Hiding Grid Resources Behind Brokers

Eliane Araújo[1], Walfredo Cirne[1], Gustavo Mendes[1]

[1]Departamento de Sistemas e Computação – Universidade Federal de Campina Grande (UFCG)
Paraíba – PB – Brazil

{eliane,walfredo,gustavo}@dsc.ufcg.edu.br

Abstract. *Grid computing is a relatively new and promising research area. Many problems regarding its deployment are still been studied and are not clearly solved yet. Scheduling on the grid is particular challenging since new aspects must be taken into account, such as grid size (which can be huge), heterogeneity of resources and their ownership by different entities. Globus Toolkit copes with the scheduling on the grid problem with a solution that involves resource managers and brokers. Our proposal is to use the broker itself to hide heterogeneous resources. This broker, exposed as a compliant OGSA/OGSI grid service, was able to access heterogeneous resources, including intermittent resources as well as GRAM/fork resources.*

1. Introduction

The use of computational grids as platform to execute parallel applications is a promising research area. The possibility to allocate unprecedented 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 attractive 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, which need to be addressed.

In particular, the area of *scheduling* faces entirely new challenges in grid computing. The grid is just too big for a single entity to control. Moreover, the resources that comprise a grid are owned by many different entities, rendering it administratively unacceptable that a single entity controls all resources. Therefore, the grid must have multiple entities that divide the responsibility of scheduling. In the current state of art, these entities are *resource managers* and *brokers* [9]. A *resource manager* controls a set of the resources that compose the grid. The only way to access a resource is submitting a request to its resource manager. A *broker* schedules applications on the grid. It does not control resources, however. Therefore, it must rely on the services provided by resource managers (or other brokers).

The distinction between resource managers and brokers has been made explicit in Globus [13], a *de facto* grid standard. In a Globus grid, all resource managers expose themselves to the grid as GRAM modules [9]. In fact, Globus GRAM can be thought as an interface[1] designed to isolate resource managers from brokers. The rationale is to hide the details of each resource manager from the brokers, allowing brokers to consider only one kind of resource manager in the grid.

In practice, however, resource managers seem to be too dissimilar to be hidden behind GRAM. For example, when using GRAM/LSF or GRAM/Easy, one must specify the number of processor and the time they must be available to the application. When using GRAM/fork, neither of these parameters is required. With GRAM/Condor, on the other hand, one must describe the job's needs as a "classified add". Therefore, in practice brokers end up having to know which kind of GRAM they are dealing with, which defeats part of the goal of isolating resource managers from brokers.

Moreover, we found it extremely hard to implement a GRAM module to give access to intermittent resources. An *intermittent resource* is computational resource that appears and disappears with no previous notice, have unknown and varying power and may return incorrect results. Although providing very weak service, intermittent resources can be very useful, as has been demonstrated by SETI@home [2] and other volunteer grids.

Here, we investigate an alternative to distinguish brokers from resource managers. Our proposal does not require brokers to use GRAM to communicate with the resources. The solution architecture is simple and makes it is much easier to access new types of resources then it would be with GRAM. The broker we implemented

[1] In the object-oriented meaning of the term *interface*.

extends MyGrid [6], a grid solution for Bag-of-Tasks applications. *Bag-of-Tasks (BoT) applications* are those parallel applications whose tasks are completely independent. Although simple, 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 [21], and computer imaging [19] [20].

Using Globus 3.x, we present MyGrid to the grid as an OGSI-compliant broker that specializes in BoT applications. MyGrid, besides using intermittent resources directly, is able to access GRAM/fork resources. We are currently working on enabling MyGrid to access spaced-shared GRAMs, such as GRAM/LSF and GRAM/Easy [7]. MyGrid is open source software and has a user base of around 40 users. MyGrid is available at http://www.ourgrid.org/mygrid.

2. Related Work

Grid computing is a very active area of research [5] [11]. 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 emerged from the merge between GridServices and WebServices technologies, as seen in standards like OGSI [22] and its successor WSMF [8]. These standards are currently being implemented by both academia and industry. Most notably, these standards are being implemented by Globus [13], maybe the project with greatest visibility in Grid Computing.

Regarding scheduling, Condor is likely the work closely related to ours. As MyGrid, Condor provides a complete solution for BoT applications, uses GRAM resources (via Condor/G [12]), and exports local resources to the grid (via GRAM/Condor). The main difference to MyGrid is exactly that Condor exports local resources via GRAM. MyGrid, on the other hand, uses resources directly, exporting only the service that can be obtained with them via its BoT broker interface.

3. MyGrid Broker

MyGrid has evolved from a personal grid enabling software to a high level service broker. First, MyGrid has come to solve a one man's problem: to run thousands of simulations in many machines distributed geographically and administratively. Later, this problem has shown to be a common one. More people come up with the same situation. So, it was decided to improve and distribute MyGrid solution.

MyGrid was initially conceived to be a complete solution to execute Bag-of-Tasks applications in a personal grid. By "personal grid", we mean all machines one has access to. By "complete solution", we mean that the solution is self-contained since it provides tools for building the grid infrastructure and offers services to distribute the application to the machines. These goals are achieved by the scheduling service and the abstractions.

MyGrid scheduling services uses its abstractions to farm out the user application to the grid machines. In MyGrid parlance, *grid machines* are those that compose the grid and execute the user's applications. In contrast, *home machine* refers to the machine where the broker itself is running.

A MyGrid application is a job composed by a set of independent tasks. The scheduler chooses a grid machine to execute each task. There are two scheduling heuristics available to MyGrid scheduler: WorkQueueWithReplication [16] and StorageAffinity [17]. The former is most suitable to CPU-intensive applications and the later to data intensive applications. Both uses replication in order to improve the overall application performance, since these heuristics does not take into account information about the environment and the application [6].

In order to ease the development of applications that run over the grid, MyGrid provides some abstractions. These abstractions virtualize grid resources, making them simple to use. *Playpen* and *storage* are abstractions that mean to hide the file system of the remote machine. *Playpen* is a temporary directory that, as it name suggests, circumvent a limited area to the application on the remote machine. It is possible to transfer and retrieve files to the *playpen*. Normally, a task is executed under this directory. The *storage* area is similar to the playpen, except that it is not a temporary directory. It can share data among executions of different tasks and even jobs.

Note that MyGrid does not control the grid machines of the grid. It simply uses them to assign tasks to be executed. Moreover, other entities or services can be used to acquire new machines, thus augmenting one's grid. This distinction became more clearly with the advent of the OurGrid [3]. MyGrid broker is one of the building blocks of OurGrid architecture.

4. MyGrid as a Grid Service

MyGrid can be accessed through an OGSI compliant grid service. This allows the MyGrid broker to be used as a high level scheduling service to the grid infrastructure that is based on this architecture specification. The exposed OGSI interface allows submitting BoT jobs and monitoring the status of the tasks as well as the use of each machine by the scheduler.

A set of requirements are necessary to create a grid service: (i) It must adhere to OGSI, which defines a minimum set of services a grid service must implement in order to support requirements demanded by OGSA; (ii) It must be deployed in a grid service hosting environment. The environment used to deploy MyGrid was the one provided by Globus Toolkit 3.0 [13].

In order to execute applications in a grid using MyGrid broker, the client needs to send it jobs and, optionally, machines to be used. The job and the grid are informed in description files. The status of the jobs and machines of the grid can be monitored by adequate methods.

5. Accessing Resources

As the current practice, we tried to use GRAM as the standard method to access resources [9]. However, there is no available GRAM implementation for intermittent resources. An *intermittent resource* is a computational resource that appears and disappears with no previous notice, has unknown and varying power and may return incorrect results. Intermittent resources are those assumed by most public computing initiatives, such as SETI@home [2]. In fact, it is amusing that one can obtain good performance and reliable results from such weak kind of resources. Good performance is assured by eager schedulers [4][14][16][17], which use task replication to tolerate computational power variability without relying on resource performance forecasts. Reliable results can be assured by massive replication [2], or by a (much more efficient) credibility-based strategy [18].

Anyhow, implementing a new GRAM module is the natural solution for the lack of GRAM implementation that gives access to intermittent resources. However, we found that hiding intermittent resources behind GRAM would be too much of a stretch. The weak guarantees provided by intermittent resources would lead brokers to treat this kind of GRAM in a totally differently way. We thus decided not to expose intermittent resources via GRAM. Instead, we hide intermittent resources behind a broker (i.e. the broker has non-Globus access to the intermittent resources). This design was motivated by the fact that intermittent resources are not useful in general. Intermittent resources are useful for applications that can trivially recover from the loss of a processor. BoT applications fit this requirement.

The first mechanism MyGrid used to access resources was User Agent (UA). User Agent acts in behalf of the client on the remote resource. As this should not be the only way to use a resource, we needed to define an interface to standardize the behavior of the various type of resource access. We realized that for a resource to be useful for BoT application, we need only three functionalities: to transfer files from base machine to remote machine, to run tasks in the remote machine and transfer files from remote machine to base machine. An interface is important because it unifies the way to access the resources, but, on the other hand, it can reduce the behavior of a given resource to what is essentially needed. Keeping this in mind, we defined the GridMachine interface (GuM) that exposes the behavior expected for a machine to be part of a grid. GuM interface has the following methods: run, transferTo, transferFrom, kill and ping. Kill is used to kill a task on the remote machine and ping verifies if the resource is up.

After we defined the GuM interface, we developed the GlobusGridMachine, a Globus proxy that implements GuM. In this way, we can use the GRAM resources that are available. GlobusGridMachine uses the GSI infrastructure [10] to authenticate the users and GridFTP to transfer files from base machine to remote machine and vice-versa. MyGrid was developed to use all resources the user has access to. In this way, if the user has a Globus grid, MyGrid can send tasks to them. Here, we must unify to concerns: MyGrid task is equivalent to a Globus job.

Note that to run jobs in remote machines via UA or GRAM we need some software previously installed such as UA daemon or Globus Toolkit, respectively. Since we intend to enable the use of "whatever resources a user has access to", we cannot take this for granted. Users often do not have specific software installed on their personal machines. With this vision in mind, we develop the GridScript, an implementation of GuM that just need scripts that describe how to transfer files and remotely execute commands. Thus, we translate "having access to a resource" into "having a command line way to access a resource". In practice, most user rely on ssh/scp for GridScript.

6 Results

Figure 1 shows the mean execution time of the executed tasks, whereas Figure 2 shows the mean overhead. It is easy to notice that GlobusGridMachine executed much slower then the other ones. As a first thought, we considered that this happens because Globus security infrastructure adds an extra overhead to the total execution time. However, GridScript also works under a security infrastructure, and, surprisingly it performed much better than Globus. In fact, GridScript performed almost as well as UserAgent, which does not currently employ strong cryptography methods to assure security (although nothing precludes it from implementing such methods). We conjecture that, since ssh/scp are very mature software, their cryptography implementation is much optimized.

In Figure 3 shows the efficiency of GuM's implementations. We measured different sizes of tasks and perceived that as granularity of the task grows, the GlobusGridMachine becomes more efficient. This behavior is explained because the overhead is amortized by the total time.

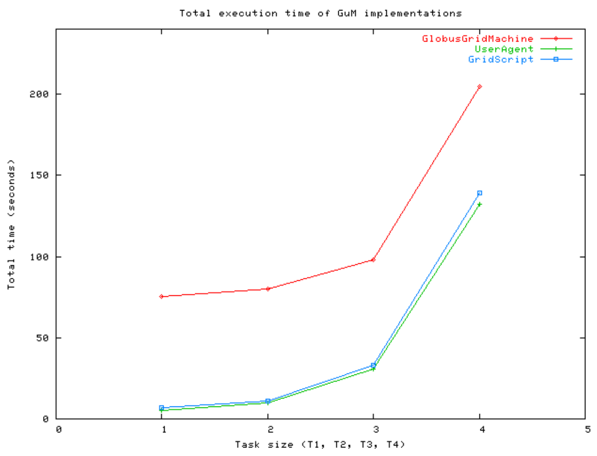

Figure 1 - Total time of execution

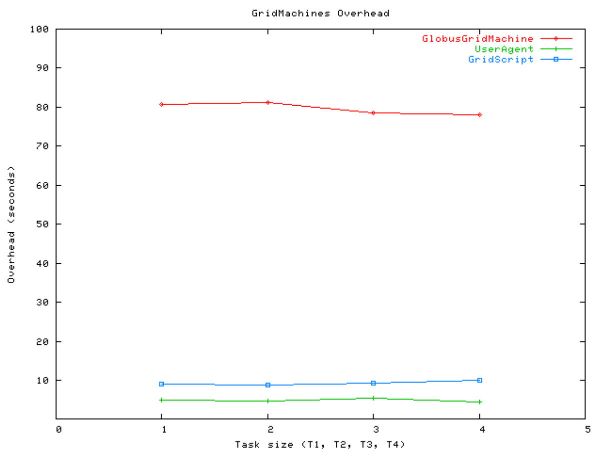

Figure 2 - Overhead of each GuM implementation

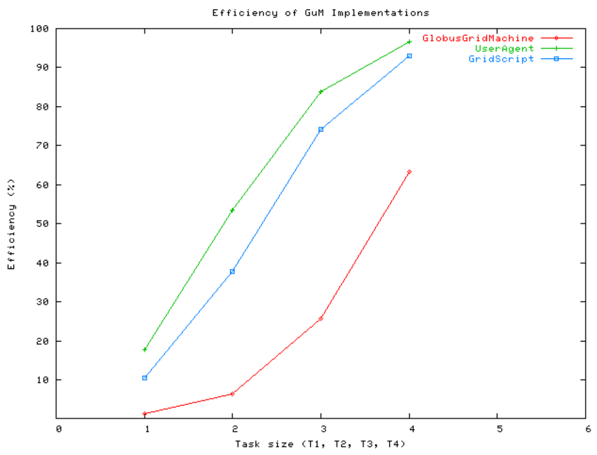

Figure 3 - Efficiency of the GuM's implementations

Anyhow, maybe even more important than gauging the overhead of each implementation, it is to show case that our architecture works as intended.

7. Conclusions

We here present an alternative way to integrate resources into a Globus grid. Instead of implementing the GRAM interface, we make these resources available via a high-level application-specific broker, effectively hiding these resources from the grid. The rationale behind our design is two-fold. First, the resources we export (intermittent resources) are considerably different from the resources traditionally made available via GRAM. Second, the applications for which such resources are useful for are well covered by our broker (called MyGrid, which targets Bag-of-Tasks applications).

However, since most designs are trade-offs, our approach has a drawback. It precludes multiple brokers from using the same resources. When this is needed, a broker must delegate work to the broker that is “hiding” the resources of interest. It is not yet clear the practical impact of this restriction, but we believe that our approach is a better trade-off than stretching GRAM as an interface.

Maybe a cleaner solution would be to define a few interfaces, instead of just one, to separate resource managers from brokers. For example, GRAM/LSF and GRAM/Easy are very alike and could be easily put together into a “GRAM-Space-Shared”. Also, one can see intermitted resources as a (weaker) special case of time-shared resources (i.e. GRAM/fork). We suspect that a few interfaces organized in a hierarchical fashion would provide a much cleaner abstraction for resources in a grid than the current one-size-fits-all GRAM.

Acknowledgements

This work was developed in collaboration with HP Brazil R&D. This work has been partially funded by CNPq/Brazil (grant 302317/2003-1). Thanks also to OurGrid team for their great work.

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.

[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] N. Andrade, W. Cirne, F. Brasileiro, P. Roisenberg. OurGrid: An Approach to Easily Assemble Grids with Equitable Resource Sharing. Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP’2003). June 2003

[4] A. Baratloo, M Karaul, Z. Kedem and P. Wyckoff. Charlotte: MetaComputing on the Web. Proceedings of 9th International Conference on Parallel and Distributed Computing Systems, 1996

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

[6] W. Cirne, D. Paranhos, L. Costa, E. Santos-Neto, F. Brasileiro, J. Sauvé, F. Alves Barbosa da Silva, C. Osthoff Barros, C. Silveira. Running Bag-of-Tasks

Applications on Computational Grids: The MyGrid Approach. Proceedings of the ICCP'2003 - International Conference on Parallel Processing - October 2003.

[7] L.Costa, W. Cirne, C. De Rose. Exploring task independence to schedule Bag-of-Tasks jobs in parallel supercomputers. Submitted for publication. http://www.ourgrid.org/papers/exploringtaskindependence.pdf

[8] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling, S. Tuecke. 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

[9] 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, pg. 62-82, 1998.

[10] 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

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

[12] 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.

[13] Globus Web Site. 07/12/2004. http://www.globus.org

[14] M. Neary, P. Cappelo. Advanced eager scheduling for Java-based adaptively parallel computing. Proceedings of the ACM-ISCOPE conference on Java Grande, 2002

[15] M. Netto, C. De Rose. CRONO: A Configurable and Easy to Maintain Resource Manager Optimized for Small and Mid-Size GNU/Linux Cluster. In Proceedings of the 32nd International Conference on Parallel Processing (ICPP'03), pp. 555-562, 2003.

[16] D. Paranhos, W. Cirne, and F. Brasileiro. Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids. Proceedings of the Euro-Par 2003: International Conference on Parallel and Distributed Computing, August 2003.

[17] E. Santos-Neto, W. Cirne, F. Brasileiro, A. Lima. Exploiting Replication and Data Reuse to Efficiently Schedule Data-intensive Applications on Grids. Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing, June 2004.

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

[19] S. Smallen et al. Combining Workstations and Supercomputers to Support Grid Applications: The Parallel Tomography Experience. Proceedings of the HCW'2000 - Heterogeneous Computing Workshop. 2000.

[20] 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.

[21] 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.

[22] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt, D. Snelling. Open Grid Services Infrastructure (OGSI) Version 1.0. Global Grid Forum Draft Recommendation, 6/27/2003. http://www.globus.org/research/papers.html