributed Systems*. Addison-Wesley, 1993, ch. 4: Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms.

[6] Barmouta, A., and Buyya, R. GridBank: A Grid Accounting Services Architecture (GASA) for distributed systems sharing and integration. In *26th Australasian Computer Science Conference (ACSC2003)* (2003 (submitted)).

[7] Berman, F., Wolski, R., Figueira, S., Schopf, J., and Shao, G. Application-level scheduling on distributed heterogeneous networks. In *Supercomputing'96* (1996).

[8] Buyya, R., Abramson, D., and Giddy, J. An economy driven resource management architecture for computational power grids. In *International Conference on Parallel and Distributed Processing Techniques and Applications* (2000).

[9] Buyya, R., and Vazhkudai, S. Compute Power Market: Towards a Market-Oriented Grid. In *The First IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2001)* (Beijing, China, 2000), IEEE Computer Society Press.

[10] Casanova, H., Hayes, J., and Yang, Y. Algorithms and software to schedule and deploy independent tasks in grids environments. In *Workshop on Distributed Computing, Metacomputing and Resource Globalization* (2002).

[11] Casanova, H., Legrand, A., Zagorodnov, D., and Berman, F. Heuristics for scheduling parameter sweep applications in grid environments. In *Proceedings of the 9th Heterogeneous Computing Workshop* (Cancun, Mexico, May 2000), IEEE Computer Socity Press, pp. 349–363.

[12] Cirne, W., and Marzullo, K. The computational Co-op: Gathering clusters into a metacomputer. In *PPS/SPDP'99 Symposium* (1999).

[13] Cirne, W., and Marzullo, K. Open Grid: A user-centric approach for grid computing. In *13th Symposium on Computer Architecture and High Performance Computing* (2001). Available at http://walfredo.dsc.ufpb.br/resume.html#publications.

[14] Cirne, W., Paranhos, D., Costa, L., Santos-Neto, E., Brasileiro, F., Sauv, J., da Silva, F. A. B., Osthoff, C., and Silveira, C. Running bag-of-tasks applications on computational grids: The MyGrid approach. Submitted to publication. Available at http://dsc.ufcg.edu.br/ourgrid/, 2003.

[15] Czajkowski, K., Foster, I., Karonis, N., Kesselman, C., Martin, S., Smith, W., and Tuecke, S. A resource management architecture for metacomputing systems. In *IPPS/SPDP'98 Workshop on Job Scheduling Strategies for Parallel Processing* (1998), pp. 62–82.

[16] Epema, D., Livny, M., van Dantzig, R., Evers, X., and Pruyne, J. A worldwide flock of Condors: Load sharing among workstation clusters. *Future Generation Computer Systems 12* (1996), 53–65.

[17] Foster, I. The anatomy of the Grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science 2150* (2001).

[18] Foster, I., and Kesselman, C. The Globus project: A status report. In *IPPS/SPDP '98 Heterogeneous Computing Workshop* (1998), pp. 4–18.

[19] Frey, J., Tannenbaum, T., Foster, I., Livny, M., and Tuecke, S. Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing 5* (2002), 237–246.

[20] Golle, P., and Mironov, I. Uncheatable distributed computations. *Lecture Notes in Computer Science 2020* (2001), 425–441.

[21] Howell, F., and McNab, R. SimJava: a discrete event simulation package for Java with applications in computer systems modelling. In *Procedings of the First International Conference on Web-based modelling and simulation* (1998), S. for Computer Simulation, Ed.

[22] Paranhos, D., Cirne, W., and Brasileiro, F. Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids. In *Proceedings of the Euro-Par 2003: International Conference on Parallel and Distributed Computing* (2002). Available at http://dsc.ufcg.edu.br/ourgrid/.

[23] Ripeanu, M., and Foster, I. Mapping the Gnutella network: Macroscopic properties of

large-scale peer-to-peer systems. In *First International Workshop on Peer-to-Peer Systems (IPTPS)* (2002).

[24] Santos-Neto, E. L., Tenrio, L. E. F., Fonseca, E. J. S., Cavalcanti, S. B., and Hickmann, J. M. Parallel visualization of the optical pulse through a doped optical fiber. In *Proceedings of Annual Meeting of the Division of Computational Physics* (Boston, MA, USA, 2001).

[25] Smallen, S., Cirne, W., Frey, J., Berman, F., Wolski, R., Su, M.-H., Kesselman, C., Young, S., and Ellisman, M. Combining workstations and supercomputers to support grid applications: The parallel tomography experience. In *Proceedings of the HCW'2000 - Heterogeneous Computing Workshop* (2000).

[26] Smith, J., and Shrivastava, S. K. A system for fault-tolerant execution of data and compute intensive programs over a network of workstations. In *Lecture Notes in Computer Science* (1996), vol. 1123, IEEE Press.

[27] Stiles, J. R., Bartol, T. M., Salpeter, E. E., and Salpeter, M. M. Monte carlo simulation of neuromuscular transmitter release using MCell a general simulator of cellular physiological processes. *Computational Neuroscience* (1998), 279–284.

[28] Thain, D., Tannenbaum, T., and Livny, M. *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley, 2003, ch. 11 - Condor and the grid.

[29] Wolski, R., Plank, J., Brevik, J., and Bryan, T. Analyzing market-based resource allocation strategies for the computational grid. *International Journal of High-performance Computing Applications 15*, 3 (2001).