Scheduling in Bag-of-Task Grids: The PAUÁ Case

Walfredo Cirne Francisco Brasileiro Lauro Costa
Daniel Paranhos Elizeu Santos-Neto Nazareno Andrade
Universidade Federal de Campina Grande
{walfredo,fubica,lauro,danielps,elizeu,nazareno}@dsc.ufcg.edu.br

César De Rose Tiago Ferreto
PUCRS
{derose,ferreto}@inf.pucrs.br

Miranda Mowbray Roque Scheer João Jornada
Hewlett Packard
{miranda.mowbray,roque.scheer,joao.jornada}@hp.com

In this paper we discuss the difficulties involved in the scheduling of applications on computational grids. We highlight two main sources of difficulties: firstly, the size of the grid rules out the possibility of using a centralized scheduler; secondly, since resources are managed by different parties, the scheduler must consider several different policies. Thus, we argue that scheduling applications on a grid require the orchestration of several schedulers, with possibly conflicting goals. We discuss how we have addressed this issue in the context of PAUÁ, a grid for Bag-of-Tasks applications (i.e. parallel applications whose tasks are independent) that we are currently deploying throughout Brazil.

1 Introduction

The use of computational grids as platform to execute parallel applications is a promising research area. The possibility to allocate unprecedent amounts of resources to a parallel application and to make it with lower cost than traditional alternatives (based in parallel supercomputers) is one of the main attractives in grid computing. On the other hand, the grid characteristics, such as high heterogeneity, complexity and wide distribution (traversing multiple administrative domains), create many new technical challenges.

In particular, the area of *scheduling* faces entirely new challenges in grid computing. Traditional schedulers (such as the operating system and the supercomputer scheduler) *control* all resources of interest. In a grid, such a central control is not possible. First, the grid is just too big for a single entity to control. Second, the resources that comprise a grid are owned by many different entities, rendering it administratively unacceptable that a single entity controls all resources. In a grid, a scheduler must strive for its traditional goals (improve system and application performance), while realizing that most of the system is not under its control. In fact, most of the system will be under control of other schedulers. Therefore, a given scheduler must interact with (or at least consider) other schedulers in order to achieve its goals. In a way, the multiple schedulers present in a grid form an *ecology*, where individual schedulers compete and/or collaborate with other schedulers, and the overall system behavior emerges from the decisions made by all schedulers.

This paper discusses the scheduling ecology found in PAUÁ, a 250-node grid that supports the execution of Bag-of-Tasks applications. Bag-of-Tasks (BoT) applications are those parallel applications whose tasks are independent of each other. Despite their simplicity, BoT applications are used in a variety of scenarios, including data mining, massive searches (such as key breaking), parameter sweeps [1], simulations, fractal calculations, computational biology [31], and computer imaging [28] [29]. Moreover, due to the independence of their tasks, BoT applications can be successfully executed over widely distributed computational grids, as has been demonstrated by SETI@home [2]. In fact, one can argue that BoT applications are the applications most suited for computational grids, where communication can easily become a bottleneck for tightly-coupled parallel applications.

Focusing in BoT applications is interesting because the problem is simplified, but remains useful and relevant. The major simplification introduced by focusing on BoT applications is that we do not need Quality-of-Services guarantees. Since the tasks that compose a BoT application are independent, having a task making progress very slowly (or even stopping!) can be dealt with no major problems. At worst, the task can be resubmitted elsewhere.

The scheduling ecology found in PAUÁ was designed to (i) respect site autonomy, (ii) cater for the user's priorities, (iii) enable multilateral collaboration (in contrast to the much more common bilateral collaboration), (iv) support both dedicated and non-dedicated resources, and (v) explicitly separate architectural components from implementation (thus easing the addition of new schedulers to the ecology).

These design goals were achieved by separating the grid scheduling concerns in three main aspects, namely: (i) improve the performance of the application over the whole grid, (ii) manage resources within a *site* (i.e. a set of resources within a single administrative domain), and (iii) gain access to resources throughout the grid. Concern (i) is the responsibility of a *job scheduler*, whereas concern (ii) is the responsibility of a *site scheduler*. The grid has many job schedulers, as well as many site schedulers. A user has rights of use (i.e., an account) in one or more sites. Moreover, sites can dynamically grant access to foreign users via a peer-to-peer mechanism called *network of favors*, thus addressing concern (iii).

Section 2 surveys the area and discusses related work. Section 3 describes the experience in building the PAUÁ's community. We present PAUÁ's scheduling ecology in Section 4. Section 5 presents a set of experiments that gauges the performance one can expect from PAUÁ. Section 6 contains our conclusions and delineates future work.

2 Related Work

Grid computing is a very active area of research [7] [17]. Although it has started within High Performance Computing, people have realized that Grid technology could be used to deliver computational services on-demand. This observation has brought about the merge between Grid and Web Services technologies, as seen in standards like OGSA/OGSI [32] and its successor WSMF [14]. These standards are currently being implemented by both academia and industry. Most notably, these standards are being implemented by Globus [19], maybe the project with greatest visibility in Grid Computing.

However, it is important to realize that WSMF-like technologies do not address scheduling or resource management directly. They rather provide the grid building blocks, the common foundation on which grids are built. Scheduling is thought to happen pervasively throughout the grids, with each service making its own scheduling decisions (which may be delegated to specialized services) [13]. That is, the overall grid scheduling is a result of the scheduling decisions made by multiple autonomous (yet related) entities. In particular, the scheduling decisions made by a given service *s* must take into account the quality of service provided by the services invoked by *s*.

The idea that a single scheduler cannot deal with the entire grid dates from the mid 1990s, with Berman et al seminal work on application-level scheduling [6]. Since then, there has been a number of works on the many aspects of scheduling in grids. These aspects include, for example, coping with the dynamicity of grid resource availability (e.g.[33]), the impact of large data transfers (e.g.[15]), coordination among many schedulers to deliver a combined service (e.g.[30]), and virtualization as a way to ease scheduling (e.g.[8]).

Closer to our work, there are scheduling efforts that target BoT applications, such as APST [11] [12], Nimrod/G [1] and Condor [16] [21]. In particular, APST and Nimrod/G are similar to MyGrid, our job scheduler, in intent and architecture. However, they require much more information than MyGrid for scheduling. Moreover, they also differ from MyGrid in the assumptions about the application and the grid. APST targets divisible workloads, whereas in MyGrid the user is the responsible for breaking the application's work into tasks. Nimrod/G assumes that the user is going to pay for resources and hence scheduling is based on a grid economy model [9].

Condor was initially conceived for campus-wide networks [21], but has been extended to run on grids [16]. Whereas MyGrid, APST and Nimrod/G are user-centric schedulers, Condor is system-centric scheduler. Condor is thus closer to OurGrid, our site scheduler. The major difference between OurGrid and Condor is that OurGrid was designed to encourage people to donate their resources to the community (since resources received are made proportional to resources donated), whereas in Condor this issue is taken off-line (e.g. altruism or administrative orders lead people into a Condor pool).

Condor and OurGrid create grids on which resource providers and resource consumers are roles played by the same people. As an alternative, *public computing* efforts suggest a more asymmetrical view, on which many people voluntarily donate resources to a few projects of great public appeal. Arguably, public computing originated from the huge success achieved by SETI@home [2], which has harvested close to 2,000,000 years of CPU so far [27]. SETI@home makes no distinction between the application itself (search of extraterrestrial intelligence evidence in radio signals) from its grid support. However, BOINC [3] has been introduced as a SETI@home sequel, promising exactly such as separation. BOINC aims to create a public computing infrastructure that can be used by different applications. The Bayanihan project also aims to create a public computing infrastructure and carries a very interesting contribution on tolerating sabotage (i.e. bogus volunteer results) [26].

3 The PAUÁ Community

PAUÁ, which means "everything" in Tupi-Guarani (an ancient language spoken by native Brazilians), is an initiative created by HP Brazil R&D to build a countrywide Brazilian Grid. PAUÁ currently involves 11 different universities and research centers that collaborate with HP Brazil R&D in what we call the "HP Brazil's research ecosystem". The goals of PAUÁ are twofold. The first goal is to take advantage of a number of computational resources available on the different research centers as well as HP Brazil R&D itself, creating a wide, geographically distributed Grid along the country. The second goal to foster grid research, so that the solution currently being developed is constantly improved based on its own usage and experience.

UFCG is responsible for the MyGrid and OurGrid as well as research on independent auditing of SLAs (Service Level Agreements) in Grids, integration with supercomputers, among others. Instituto Atlântico focuses on security aspects in the Grid. UNISINOS is also focusing on security as well as management aspects. Instituto Eldorado is adding Windows support as well as helping the community on the configuration management and training. Hewlett Packard Brazil R&D is doing research on idle cycle exploitation and applications' execution security (sandboxing). IPT/SP is working on testing, applications development and web services. CPAD/PUC-RS is performing research on clusters' integration so that cluster's resources can be used in a transparent manner. CAP/PUC-RS is developing Grid applications. LNCC is working on the field of Bioinformatics applications. Finally, UNIFOR and UNISANTOS are doing research on using grids to perform data mining.

There are several challenges that arise when coping with the decentralized administration of such geographically distributed community. Just to cite a few, one must take into account the evolution of each research being carried out, synchronize the correct time each research institution joins the community, increases or decreases the resources allocated to a given institution, and plan the integration of a new piece of technology into the community common software. To cope with these challenges, the grid's policies are managed and defined by a general committee, which is formed by research center's representatives. The committee is responsible for establishing and defining a flexible, dynamic and non-anarchical community, as well as synchronizing and attuning all the different activities being developed. The committee has regular tele-conference meetings, which are used to track integration activities and define the next steps. Because of the number of different people involved, besides having regular tele-conference meeting, the committee also meets face-to-face a couple of times a year.

4 Scheduling in PAUÁ

A grid such as PAUÁ poses many challenges for its schedulers. The resources are widely spread, making it very difficult to have an efficient global snapshot of the grid. Also, there are multiple users and multiple resource owners, each with particular wishes and priorities. This scenario creates the need for the system to have multiple schedulers. We thus have designed and implemented a set of schedulers that are collectively responsible for the scheduling in PAUÁ. As discussed before, these schedulers must respect the autonomy of each site, considering the priorities associated to different users. Further, it must support both dedicated and non-dedicated resources. Finally, their interaction needs to be so that facilitates the addition of new schedulers to the grid.

We achieved the above design goals by separating the grid scheduling concerns in three main aspects. A key aspect is the improvement of the performance of the application over the whole grid. This is achieved by a *job scheduler* that does so by following a very efficient and lightweight approach, as will be explained shortly. The next aspect is the definition of the concept of a *site*, within which resources are managed following a particular policy (i.e. a site comprises a set of resources within a single administrative domain). A *site scheduler* is in charge of imposing the site policy. The final aspect is that of providing a way to gain access to resources throughout the grid (i.e. resources of a foreign site). This is the responsibility of a *peer-to-peer resource exchange network* involving all site schedulers.

The user submits a BoT job to a job scheduler, which sends a request for resources to all sites the user has an account on. Each of these sites is controlled by a site scheduler, which allocates resources to the job scheduler in a best-effort basis. These resources may be local resources (which are controlled by the site scheduler itself) or foreign resources (which are obtained via the network of favors). As the job scheduler begins to receive resources from the site schedulers, it starts to farm out the tasks that compose the application. The goal of the job scheduler is to minimize the application execution time. Note that resources are offered to the job scheduler in a best-effort basis. That is, resources may "disappear" at any time. As such, the job scheduler itself must guarantee that tasks finish by resubmitting them whenever necessary.

In PAUÁ parlance, the peer-to-peer resource exchange network of favors is called *OurGrid*. Therefore, the site scheduler is an *OurGrid peer*. The job scheduler is termed *MyGrid*.

4.1 Job Scheduler

Despite the simplicity of BoT applications, scheduling BoT applications on grids is difficult. Grids introduce two issues that complicate matters. First, efficient schedulers depend on a lot of information about application (such as estimated execution time) and resources (processor speed, network topology, and so on), however this kind of information is typically difficult to obtain [10]. Second, since many important BoT applications are also data-intensive applications, considering data transfers is paramount to achieve good performance. Thus, in order to achieve efficient schedules, one must provide a coordinated data and computation scheduling, which is a non-trivial task.

MyGrid's first scheduler (Workqueue with Replication, or WQR) dealt only with the first issue. WQR uses task replication to recover from bad task to machine allocations (which are inevitable, since it uses to information). WQR performance is as good as traditional knowledge-based schedulers fed with perfect information, at the cost of consuming more cycles [23]. However, WQR does not take data transfers into account.

With version 2.0 of MyGrid, we released an alternative scheduler for MyGrid: Storage Affinity, which does tackle both problems simultaneously. But note that WQR is still available within MyGrid because it does quite a good job with CPU-intensive BoT applications.

There are a few grid schedulers that take data transfers into account in order to improve the performance of the applications. Of those, the one that likely had greater visibility is XSufferage [10]. As its name suggest, XSufferage is an extension of the Sufferage scheduling heuristic, and therefore, is a knowledge-based scheduler.

In order to cope with lack of information about environment and data placement concerns, we have developed a novel scheduling heuristic for data-intensive BoT applications. This heuristic is named Storage Affinity. The idea is to exploit data reutilization to avoid unnecessary data transfers. The data reutilization appears in two basic flavors: *inter-job* and *inter-task*. The former arises when a job uses the data already used by (or produced by) a job that executed previously, while the latter appears in applications whose tasks share the same input data.

In order to take advantage of the data reutilization and improve the performance of Data-Intensive BoT applications, we introduce the *storage affinity* metric. This metric determines *how close* to a site a given task is. By *how close* we mean *how many bytes* of the task input dataset are already stored at a specific site. Thus, *storage affinity* of a task to a site *is the number of bytes within the task input dataset that are already stored in the site*.

We claim that information (data size and data location) can be obtained a priori without difficulty and loss of accuracy, unlike, for example, CPU and network loads or the completion time of tasks. For instance, this information can be obtained if a data server at a particular site is able to answer requests about *which* data elements it stores and *how large* is each data element. Alternatively, an implementation of a Storage Affinity heuristic can easily store a history of previous data transfer operations containing the required information.

Naturally, since Storage Affinity does not use dynamic information about the grid and the application, inefficient *task-to-processor* assignments may occur. In order to circumvent this problem, Storage Affinity uses a task replication strategy similar to that used by WQR [23]. Replicas have a chance to be submitted to faster processors than those processors assigned to the original task, thus increasing the chance of the task completion time to be decreased.

A total of 3,000 simulations were performed to investigate the efficiency of Storage Affinity against other heuristics. Each simulation consisted of a sequence of 6 executions of the same job. These executions are repeated for each of the 3 analyzed scheduling heuristics (WQR, XSufferage and Storage Affinity). Therefore, we have 18,000 execution time values for each scheduling heuristic analyzed.

Table 1 presents a summary of the simulation results. It is possible to note that, in average, Storage Affinity and XSufferage achieve comparable performances. The results show that both data-aware heuristics attain much better performance than WQR. This is because data transfer delays dominate the execution time of the application, thus not taking them into account severely hurts the performance of the application. In the case of WQR, the execution of each task is always preceded by a costly data transfer operation (as can be inferred from the large bandwidth and small CPU waste). This impairs any improvement that the replication strategy of WQR could bring. On the other hand, the replication strategy of Storage Affinity is able to cope with the lack of dynamic information and yields a performance very similar to that of XSufferage. The main inconvenience of XSufferage is the need for knowledge about dynamic information, whereas the drawback of Storage Affinity is the consumption of extra resources due to its replication strategy (an average of 59% of extra CPU cycles and a negligible amount of extra bandwidth). Naturally, we do not report any wasting values for XSufferage because this heuristic does not apply any replication strategy.

		Storage Affinity	WQR	XSufferage
Execution Time (sec)	**Mean**	14377	42919	14665
	Std Dev	10653	24542	11451
Wasted CPU (%)	**Mean**	59.24	1.08	N/A
	Std Dev	52.71	4.12	N/A
Wasted Bandwidth (%)	**Mean**	3.19	130.88	N/A
	Std Dev	8.57	135.82	N/A

Table 1 – Storage Affinity simulations results

From this result we can state that the Storage Affinity task replication strategy is a feasible technique to obviate the need for dynamic information when scheduling data-intensive BoT applications, although at the expenses of consuming more CPU. We refer the reader to [24] for a complete performance analysis of Storage Affinity.

4.2 Site Scheduler

Computational Grids are composed by resources from several sites. Such resources can have different processor architectures and use diverse operating systems. Consequently they may also differ in performance, ranging from desktop machines to supercomputers. In BoT grids we consider only two types of resources: (i) space-shared parallel machines, such as MPPs (Massive Parallel Processors) and clusters; and (ii) time-shared resources, such as desktop-machines that can be accessed at any time.

The access to site resources cannot be made without a request. For example, in a MPP system, the access to nodes cannot be made without a request to the resource manager. This happens because the resource manager decides when a job will be executed and on which machine nodes it will run. Strictly speaking, this is also true for desktop machines, since there is no resource that can be shared without the intervention of a resource scheduler. In this last case the operating system will be the resource scheduler.

According to the Global Grid Forum Scheduling Dictionary Working Group [18] there are two types of resource schedulers involved at site level. The *Local Scheduler* determines how the system processes its job queue in the case of space shared resources like a MPP or cluster. It is implemented usually in the system resource manager. The *Machine Scheduler* is used when the resource is just one machine (i.e. a desktop computer). This type of scheduler uses some criteria to schedule jobs, such as priority, length of time in the job queue and available resources. It is implemented in the operating system of the machine.

In this paper we are introducing a new type of resource scheduler called *Site Scheduler*. It represents the site resources in the grid making them available to the higher schedulers. Thus, access to the schedulers described above (local and machine) must go through the site scheduler. The main responsibilities for a site scheduler are: (i) verification of access rights for grid jobs; (ii) abstraction of site resource types for the grid; and (iii) arbitration between site demand and grid demand.

The verification of access rights is needed for security reasons (i.e. to block grid users that are not eligible to use site resources). Access rights can also be used to impose limitations related to the maximum number of allocated resources or the exclusion of some specific resource types. Time related restrictions are also possible like the exclusion of grid accesses in peak hours. The resource abstraction is an interesting feature since grid users do not have to care about site resource types. The negotiations the site manager has to do with the local and machine schedulers to allocate the site resources should be transparent to the grid. Arbitration is also a key aspect since local users and grid users will be competing for the same resources. The site manager has to find a good balance between them to maintain the external interference to a reasonable level, and thus not delay local users too much. A priority policy could be used to guarantee a better response to special users.

Additional services may include caching for tasks and executables (so the site scheduler would function as a proxy), resource monitoring and performance prediction of the whole site (to be used in higher resource levels). Another interesting issue is to consider the site scheduler the only known IP of a site. In this case the grid would see the site as one virtual resource that would have the sum of a site's available resources.

4.3 Network of Favors

To form a grid that shares resources between multiple organizations, it is not only necessary to have an infrastructure that allows the site schedulers to use each other's resources. It is also necessary for the resource owners to make their resources available to the grid. Although this is an obvious statement, making this happen is not straightforward. Our experience shows that making people contribute their resources to the community is one of the hardest tasks in assembling a grid. This experience is backed up by empirical studies of several peer-to-peer resource sharing communities, showing that in the absence of incentives for resource donation, most users only consume resources from the system, donating nothing back [20] [25].

To provide incentives for donating resources to the grid, OurGrid implements a scheme called the network of favors [4] [5]. Each site offers to the community access to its idle resources, expecting to gain access to the idle resources of other participants when its work exceeds its local capacity. To motivate sites to share as many resources as possible, the network of favors is designed to promote fairness in the resource sharing; that is, the more a site donates to the community, the more it should expect to receive from the community.

In the network of favors, when one site consumes resources owned by another site, that is regarded as a favor paid by the resource owner to the consumer. Every site in the system stores a local balance of favors received minus favors given for each other known site, based on its past interactions with the other site. This balance is updated on providing or consuming favors. When there are conflicting requests for resources, the resource owner prioritizes requests made by sites with higher balances. The quantification of each favor's value is done locally and independently – negotiations and agreements are not used – and affects only decisions of future resource allocations made by the two sites involved.

Sites that do not reciprocate favors satisfactorily will over time be given lower priority by the community. The non-retribution may happen for several reasons, such as local resource failures, the absence of resources at the site, or the use of the desired resources locally or by other users at the moment of the request. Free-riding sites may even choose not to reciprocate favors. In any case, the non-retribution of the favors gradually diminishes the ability of the site to access the grid's resources. This behavior is illustrated in Figure 1. This figure shows the results of a simulation of a 100-site community with different proportions *f* of free-riders. It is possible to see that the fraction of the community resources obtained by the free-riders (*epsilon*) diminishes over time, tending to a very small value.

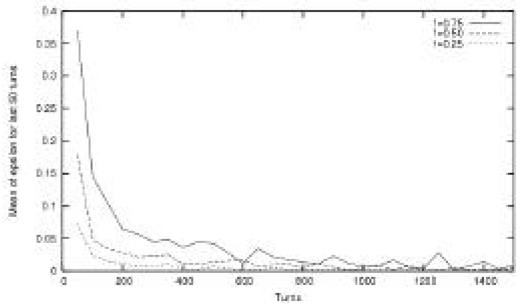

Figure 1 – Resources obtained by free riders

We have also verified through simulations that the amount of resources that a collaborator receives divided by the amount it donates (denoted *FR*) is approximately 1. Figure 2 illustrates this for a 100-site community in which the amount a site donates is given by a uniform distribution U(1,19). It is reasonable to assume that the cost of donating a resource is smaller than the utility gained by receiving it. Therefore, it is in the interest of sites to donate as many resources as they can.

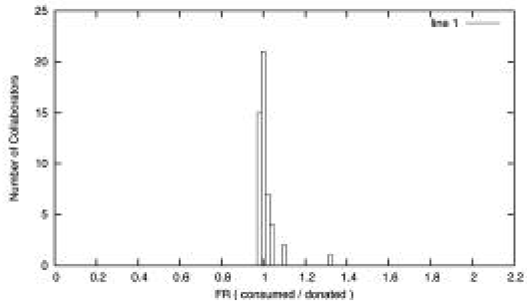

Figure 2 – Distribution of Favor Ratio

5 Running on PAUÁ

We have just started running BoT applications over PAUÁ. Here we present the results attained by a very simple experiment we have conducted. The application used is a CPU-bound BoT application that finds the divisors of a very large number. The experiment was conducted in a small subset of PAUÁ, composed of four different sites. Each site has a peer acting as a site scheduler providing nodes to grid users. Table 3 shows the sites used, their localizations, the maximum number of nodes available[1], nodes configuration and peers names.

[1] Only the idle resources were made available to the grid. By idle we mean resources that are not being used by local users. This shows how site schedulers

Site	Location	Max. Number of Nodes	Peers names
CPAD	Porto Alegre	24	marfim
LSD	Campina Grande	30	lula, robalo[2]
LCC	Campina Grande	5	seulunga
Lab-Petri	Campina Grande	3	bandolim

Table 3 – Testbed configuration

Site schedulers communicate with the resource schedulers within their sites in order to obtain resources. The peers LSD, LCC and LabPetri have to deal with only machine schedulers (the O.S. of their machines). The CPAD site scheduler is more complex; its major resources are controlled by CRONO [22], a local scheduler for clusters. In this way, this peer communicates with two types of resource schedulers: the O.S. of desktop machines and the CRONO local scheduler of its cluster nodes.

The experiments were executed as follows. A MyGrid running on a desktop machine at the CPAD site acts as the job scheduler. We performed two sets of experiments with the environment described above in order to analyze the speed up attained by the grid when compared to the execution of the application on a standalone setting. The first set of experiments was composed by jobs with small tasks (1 minutes on a dedicated Pentium III 733 MHz). The second set has tasks that were 5.2 times longer than the tasks of the first task. The first set obtained speedup ranging from 6.2 to 11.1, with an average of 7.5. The peak number of machine gained by the site scheduler to its job scheduler was 33, in the fastest job executed. The set of longer tasks had speedup ranging from 14 to 31.3, with an average of 22.3. This set reached a peak of 35 machines in utilization.

As it was expected, we obtained an improvement in the application performance. However, they also highlighted the overhead impact of the PAUÁ structure. The set with small tasks does not present a speedup as good as the second set due to the latency to gain a grid machine (communication between peers) and prepare an environment (e.g. transfer the binary of task) to execute the tasks. This implies that, at least for the moment, application must have reasonable large granularity to execute well in PAUÁ.

6 Conclusions and Future Work

We have presented a strategy to deal with the scheduling of BoT application on large scale grids. The strategy proposed is supported by a set of schedulers divided in two distinct classes and a peer-to-peer resource harnessing mechanism. Site schedulers are responsible for providing grid resources to job schedulers that, in turn, provide efficient scheduling of BoT applications over the available resources. We currently have two job schedulers that use a replication mechanism to achieve efficient scheduling without requiring any information about the grid or the applications being scheduled. Site schedulers ensure the implementation of policies set by the resource owner. They also try to find remote resources using a peer-to-peer resource trading protocol (the network of favors protocol). Our results show that the ecology of schedulers work as intended, although they require parallel applications of course grain to fully benefit from the grid.

It is important to point out that the scheduling problem is just one of the many problems that need to be tackled in order to deploy a grid such as PAUÁ. In particular, security issues are very challenging as well. We are currently working a virtual-machine-based sandbox technology to address some of these issues. Another clear avenue for improvement is the relaxation of the BoT requirement we currently pose for the application. Our strategy is to relax the application constraints incrementally, supporting a broader class of applications at each step. Our next step will be what we call workflow applications, i.e. parallel applications with tasks whose input comes from another task output.

As described here, PAUÁ is currently deployed in 4 of the 11 institutions that compose the grid. The other 7 institutions currently use MyGrid alone, i.e. they form a grid with (local and remote) resources they have an account on. They are installing OurGrid and moving to the architecture herein described in the next three months.

Although PAUÁ is (at least for now) a closed grid, all the software described in this paper is open source. MyGrid and OurGrid are available at www.ourgrid.org, whereas CRONO is available at sourceforge.net/projects/crono.

can be configured in order to maintain performance of local resources that are in use by local users.

[2] The reason for the LSD site to have two peers is the following: one acts as a site scheduler (lula), while the other (robalo) acts as a relay. This is required because there is a connection lack due to the UFCG firewall configuration. Only a peer (robalo) can make connections to LCC, thus this machine was used as an application relay of the grid to LCC resources.

References

[1] D. Abramson, J. Giddy, L. Kotler. *High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?* IPDPS'2000, pp. 520-528, 2000.

[2] D. Anderson, J. Cobb, E. Korpela. *SETI@home: An Experiment in Public-Resource Computing*. Comm. ACM, vol. 45, no. 11, pp 56-61, Nov. 2002.

[3] D. Anderson. *Public Computing: Reconnecting People to Science*. Conference on Shared Knowledge and the Web, Madrid, Spain, 2003.

[4] N. Andrade, W. Cirne, F. Brasileiro, P. Roisenberg. *OurGrid: An Approach to Easily Assemble Grids with Equitable Resource Sharing*. JSSPP'2003.

[5] N. Andrade, F. Brasileiro, W. Cirne, M. Mowbray. *Discouraging Free Riding in a Peer-to-Peer CPU-Sharing Grid*. HPDC'2004, June 2004.

[6] F. Berman et al. *Application-Level Scheduling on Distribute Heterogeneous Networks.* Supercomputing'96, Pittsburgh, 1996.

[7] F. Berman, G. Fox, T. Hey (Editors). *Grid Computing: Making The Global Infrastructure a Reality*. John Wiley & Sons, 2003.

[8] L. Burchard et al. *The Virtual Resource Manager: An Architecture for SLA-aware Resource Management*. 4th International Symposium on Cluster Computing and the Grid, 2004.

[9] R. Buyya, D. Abramson, J. Giddy. *An Economy Driven Resource Management Architecture for Global Computational Power Grids*. PDPTA 2000.

[10] H. Casanova et al. *Heuristics for Scheduling Parameter Sweep Applications in Grid Environments*. 9th HCW, pp 349-363, 2000.

[11] H. Casanova, J. Hayes, Y. Yang. *Algorithms and Software to Schedule and Deploy Independent Tasks in Grid Environments*. Workshop on Distributed Computing, Metacomputing, and Resource Globalization. Aussois, France. December 2002.

[12] H. Casanova and F. Berman. *Parameter Sweeps on the Grid with APST.* In [7], April 2003.

[13] K. Czajkowski et al. *A Resource Management Architecture for Metacomputing Systems*. JSSPP'1998, pp. 62-82, 1998.

[14] K. Czajkowski et al. *From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring and Extension*. http://www-106.ibm.com/developerworks/library/ws-resource/ogsi_to_wsrf_1.0.pdf

[15] W. Elwasif, J. Plank, R. Wolski. *Data Staging Effects in Wide Area Task Farming Applications*. IEEE International Symposium on Cluster Computing and the grid, pp. 122-129, May 2001,

[16] J. Frey et al. *Condor-G: A Computation Management Agent for Multi-Institutional Grids.* 10th HPDC, August 7-9, 2001.

[17] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[18] Global Grid Forum. *Grid Scheduling Dictionary of Terms and Keywords*. http://www.fz-juelich.de/zam/RD/coop/ggf/sched-sd.html

[19] Globus Web Site. http://www.globus.org

[20] N. Leibowitz, M. Ripeanu, A. Wierzbicki. *Deconstructing the KaZaA Network*. 3rd IEEE Workshop on Internet Applications, June 2003.

[21] M. Litzkow, M. Livny, and M. Mutka. *Condor: A Hunter of Idle Workstations.* 8th ICDCS, 1988.

[22] M. Netto, C. De Rose. *CRONO: A Configurable and Easy to Maintain Resource Manager Optimized for Small and Mid-Size GNU/Linux Cluster.* 32nd ICPP'03, pp. 555-562, 2003.

[23] D. Paranhos, W. Cirne, F. Brasileiro. *Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids*. Euro-Par 2003.

[24] E. Santos-Neto, W. Cirne, F. Brasileiro, A. Lima. *Exploiting Replication and Data Reuse to Efficiently Schedule Data-intensive Applications on Grids*. 10th JSSPP, June 2004.

[25] S. Saroiu, P. Gummadi, S. Gribble. *A Measurement Study of Peer-to-Peer File Sharing Systems*. MMCN'02, Jan 2002

[26] L. Sarmenta. *Sabotage-Tolerance Mechanisms for Volunteer Computing Systems*. Future Generation Computer Systems, 18:4, Elsevier, 2002.

[27] SETI@home Statistics Page. http://setiathome.ssl.berkeley.edu/totals.html

[28] S. Smallen et al. *Combining Workstations and Supercomputers to Support Grid Applications: The Parallel Tomography Experience*. HCW'2000.

[29] S. Smallen, H. Casanova, and F. Berman. *Applying Scheduling and Tuning to On-line Parallel Tomography*. Supercomputing'2001, Nov. 2001.

[30] W. Smith, I. Foster, V. Taylor. *Scheduling with Advanced Reservations*. IPDPS'2000.

[31] J. R. Stiles et al. *Monte Carlo Simulation of Neuromuscular Transmitter Release Using MCell, a General Simulator of Cellular Physiological Processes*. Comp. Neuroscience, pp. 279-284, 1998.

[32] S. Tuecke et al. *Open Grid Services Infrastructure (OGSI) Version 1.0*. Global Grid Forum Draft Recommendation, 6/27/2003. http://www.globus.org/research/papers.html

[33] L. Yang, J.M. Schopf, I. Foster. *Conservative Scheduling: Using Predicted Variance to Improve Scheduling Decisions in Dynamic Environments*. Supercomputing 2003, Nov. 2003.