Running Bag-of-Tasks Applications on Computational Grids: The MyGrid Approach

Walfredo Cirne *Daniel Paranhos* *Lauro Costa*

Elizeu Santos-Neto *Francisco Brasileiro* *Jacques Sauvé*

Universidade Federal de Campina Grande

{walfredo,danielps,lauro,elizeu,fubica,jacques}@dsc.ufcg.edu.br

Fabrício Alves Barbosa da Silva

UniSantos – Universidade Católica de Santos

fabricio@unisantos.br

Carla Osthoff Barros

Laboratório Nacional de Computação Científica

osthoff@lncc.br

Cirano Silveira

Hewlett Packard

cirano_silveira@hp.com

We here discuss how to run Bag-of-Tasks applications (those parallel applications whose tasks are independent) on computational grids. Bag-of-Tasks applications are both relevant and amendable for execution on grids. However, few users currently execute their Bag-of-Tasks applications on grids. We investigate the reason for this state of affairs and introduce MyGrid, a system designed to overcome the identified difficulties. MyGrid provides a simple, complete and secure way for a user to run Bag-of-Tasks applications on all resources she has access to. Besides putting together a complete solution useful for real users, MyGrid embeds two important research contributions to grid computing. First, we introduce some simple working environment abstractions that hide the configuration heterogeneity of the machines that compose the grid from the user. Second, we introduce Work Queue with Replication (WQR), a scheduling heuristics that attains good performance without relying on information about the grid or the application, although consuming a few more cycles. Note that not depending on information makes WQR much easier to deploy in practice.

1 Introduction

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 [26], and computer imaging [23] [24]. 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.

However, few users of BoT applications are currently using computational grids, despite the potential dramatic increase in resources grids can bring to bear for problem resolution. We believe that this state of affairs is due to (i) the complexities involved in using grid technology, and (ii) the slow deployment of existing grid infrastructure. Today, one must commit considerable effort to make an application run efficiently on a grid. The user, who is ultimately interested in getting the application's results, seldom has the training or the inclination to deal with the level of detail needed to use current grid infrastructure. Furthermore, the existing grid infrastructure is not ubiquitously installed yet. Users often have access to resources that are not grid-ready.

In this paper we present MyGrid, a system designed to change this state of affairs. MyGrid aims to *easily* enable the execution of BoT application on *whatever resources the user has available*. MyGrid chooses a different trade-off compared to existing grid infrastructure. It forfeits supporting arbitrary applications in favor of supporting only BoT applications (which are relevant and amenable to execution on grids). By focusing on BoT applications, MyGrid can be kept simple to use, simple enough to be a solution for *real users*, who want to run their applications *today* and don't really care for the underlying grid support they might use.

This is not to say, however, that MyGrid is a replacement for existing grid infrastructure. MyGrid uses grid infrastructure whenever available. It simply does not depend on it. MyGrid can be seen more as a representative of the user in the grid. It provides simple abstractions through which the user can easily deal with the grid, abstracting away the non-essential details (as we shall see in Section 4). It schedules the application over whatever resources the user has access to, whether this access is through some grid infrastructure (such as Globus [18]) or via simple remote login (such as `ssh`). MyGrid's scheduling solution is a particularly interesting contribution because it uses task replication to achieve good performance relying on no information about the grid or the application (as we will see on

Section 5). Note that not needing information for scheduling simplifies MyGrid usage. MyGrid is open source software, available at http://dsc.ufcg.edu.br/mygrid.

This paper is structured as follow. Section 2 presents the goals that drove MyGrid's design. Section 3 describes the architecture created to achieve such goals. Sections 4 and 5 describe the major contributions of MyGrid to Grid Computing state of art. Section 4 describes MyGrid's working environment, which hides details of individual machines from the user. Section 5 presents MyGrid's scheduling strategy, which attains good performance without relying on information about the grid or the application. Section 6 describes MyGrid's implementation, whereas Section 7 presents some experiments that evaluate its performance in practice. Section 8 compares MyGrid against related work. Finally, Section 9 closes the paper with final remarks and delineation of future work.

2 Design Goals

We intend MyGrid to be a production-quality solution for users who want to execute BoT applications on computational grids today. Note that "today" implies that we cannot assume that some new software will be widely deployed. Our design goals were thus established with this vision in mind. We want MyGrid to be *simple*, *complete*, *encompassing*, and *secure*.

By *simple*, we imply that MyGrid should be as close as possible to an out-of-the-box solution. The user wants to run her application. The least she gets involved into grid details, the better. Towards this goal, we worked on minimizing the installation effort. This is important because if the user had to manually install MyGrid on many machines, the simplicity of the solution would suffer.

Complete means that MyGrid must cover the whole production cycle, from development to execution, passing through deployment and manipulation of input and output. This goal is key for MyGrid to be useful in practice. In order to support all activities within the production cycle of a BoT application, MyGrid provides the notion of *working environment*, which consists of a small set of abstraction that enables the user to manipulate her files on the Grid.

Due to their loosely coupled nature, BoT applications can potentially use a very large number of processors. Therefore, we do not want MyGrid to preclude the user from using a given processor. We want MyGrid to be *encompassing*, in the sense that all machines the user has access to can be utilized to run her BoT applications. An important consequence of this goal is that MyGrid must be an user-centric solution, i.e. MyGrid cannot assume that some given software is installed in a machine for it to be employed in the computation. That is, we cannot assume that a particular grid infrastructure (e.g. Globus [18]) is ubiquitously installed on all machines a user has access to. Note also that, simplified installation also helps here. It does not suffice that the user potentially can employ all machines she has access to. It has to be simple to do so.

MyGrid should not weaken the user's security; otherwise it probably will not be used. Consequently, we want MyGrid to be *secure* in the sense that the user's security will not be compromised by a security problem in one of the many machines that compose the user's Grid.

3 Architecture

We assume the user has a machine that coordinates the execution of BoT applications through MyGrid. This machine is called *home machine*. The user submits the tasks that compose the application to the home machine, which is responsible for farming out the tasks in the user's grid. We assume that the user has good access to her home machine, having set up a comfortable working environment on it. Moreover, the user has no problems installing software on the home machine. We envision that the home machine will oftentimes be the user's desktop.

The home machine schedules tasks to run on *grid machines* (see Section 5 for a discussion of scheduling in MyGrid). In contrast to the home machine, we assume that grid machines have not been wholly customized by the user to create a familiar working environment. Moreover, grid machines do not necessarily share file systems with the home machine, nor do they have the same software installed on them (even the operating system can differ). Ideally, the user does not want to treat grid machines individually. For example, the user should not have to install MyGrid software on them. The idea is that grid machines are dealt with through MyGrid abstractions (see Section 4).

Note that the differentiation between home and grid machines makes our design goals (see Section 2) more concrete. Our security concerns, for example, can be phrased as "compromising a grid machine should not give the intruder the ability to compromise the home machine". Also, easy installation applies solely to grid machines. On the home machine, MyGrid is software as any other and must be installed and configured. Likewise, simplicity implies that grid machines are dealt with through abstractions that hide their heterogeneity in hardware, software and configuration.

However, enabling the user to benefit from "whatever resources she has access to" is essentially impossible in the sense that we do have to assume something about a grid machine in order to be able to use it. Therefore, our design tries to get as close as possible to this goal. This is done by defining the *Grid Machine Interface* as the minimal set of services that must be available in a machine for it to be used as a grid machine. These services are:

Service
Task start-up on a grid machine (remote execution)
Cancellation of a running task
File transfer from the grid machine to the home machine
File transfer from the home machine to the grid machine

Table 1 – Grid Machine Interface

As illustrated by Figure 1, there can be many ways to implement the Grid Machine Interface. Actually, Grid Machine Interface is a *virtualization* [19] of the access services to a grid machine. One way to implement the Grid Machine Interface lets the user furnish MyGrid with scripts that implement the four services listed in Table 1. In this case, MyGrid uses the *Grid Script* module to access the machine. Note that Grid Script enables the user to inform MyGrid on how to access a given machine in a very flexible way. As long as the user is able to translate "having access to a given machine" into "providing scripts that encapsulate such access", MyGrid will be able to use the machine.

Other ways to implement the Grid Machine Interface rely on access methods that are known to MyGrid. For example, we are currently implementing *MyGrid's Globus Proxy*. The idea is that if a grid machine can be accessed through Globus' GSI, GRAM, and GridFTP [18], then the user does not need to supply scripts, but simply indicate that the access is to be done via MyGrid's Globus Proxy. MyGrid also provides its own access mechanism, called *User Agent*. The User Agent is useful when no service that implements the operations described by the Grid Machine Interface (see Table 1) is available.

In addition to hiding the grid machine access mechanism from the scheduler, the Grid Machine Interface provides a clean and elegant way (i) to deal with communication restrictions, and (ii) to support space-shared machines (i.e. parallel supercomputers).

Communication restrictions are a fact in today's Internet [25], where firewalls and private IP addresses are commonplace. Communication restrictions have great practical impact on MyGrid because sometimes the home machine cannot freely communicate (i.e. open a TCP socket) with a grid machine. We deal with this problem by using the *Grid Machine Gateway*. The Grid Machine Gateway is an application-level relay. It runs on a machine accessible to both home and grid machines (e.g. in a NAT converter) and forwards Grid Machine Interface services requests (see Table 1). In design pattern terminology [20], the Grid Machine Gateway is a *decorator* in the sense that it implements and uses the interface, namely the Grid Machine Interface.

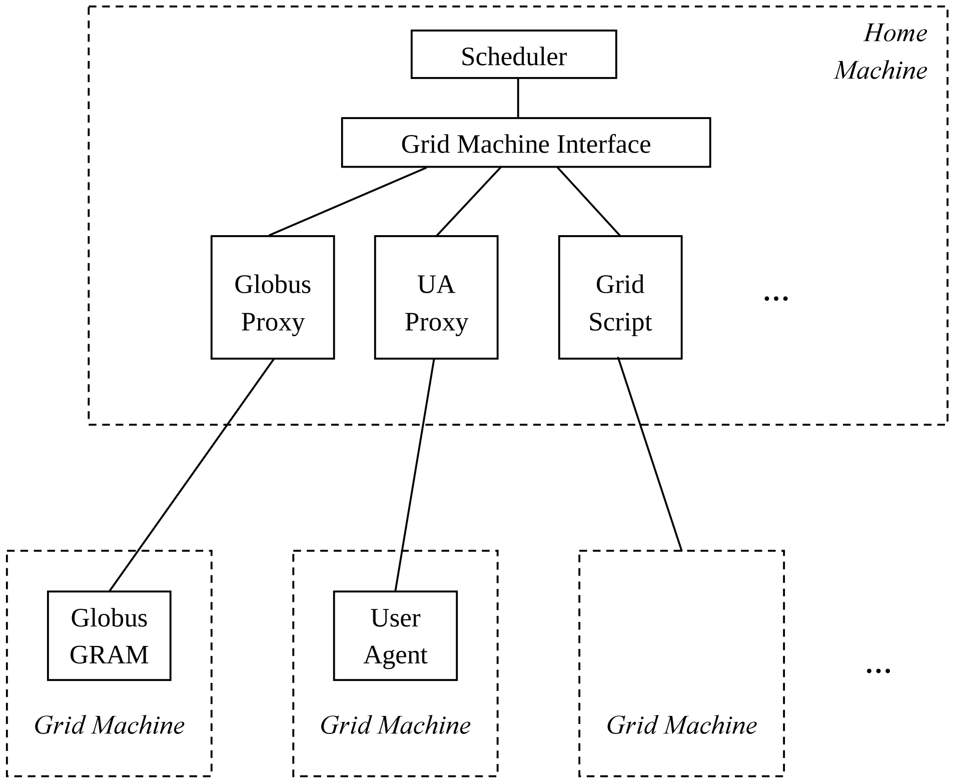

Figure 1 – Implementations of the Grid Machine Interface

Space-shared machines (such as parallel supercomputers) are powerful resources the user might have access to. Using a space-shared computer, however, is different than using a time-shared workstation. In order to run a job in a space-shared computer, the user typically must submit a job request specifying how many processors are needed and the time they are to be available for the job. When no resources are currently available to fulfill the request, it sits in a wait queue until it can run.

Having to deal with two kinds of resources would complicate scheduling. In particular, directly submitting requests to a space-shared machine would imply in crafting good requests (number of processor and execution time), a non-trivial task [11]. We introduced the *Space-Shared Gateway* to address this issue. Whenever MyGrid needs space-shared processors, the Space-Shared Gateway submits the largest request that can be fulfilled immediately, a technique introduced in [23]. Instead of submitting application's tasks, however, the Space-Shared Gateway submits instances of the User Agent, much like Condor-G's GlideIn [16]. The scheduler then farms out tasks on the space-shared processors via the User Agent. In short, the Space-Shared Gateway transforms space-shared resources in timed-shared resources that appear and leave the system, simplifying the overall system design. We should also mention that the Space-Shared Gateway is often used in conjunction with the Grid Machine Gateway. This happens because the processors of a space-shared machine typically cannot directly communicate with the outside world.

Figure 2 exemplifies a small but somewhat complex grid. The grid in question is composed by 18 machines, 9 of which are directly accessed by the home machine. The other 9 machines cannot be directly accessed. Communication to those machines is achieved with the help of the Grid Machine Gateway. Of the 9 directly accessed grid machines, 4 employ User Agent to implement the Grid Machine Interface, 3 use Globus Proxy, and 2 use Grid Script. Eight of the machines accessed via Grid Machine Gateway utilize User Agent. The other machine uses Grid Script. Note also that 5 machines accessed via Grid Machine Gateway are part of a space-shared machine and thus have their User Agent started by the Space-Shared Gateway.

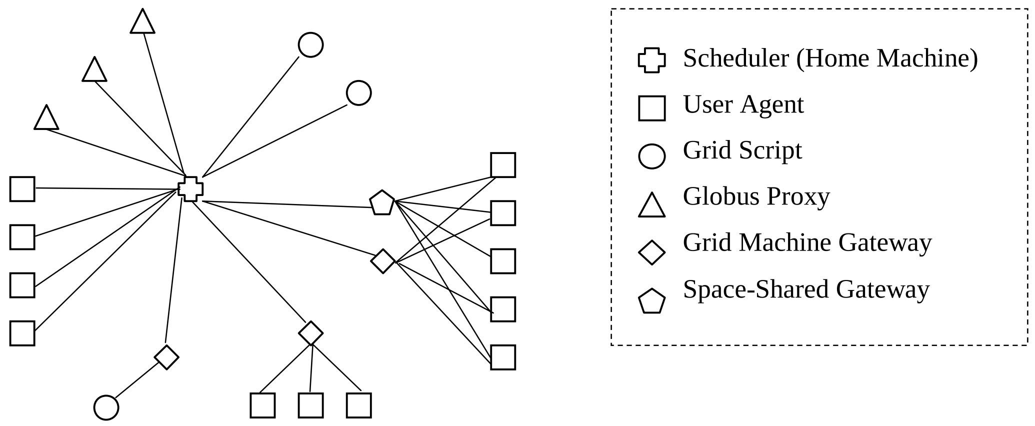

Figure 2 – Example of a MyGrid

One salient feature of the architecture presented here is that the home machine is a centralized component of the system. Therefore, concerns about the scalability of our design arise naturally. Our hope is that, despite the centralization of the home machine, MyGrid is going to be able to efficiently support most BoT applications. Such a hope is based (i) on experiments we performed (see Section 6), and (ii) on the fact that Condor, a successfully and mature system for the execution of BoT applications, also uses centralized job control [21]. But, of course, we intend to eventually remove such an architectural restriction from our design. The Grid Machine Gateway would be a natural place for leveraging towards a more scalable design. The challenge is to keep the view the user has from the system.

4 Working Environment

The user needs a grid-wide *working environment*, i.e. a set of abstractions that enable her to conveniently use her grid, in the same way that files and processes make it convenient to use a single computer. MyGrid's working environment provides a common denominator that users can rely upon when programming for grids, despite differences in the configuration of the multiple resources that comprise the grid. Moreover, a working environment is crucial in providing a *complete* solution, one that eases managing input and output files, distributing application code, and otherwise carrying on daily computational activities, now based on a computational grid.

A MyGrid task is formed by *initial*, *grid*, and *final* subtasks, which are executed sequentially in this order. Subtasks are external commands invoked by MyGrid. Consequently, any program, written in any language, can be a subtask. The initial and final subtasks are executed on the home machine. The initial subtask is meant to set up the task's environment by, for instance, transferring the input data to the grid machine. The final subtask is typically used to collect the task's results back to the home machine. The grid subtask runs on a grid machine and performs the computation per se. Besides its subtasks, a task definition also includes the *playpen size* and the *grid machine requirements*, as we shall shortly see.

MyGrid abstractions allow for writing the subtasks without knowing details about the grid machine used (such as file systems organization). The abstractions are *mirror*, *playpen*, and *file transfer*. Mirroring enables replication of home machine files on grid machines. Mirrored files are put in the directory `$MIRROR`, which is defined by MyGrid taking into account the local file system organization. Therefore, a grid subtask refers to mirrored file `F` through `$MIRROR/F`, without having to know details about the grid machine file system. Mirroring is useful for distributing files that are going to be used more than once, such as program binaries. In fact, to ease mirroring binaries, `$MIRROR` is automatically included in the `PATH` by MyGrid. Mirroring is implemented efficiently by using the modification date and a hash of mirrored files, avoiding unnecessary file transfers.

Playpens provide temporary disk space independently of the local file system arrangements of a given grid machine. Playpens are directories created automatically to serve as the working directory of the grid subtask. Besides the initial working directory, a grid subtask can also refer to its playpen via the `$PLAYPEN` environment variable.

MyGrid creates the playpen in a file system that can hold the amount of data specified by the user as the task's *playpen size*. (If there is no such file system in a given grid machine, the task cannot be scheduled to this machine.) Unlike mirroring, playpens are meant to store temporary files as well as input and output data.

Note that the name playpen makes greater sense from the grid machine viewpoint. We envision that the playpen implementer may want to protect the grid machine from a malicious task. This could be done by isolating the files that can be accessed, perfectly matching the playpen abstraction.

File transfer allows for sending files between grid machines and the home machine. They are typically used for the initial subtask to send input data to the playpen, and for the final subtask to collect output data from the playpen. In order to ease writing the initial and final subtasks, MyGrid automatically defines the environment variables `$PROC`, `$PLAYPEN` and `$TASK`. They respectively contain the grid machine chosen to run the task, the directory created as the playpen, and the unique task number.

For example, suppose we want to run the binary `tarefa`, which has the file `ENTRADA` as input and the file `SAÍDA` as output. The initial subtask would then be:

```
mg-services mirror $PROC tarefa
mg-services put $PROC ENTRADA $PLAYPEN
```

The grid subtask would be simply:

```
tarefa < ENTRADA > SAÍDA
```

And the final subtask would collect `SAÍDA` to the `results` directory, renaming the file by appending the unique task number to its name.

```
mg-services get $PROC $PLAYPEN/SAÍDA results/SAÍDA-$TASK
```

Appending the task number to a file is useful for the quite common case where the tasks that compose the application produce output with the same name. Appending the task number ensures the uniqueness of each output.

The final component of a task is its *grid machine requirements*. In MyGrid, grid machines are labeled by *attributes*. Attributes are user-defined strings that express the characteristics of a grid machine. For example, the user can assign attributes `linux` and `lsd` to `lula.dsc.ufcg.edu.br` to denote that such a machine runs Linux and is located at LSD (Laboratório de Sistemas Distribuídos). A task's grid machine requirement consists of a boolean expression involving grid machines attributes. For example `linux and not lsd` denotes machines that have been labeled with the attribute `linux` but not with the attribute `lsd` (`lula.dsc.ufcg.edu.br` would not qualify).

Any subtask can also determine the attributes of a grid machine. This makes it possible for the subtasks to adapt to different kinds of grid machines. Refining the above example, suppose that tarefa has binaries for Linux and Solaris, placed respectively at linux and solaris directories. Assuming that the user has labeled each grid machine with its operating system, the initial subtask could then use attributes to mirror the right binary.

```
if mg-services attrib $PROC linux; then
        mg-services mirror $PROC linux/tarefa
else
        mg-services mirror $PROC solaris/tarefa
endif
mg-services put $PROC ENTRADA $PLAYPEN
```

5 Scheduling

Another key component of MyGrid is the Scheduler. The Scheduler receives from the user the description of the tasks that compose the application, chooses which machine runs each task, submits the tasks for execution, and monitors their progress. However, scheduling BoT applications on grids is not as easy as it might look at first. Good scheduling requires good information about the tasks that compose the application and the capabilities of grid resources. Requiring information about tasks (such as expected execution time) would make MyGrid harder to use. Information about grid resources (such as speed and load) is often difficult to obtain due to the grid's distributed and multi-institutional nature. Moreover, we would need a richer definition of the Grid Machine Interface (see Table 1) in order to obtain information about grid resources. A richer definition of the Grid Machine Interface would of course be harder to implement, negatively affecting our goal of using whatever resources the user has access to.

An alternative is to use a scheduler that does not rely on information *about* tasks or resources, such as *Workqueue*. Workqueue only uses information absolutely necessary to schedule the application, namely the tasks to run, and the processors available to execute the tasks. In Workqueue, yet-to-execute tasks are chosen in an arbitrary order and sent to the processors, as soon as they become available. After the completion of a task, the processor sends back the results and the scheduler assigns a new task to the processor. That is, the scheduler starts by sending a task to every available host. Once a host finishes its task, the scheduler assigns another task to the host, as long as there are still tasks to be executed. Unfortunately, knowledge-free schedulers (as Workqueue) do not attain performance comparable to schedulers based on full knowledge about the environment (provided that these schedulers are fed with good information) [22].

We developed the *Workqueue with Replication* (WQR) algorithm to deal with this problem. WQR delivers good performance without using any kind of information about the resources or tasks. Initially, WQR behaves as the conventional Workqueue. The difference appears when there are no more tasks to execute. At this time, a machine that finishes its tasks would become idle during the rest of the application execution in Workqueue. Using replication, such a machine is assigned to execute a replica of an unfinished task. Tasks are replicated until a predefined maximum number of replicas is achieved. When a task is replicated, the first replica that finishes is considered as the valid execution of the task and the other replicas are cancelled. Of course, WQR assumes that tasks are idempotent, i.e. can be re-executed with no side effects. Since MyGrid's abstractions encourage the use of file transfer to deal with input and output, this assumption seems appropriate.

WQR minimizes the effects of the dynamic machine load, machine heterogeneity and task heterogeneity, and does so without relying on information on machines or tasks. It improves performance in situations where tasks are delaying the application execution because they were assigned to slow hosts. When a task is replicated, there is a greater chance that some replica is assigned to a fast host. A way to think about WQR is that it trades off additional CPU cycles for the need of information about the grid and the application. Moreover, BoT applications often use cycles that would otherwise go idle [21]. Thus, trading CPU cycles for the need of information can be advantageous in practice.

We investigated the performance of WQR in a variety of scenarios, in which we varied the granularity of the application, the heterogeneity of the application, and the heterogeneity of the grid [22]. Table 2 summarizes the results of 7,885 simulations. Sufferage and Dynamic FPLTF (Fastest Processor to Largest Task First) are known scheduling algorithm that were fed with perfect knowledge in the simulations. The qualifier to WQR (i.e. 2x, 3x, 4x) denotes the maximum replication allowed. For example, WQR 2x allows only 2 replicas of each task (or the original and the replica, if you will).

		Sufferage	**Dynamic FPLTF**	**Workqueue**	**WQR 2x**	**WQR 3x**	**WQR 4x**
Execution time (sec)	**Mean**	13530.26	12901.78	23066.99	12835.70	12123.66	11652.80
	Std Dev	9556.55	9714.08	32655.85	10739.50	9434.70	8603.06
Wasted CPU (%)	**Mean**	N/A	N/A	N/A	23.55	36.32	48.87
	Std. Dev	N/A	N/A	N/A	22.29	34.79	48.94

Table 2 – WQR simulation results

Overall, the performance of WQR appeared to be equivalent to solutions that have perfect knowledge about the environment (which is not feasible to obtain in practice), even when we were limited to two replicas of each task. In average, the wasted CPU varied from 23.5% (when using only 2 replicas) to 48.9% (when using 4 replicas, the maximum we tried).

Note also that the high values of the standard deviation suggest a great deal of variability in the results. In fact, we found that application granularity (the relation machines by tasks) has a strong impact on the results. As can be seen in Figure 3, WQR attains good performance, except for applications with very large granularity (where there are more machines than tasks). The difficulty faced by WQR when there are more machines than tasks is that the application execution takes only one "round" of tasks to processors assignments. Therefore, assigning a large task to a slow machine has great impact on the application execution time. Algorithms based on knowledge about the application and the grid (Sufferage and Dynamic FPLTF) can avoid such bad assignments. Replication helps (as can be seen comparing WQR to Workqueue) but not enough to overcome the problem totally. Not surprisingly, much more cycles are wasted when only a single round to tasks to processors assignments, as shown by Figure 4. Consequently, when there are more machines than tasks, one might want to limit cycles waste by limiting replication to 2 (i.e. by using WQR 2x). Performance is still reasonable (although not as good as what achieved by scheduling algorithms with perfect knowledge of the application and the grid). Application and grid heterogeneity also influence WQR performance, but to a smaller degree. We refer the reader to [22] for a complete performance analysis of WQR.

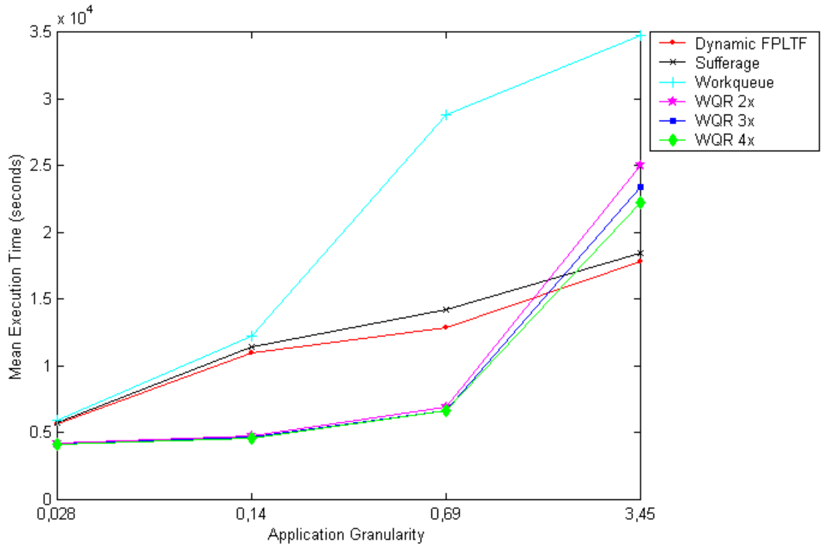

Figure 3 – Execution time by application granularity

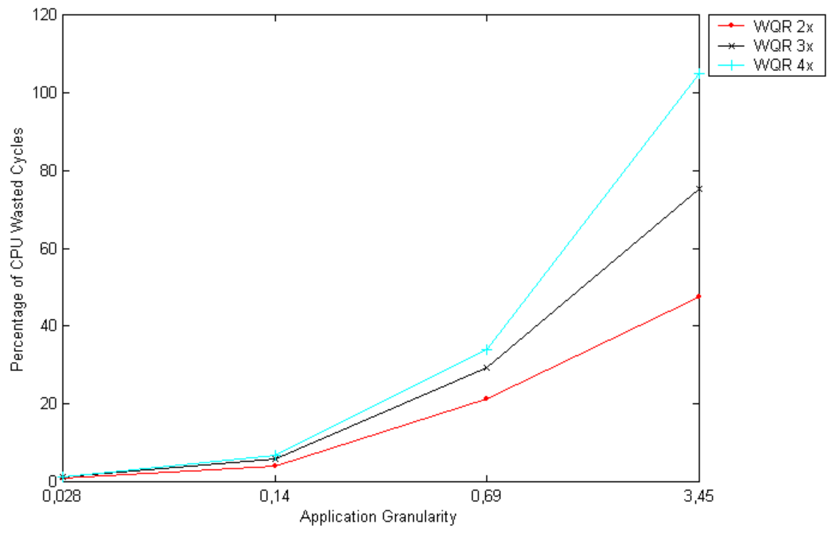

Figure 4 – Wasted cycles by application granularity

6 Implementation

MyGrid implementation consists of two major components: the Scheduler (which runs on the home machine) and the User Agent (one of the implementations of the Grid Machine Interface). Both were written in Java and using XP as development methodology [3].

There is not much to say about the implementation of the Scheduler. The only important point that comes to mind is that we found it very hard to write automated tests for the Scheduler. This is because the Scheduler is multi-threaded and its actions must be carried out in a distributed system, therefore introducing non-determinism in the Scheduler execution and making automated testing much harder.

The implementation of the User Agent provided a much richer experience. In short, we found out that it is nowadays very hard to develop global distributed applications. Major problems are heterogeneity (which makes installation and portability very tough) and lack of point-to-point connectivity (due to widespread firewalls and private IP addresses).

The User Agent is implemented in Java, which certainly helped to make it portable. However, being written in Java makes the User Agent dependent on the Java Virtual Machine (JVM). This is an annoyance because we find machines with no JVM installed, JVM not present on the `PATH`, and obsolete JVMs. We are now working on adding JVM detection (and eventual installation) as part of the User Agent's self-installation process.

The Scheduler and the User Agent communicate via Remote Method Invocation (RMI), what brought two concerns. First, we needed a way to authenticate the invoker of remote methods, otherwise anyone could invoke the services defined in the Grid Machine Interface, performing remote execution and file transfers as the MyGrid user. We closed this security breach by using Java's own public key authentication within RMI's socket factory. Second, we avoided using `rmiregistry` in order to make it easier configuring firewalls to let MyGrid traffic pass. To access an RMI service, one needs to contact two TCP ports, the well-known `rmiregistry` port, which informs the service port, and then the service port itself. But we found firewall managers somehow more reluctant in opening two ports than opening one. (Maybe that is because most services only need a single port, and hence asking to free traffic for two ports sounds suspicious.) We addressed this by writing our own implementation of `UnicastRemoteObject`, which can be contacted directly by providing only machine IP and port number. (Many say "Computer Science is the science that solves problems by adding indirection levels". This is the first time we see something being solved in Computer Science by *removing* an indirection level.)

It is also worth mentioning that we found Grid Scripts to be a source of headaches. The problem is related to the lack of strongly-typed interfaces in human-oriented commands. The interfaces of commands like `ssh` are designed for software-human interaction, not for software-software interaction. It is common to find different versions of the same command with subtle variations. A human would have no problem in understanding these differences (for instance, a `-t` option becoming `-T`), but this is enough to break our scripts.

Without standard strongly-typed software-oriented interfaces, this problem is very hard to solve completely. We tried to minimize it by using self-installation. (Actually, this very problem was a major motivation for self-installation.) Self-installation consists of using Grid Scripts solely to install the User Agent (i.e. to transfer the software to the grid machine and start it remotely). This is not always possible (e.g. there might be a firewall that blocks User Agent traffic), but when it is, it reduces the problems caused by Grid Script, because it is much less used (only to bring the User Agent up).

7 Performance Evaluation

We ran three experiments to evaluate MyGrid's performance in practice. The first two experiments are proof of concept demonstrations. They consist of using MyGrid to run two real applications on real grids and show that MyGrid can be useful in practice. The first experiment ran a series of simulations designed to investigate how to leverage the fact that most supercomputer jobs are moldable to improve their performance [11]. The second experiment uses MyGrid to run multiple instances of THOR, a dynamic molecular application, and investigates the impact of AIDS drugs on regional mutants of the HIV-1 virus. The third experiment is different in nature in the sense that it does not execute a real application. Instead, it runs a benchmark application designed to investigate the performance bottleneck due to MyGrid's architecture, which is centralized around the home machine.

Simulation of Supercomputer Jobs

In 2000, we have run a large-scale experiment using Open Grid [10], a MyGrid predecessor that used traditional Workqueue (instead of WQR) for scheduling and had only Grid Script to access grid machines. We conducted around 600,000 simulations to study scheduling of moldable supercomputer jobs [11] using 178 processors located in 6 different administrative domains (4 at University of California San Diego, 1 at San Diego Supercomputer Center and 1 at Northwestern University). The processors were in normal production (i.e., they were not dedicated to us at any point in time). The processors were either Intel machines running Linux or Sparc machines running Solaris.

The 600,000 simulations took 16.7 days, distributed over a 40-day period (the remaining time was used to analyze the latest results and plan the next simulations). In contrast, our desktop machine (an UltraSparc running at 200MHz) would have taken about 5.3 years to complete the 600,000 simulations (had it been dedicated only to that task). Nevertheless, even more important than the achieved speed-up is the fact that we were able to use everyday machines located in different administrative domains as the platform for our application. In fact, we were required to run at lower priority in four of the administrative domains.

It is also important to stress that the machines we used shared no common software except ubiquitous Unix utilities such as `emacs`, `ssh`, and `gcc`. In particular, Grid Computing software (more precisely, Globus [18]) was only installed in a single administrative domain. Moreover, access mechanisms varied from one administrative domain to another. For example, one of the domains had machines with private IP (we accessed these machines via Grid Script using double `ssh`, i.e. `ssh <gateway> ssh <grid-machine> <command>`).

Fighting AIDS

MyGrid is being currently used by Paulo Bisch's group at UFRJ to support a research on how the protease of mutants of the HIV-1 virus interact with protease inhibitors (i.e. AIDS drugs). A single mutant × protease inhibitor interaction is performed using THOR, a package for modeling and molecular dynamics developed at UFRJ. Since many interactions must be tried whenever the biologist wants to examine a conjecture, the problem is amenable to run on MyGrid.

A typical execution involves around 50 10-minute tasks, where each task demands the transfer of around 4MB. For a simple test, we have submitted 60 tasks to MyGrid. Each task reads a 3.3 MB input file and writes a 200KB output file. The grid consisted of 58 processors, distributed throughout the following 7 independent computing sites:

- Carcara/LNCC/Petropolis/Brazil - 1 linux processor
- ApeLab/USCD/San Diego/USA - 20 solaris procecessors
- LSD/UFCG/Campina Grande/Brazil - 8 linux processors
- NACAD/UFRJ/Rio de Janeiro/Brazil - 1 linux processor
- NCE/UFRJ/Rio de Janeiro/Brazil - 4 linux processors
- LCP/UFRJ/Rio de Janeiro/Brazil - 14 linux processors
- GridLab/UCSD/San Diego/USA - 10 linux processors

There were no dedicated processors on the grid. Moreover, each task was submitted using `nice` (i.e. with low priority). Each task executed from 4 minutes at the fastest machine from the grid, up to 33 minutes, at the slowest machine on the grid. We performed a total of 20 executions. The total average execution time on the grid was 43 minutes.

Gauging the Home Machine Bottleneck

MyGrid architecture, although simple and effective, raises concerns about its scalability. Since the home machine is a centralized component, it will become a performance bottleneck sooner or latter. We here describe an experiment we devised to determine whether such a bottleneck would appear in the execution of a typical CPU-bound BoT application, composed of 100 tasks.

Our experiment used three administrative domains: LSD (at UFCG, with 8 machines), APE Lab (at UCSD, with 23 machines), and Grid Lab (at UCSD, with 12 machines). The home machine was at LSD. The home machine was dedicated to the experiment, but all grid machines were dealing with their normal loads. Three kinds of tasks were used: small, medium and large. Each task was a dummy loop, which received an input determining how long the loop would run. We set up the input such that the fastest machine in our grid (AMD Athlon XP 1800) would finish tasks in 10, 100 and 1000 CPU seconds, thus creating small, medium and large tasks. Note that tasks were essentially CPU-bound. File transfer was done only for the task code, whose size was of 4 KB.

Besides task size, we had three choices for the location of the grid machines, namely at LSD (i.e. in the same LAN as the home machine), at APE Lab, and at Grid Lab. The purpose of location variation was to understand MyGrid behavior in presence of different network connectivity and machine speed (the different domain have machines of different age and architecture, and thus different speed).

Combining *grid machine location* and *task size*, 9 different scenarios were investigated. For each scenario, we run a 100-task application 10 times. It is important to point out that the applications ran back-to-back in an attempt to give similar grid conditions to the different scenarios. That is, we would run an application for scenario 1, then scenario 2, ... then scenario 9 before repeating the executions 10 times.

In order to determine MyGrid performance in these scenarios, we defined a task's *efficiency* and *overhead.* Let i_t, g_t and f_t be the initial, grid and final execution times of a task, respectively. Consequently, the task total execution time t_t is given by $t_t = i_t + g_t + f_t$. Efficiency e denotes the fraction of the total execution that was effectively spent on the grid subtask, i.e. $e = g_t / t_t$. Overhead o is the time spent in the initial and final subtasks, i.e. $o = i_t + f_t$.

Figure 5 shows efficiency as a function of task size. As one would expect, due to the local connection between home and grid machines, efficiency at LSD was excellent. Also very good, efficiencies at Grid Lab and APE Lab display an interesting effect. APE Lab and Grid Lab have basically the same connectivity to LSD (where the home machine is located). However, Grid Lab machines are faster than APE Lab machines (as can be seen in Figure 5). Consequently tasks finish quicker when using Grid Lab resources, thus generating greater demand on the home machine, therefore reducing task efficiency.

Figure 6 shows the overhead results. For Grid Lab, overhead decreases somewhat as task size grows. For LSD and APE Lab, overhead appears to be very stable. These results indicate that MyGrid has not reached the home machine bottleneck for the scenarios here explored.

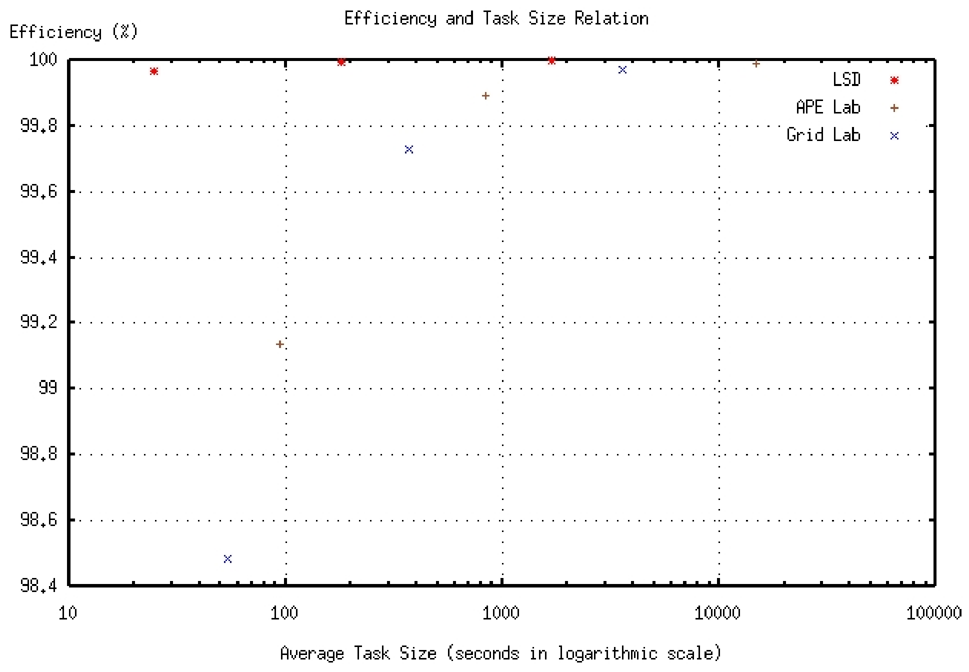

Figure 5 – Efficiency × Task Size

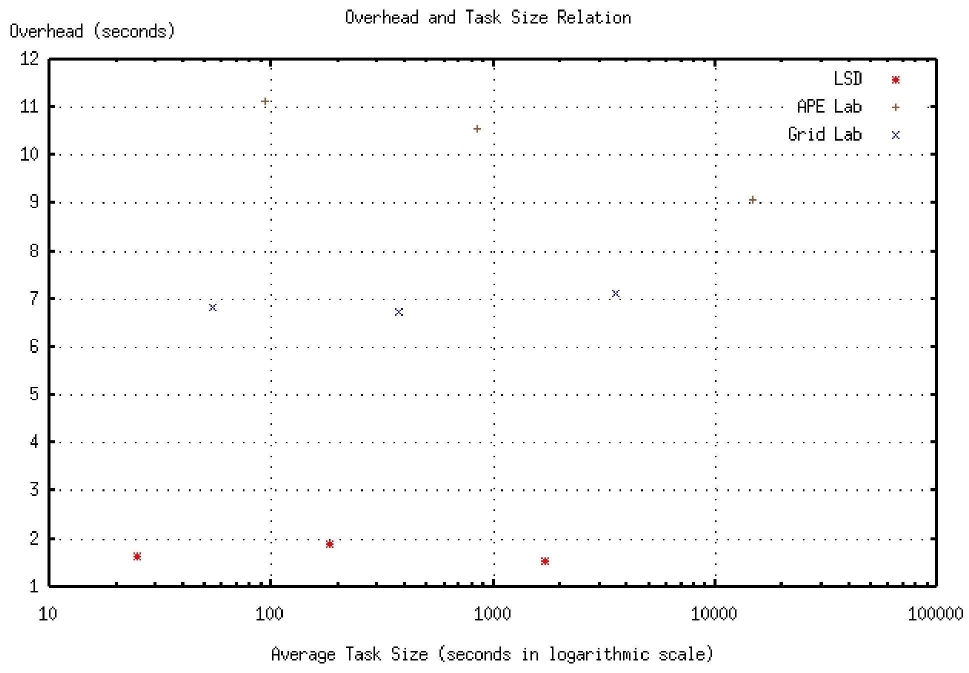

Figure 6 – Overhead × Task Size

However, an analysis of home machine's CPU usage during the experiment showed that CPU consumption peaked at 87.1% for small tasks when using LSD grid machines, indicating the bottleneck is close (at least for this scenario). A closer analysis at the CPU usage logs revealed that most of the CPU was spent starting new processes (especially the JVM). We then realized that our design decision of implementing the initial and final subtasks as invocations of external processes, while very flexible, has serious performance consequences. Since the initial and final subtasks run at the home machine, they not only have to be forked, but also need to fork a JVM every time they use a MyGrid service. For example, in the little initial subtask presented at the end of Section 4, there are 3 invocations of a JVM (not exactly a lightweight operation).

In virtue of this problem, we are currently working in providing an alternative to running initial and final subtasks as external processes. The idea is to embed in MyGrid a very simple script language to enable simple file transfer based on the processor's attributes, a functionality that should be enough for most applications.

After this modification, we are going to reassess when the centralization of the home machine functionality becomes a bottleneck. We hope that this point will not be reached by most applications. Nevertheless, there will be applications to be affected by the centralized home machine. To address this issue, we intend to investigate how to distribute the home machine functions to Grid Machine Gateways. The challenge is to do so without affecting the view the user has from the system.

8 Related Work

Grid computing is a very active area of research [17]. There are important efforts, both in academia and industry, that aim to provide generic software infrastructure for grid computing. In academia, the project with greatest impact and visibility is Globus [18]. In industry, Entropia [15] appears to have built the largest commercial grid so far.

Closer to our work, there are systems that target BoT applications, such as APST [7] [8], Nimrod/G [1] and Condor [16] [21]. In particular, APST and Nimrod/G are similar to MyGrid in intent and architecture. However, they are more specialized than MyGrid in the sense that they target parameter-sweep applications, a sub-class of BoT applications. Also, both APST and Nimrod/G require much more information than MyGrid for scheduling. Besides depending on rich information for scheduling, APST and Nimrod/G 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 task. Nimrod/G assumes that the user is going to pay for resources and hence scheduling is based on a grid economy model [5].

Condor was initially conceived for campus-wide networks [21], but has been extended to run on grids [16]. Condor's main difference to MyGrid is the fact that Condor is system-centric, whereas MyGrid is user-centric. That is, Condor is installed and configured by a system administrator who controls the resources that form the Condor Pool, which provides service for its users. In MyGrid, conversely, each user creates her own grid with all resources she has access to. One's MyGrid can thus use resources via Condor, Globus, Unix and other systems. Besides, Condor and MyGrid take different approaches regarding hiding grid machines' configuration heterogeneity from the users. Condor redirects systems calls to the home machine, creating the illusion that all tasks run at the home machine. MyGrid provides working environment abstractions that enable describing the tasks without knowing details about the configuration of the grid machines.

Other efforts that relate to MyGrid aim to scheduling BoT applications on grid and report experiences of running their applications. Casanova et al. describe a very interesting scheduling heuristics for BoT applications that process large amounts of data [6]. The proposed heuristics exploits the grid topology (i.e. the fact that a grid is composed by sites whose machines have local connectivity) to improve application's performance. Still on applications that process large amounts of data, Elwasif et al. investigate the impact of data staging (i.e. the previous transfer of data to selected places on the grid) on the performance of such applications [14].

Reports of the execution of BoT applications on grids include SETI@home [2] and GTOMO [23] [24]. SETI@home is probably the largest application to have run on grids. SETI@home does not separate the grid infrastructure from the application, whereas MyGrid focuses on the infrastructure, providing support for any BoT application. In fact, MyGrid can be thought as a framework to build SETI@home-like applications. GTOMO is a tomography application that runs over Globus and combines time- and space-shared machines to deliver good and consistent performance [23] [24].

9 Conclusions and Future Work

MyGrid is a complete solution for running Bag-of-Tasks (BoT) applications on all resources one has available. MyGrid represents a different trade-off in designing grid infrastructure. Traditional grid infrastructure, such as Globus [18], aims to support arbitrary applications. Arbitrary applications may of course have complex requirements.

In order to be flexible enough to deal with such potential complexity, traditional grid infrastructure leaves decisions and tasks to the user, making it hard to use the grid. That is, the grid infrastructure typically solves only part of the problem, leaving user to complement the solution by writing, for example, the application scheduler [4] [12]. MyGrid, on the other hand, is specific (provides support solely for BoT applications) but complete (the user does not have to add anything but the application itself). We hope that a complete solution will help to make grid technology more used in practice, to real users. MyGrid is open-source and can be downloaded from http://dsc.ufcg.edu.br/mygrid.

From the grid research viewpoint, MyGrid's main contributions are two-fold. First, we introduce a scheduling heuristics (denominated Work Queue with Replication) that achieves good performance without relying on information about the grid state or the tasks execution times. Having a scheduler algorithm that works well without information about the grid or the tasks is important because such information is hard to obtain in practice. Moreover, it keeps the system easy to use (by not requiring the user to provide task execution time estimates). Second, MyGrid hides the configuration heterogeneity of the machines that compose the grid by providing working environment abstraction that are easy to use and efficiently implemented on grids.

MyGrid has currently a dozen of real users and we learned a couple of lessons from working with our users. One lesson is that today's Internet connectivity is very complex and must be taken into account when designing any widely distributed system such as grid infrastructure. The nice end-to-end connectivity implied by socket abstraction no longer works in today's Internet [25]. Firewalls, gateways and private IPs are commonplace. This complexity has worked against our goal of creating a user-level grid infrastructure. We did our best to supply work-around solutions such as tunneling support and application gateways, but there are still situations where the user needs to involve the system administrator to make MyGrid work.

Another lesson was that, if we do not hear the user, we might create solutions for problems that do not present themselves in practice. For example, we have implemented the proportional-share ticket-based scheduler introduced in [9]. Such a scheduler is technically very interesting. It allows the user that runs more than one application simultaneously to define what fraction of her grid's resources should be allocated to each application. A bit to our dismay, however, so far none of our users found this capability useful.

As far as future work, we plan to pursue three lines of research and development. The first line is more developmental, but very important for the practical success of MyGrid. Since we experienced problem with Grid Script (out generic mechanism of accessing a grid machine), we intend to extend MyGrid to directly support other kinds of resources. OGSA [19] and vCluster [13] resources are our next targets. Condor would also be a natural choice to directly support.

Second, we would like to have more efficient support for BoT applications that process large amounts of data. Of course, MyGrid currently runs data intensive BoT applications, but it does not attempt to reduce the impact large data transfers have on the execution time of these applications. For example, MyGrid at this time does not exploit grid topology as in [6]. One challenge here is how to have better support for data intensive BoT applications without making the system harder to use and/or install (by making it dependent on information about grid state, for example).

Our third future effort comes as an answer to our user community. People are not satisfied just with good grid infrastructure. They actually want the resources on which to use the infrastructure, i.e. they want a grid. We hence intend to create OurGrid, a peer-to-peer resource sharing system targeted to BoT applications. OurGrid is thought as a favor-based community, where each peer offers access to its idle resources. In return, when there is work that exceeds local capacity, a peer expects to gain access to other people's resources. The challenge is doing so in a decentralized manner. A decentralized solution is key to keep OurGrid simple, not dependent on centralized services that might be hard to deploy, scale and trust.

Acknowledgments

We would like to thank Hewlett Packard and CNPq/Brazil for the financial support. This work could not be done without it. We would also like to thank Márcio Vasel, Nigini Abílio and Nazareno Andrade for their great work in different parts of MyGrid. Thanks to Allan Snavely for the insightful discussion that gave rise to the MyGrid name.

References

[1] D. Abramson, J. Giddy and L. Kotler. *High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?* IPDPS'2000, pp. 520-528, Cancun Mexico, IEEE CS Press, USA, 2000. http://ipdps.eece.unm.edu/2000/papers/Abramson.pdf

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

[3] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 1999.

[4] Fran Berman, Richard Wolski, Silvia Figueira, Jennifer Schopf, and Gary Shao. *Application-Level Scheduling on Distributed Heterogeneous Networks*. Supercomputing'96, Pittsburgh, 1996. http://www-cse.ucsd.edu/groups/hpcl/apples/hetpubs.html

[5] R. Buyya, D. Abramson, J. Giddy. *An Economy Driven Resource Management Architecture for Global Computational Power Grids*. The 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000), Las Vegas, USA, June 26-29, 2000. http://www.cs.mu.oz.au/~raj/ecogrid/

[6] H. Casanova, A. Legrand and D. Zagorodnov et al. *Heuristics for Scheduling Parameter Sweep Applications in Grid Environments*. In Proceedings of the 9th Heterogeneous Computing Workshop, pp 349-363, 2000. http://apples.ucsd.edu/hetpubs.html

[7] 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. http://grail.sdsc.edu/

[8] H. Casanova and F. Berman. *Parameter Sweeps on the Grid with APST*. In "Grid Computing: Making the Global Infrastructure a Reality", edited by Fran Berman, Geoffrey Fox and Tony Hey, John Wiley & Sons, April 2003. http://grail.sdsc.edu/

[9] W. Cirne e K. Marzullo. *The Computacional Co-op: Gathering Clusters into a Metacomputer*. Proceeding of the IPPS/SPDP'99 Symposium. April 1999. http://walfredo.dsc.ufcg.edu.br/resume.html#publications

[10] W. Cirne e K. Marzullo. *Open Grid: A User-Centric Approach for Grid Computing*. Proceedings of the 13th Symposium on Computer Architecture and High Performance Computing, September 2001 http://walfredo.dsc.ufcg.edu.br/resume.html#publications

[11] W. Cirne and F. Berman. *Using Moldability to Improve the Performance of Supercomputer Jobs*. Journal of Parallel and Distributed Computing, vol. 62, no. 10, pages 1571-1601, October 2002. http://walfredo.dsc.ufcg.edu.br/resume.html#publications

[12] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke. *A Resource Management Architecture for Metacomputing Systems*. Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, po. 62-82, 1998. http://www.globus.org/research/papers.html

[13] C. De Rose, F. Blanco, N. Maillard, K. Saikoski, R. Novaes, O. Richard, B. Richard. *The Virtual Cluster: a Dynamic Environment for Exploitation of Idle Network Resourses*. In 14th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'2002), 2002, Vitória, ES.

[14] W. Elwasif, J. Plank and R. Wolski. *Data Staging Effects in Wide Area Task Farming Applications*. IEEE International Symposium on Cluster Computing and the grid, Brisbane, Australia, May, 2001, pp. 122-129. http://www.cs.utk.edu/~plank/plank/papers/CC-GRID-01.html

[15] Entropia Web Page. http://www.entropia.com/

[16] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. *Condor-G: A Computation Management Agent for Multi-Institutional Grids*. Proc. 10th IEEE Symposium on High Performance Distributed Computing, HPDC'10, San Francisco, California, August 7-9, 2001. http://www.cs.wisc.edu/condor/publications.html

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

[18] I. Foster and C. Kesselman. *The Globus Project: A Status Report*. Proceedings of IPPS/SPDP'98 Heterogeneous Computing Workshop, pg. 4-18, 1998. http://www.globus.org/research/papers.html

[19] I. Foster, C. Kesselman, J. Nick, S. Tuecke. *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. June 22, 2002. http://www.globus.org/research/papers.html

[20] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison-Wesley Pub Co., 1995.

[21] M. Litzkow, M. Livny, and M. Mutka. *Condor: A Hunter of Idle Workstations*. In Proceedings of the 8th International Conference of Distributed Computing Systems, pages 104-111, June 1988.

[22] D. Paranhos, W. Cirne, and F. Brasileiro. *Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids*. Submitted for publication. http://walfredo.dsc.ufcg.edu.br/resume.html#publications

[23] S. Smallen, W. Cirne and J. Frey et al. *Combining Workstations and Supercomputers to Support Grid Applications: The Parallel Tomography Experience*. Proceedings of the HCW'2000 - Heterogeneous Computing Workshop. 2000. http://walfredo.dsc.ufcg.edu.br/resume.html#publications

[24] S. Smallen, H. Casanova, and F. Berman. *Applying Scheduling and Tuning to On-line Parallel Tomography*. Proceedings of Supercomputing 01, Denver, Colorado, USA, November 2001.

[25] S. Son and M. Livny. *Recovering Internet Symmetry in Distributed Computing*. GAN'03 Workshop on Grids and Advanced Networks. Tokyo, Japan. May 12-15, 2003.
http://www.cs.wisc.edu/~sschang/shortGAN03.pdf

[26] J. R. Stiles, T. M. Bartol, E. E. Salpeter, and M. M. Salpeter. *Monte Carlo Simulation of Neuromuscular Transmitter Release Using MCell, a General Simulator of Cellular Physiological Processes*. Computational Neuroscience, pages 279-284, 1998.