# CSBase: A Framework for Building Customized Grid Environments

Maria Julia de Lima, Cristina Ururahy, Ana Lúcia de Moura, Taciana Melcop, Carlos Cassino,
Marcelo N. dos Santos, Bruno Silvestre, Valeria Reis, Renato Cerqueira
Tecgraf/PUC-Rio
Rio de Janeiro, Brazil
csbase@tecgraf.puc-rio.br

## Abstract

*This paper describes CSBase, a framework that allows the development of customized grid environments. CSBase provides end-users with a simplified access to grid resources and applications. Although all CSBase instances share some common services, such as user and algorithm management services, each instance has its own web interface and specialized applications and services. CSBase is based on the experience gathered from projects developed by our research group in collaboration with industrial partners. This paper also describes some of these projects.*

## 1. Introduction

The coordinated use of computational resources is an ever-growing need in several companies. In some corporations, the diversity of computer architectures and applications, as well as spread data, hamper resource sharing and cooperation. This kind of scenario is very similar to the one addressed by grid computing infrastructures [7].

A typical grid infrastructure, such as Globus [6], provides software tools to help share computing power, databases, and other resources across different administrative domains. Although such infrastructure is usually applied to integrate resources from different institutions, they can also be used to share resources within the same institution.

Grid middleware provides general mechanisms to integrate resources and to present them to their end-users. This is a convenient feature of the current grid middleware, since it allows its (re)use in different situations. However, this general approach usually involves complex user interfaces.

This paper presents CSBase, a framework that supports the development of customized grid systems. CSBase offers a friendly user interface that can be seamlessly integrated to the users' routine. Originally, CSBase was designed to support the implementation of grid environments within the same institution. As in other grid infrastructures, a CSBase instance provides facilities for resource management and application execution in a distributed and heterogeneous computing environment. However, each instance of CSBase has its own security model and sets of services and applications.

All these facilities are presented to the end-user through a project-centric web interface, which represents a virtual desktop where data files are organized per project. Through this desktop, the user can start and monitor the execution of tasks in the grid. Since an instance of CSBase is customized to some specific use, the web desktop also provides pre-processing and post-processing applications. Preprocessing applications prepare the input data that will be used by a task submitted to the grid, while post-processing applications visualize the task results.

CSBase resulted from two projects developed by Tecgraf[1] in partnership with Petrobras (Brazilian Oil Company). In the first project, we built and deployed an end-user environment that integrates several seismic data-processing algorithms. Although involving a different application domain, the second project required similar grid facilities. This motivated us to build the CSBase framework, so we could easily reuse common facilities in similar projects. To this day, we have already deployed 3 CSBase instances and two other systems are under development.

This paper is organized as follows: Section 2 provides an overview of CSBase, describing its architecture and discussing how to customize it to generate a new instance. Section 3 presents some grid systems implemented with CSBase. Then Section 4 discusses related work, and Section 5 concludes this paper.

## 2. CSBase

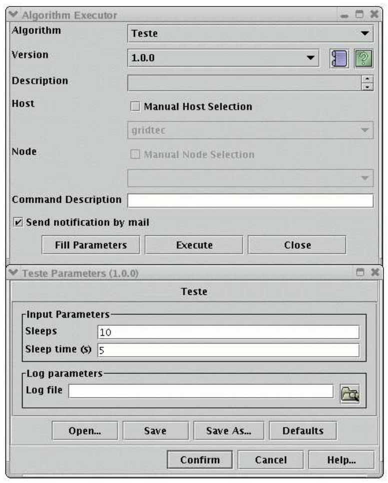
CSBase provides a front-end user interface that abstracts the complexities of a grid computational environment. This

[1]Tecgraf/PUC-Rio – Computer Graphics Technology Group – is one of the laboratories from PUC-Rio's Computer Science Department.

front-end interface allows users not only to access grid resources but also to add new ones. Users can extend the resources of a grid environment by installing *algorithms* — domain-specific non-interactive programs that run on remote *execution hosts*. An execution host is a machine that is connected to the grid environment, providing both execution and monitoring services.

An algorithm execution facility allows users to request the execution of a selected algorithm on a remote host. When the execution of an algorithm ends, CSBase notifies the user who requested it. Algorithm monitoring facilities also allow users to monitor an algorithm execution and, if needed, to interrupt it.

To install a new algorithm, it is only necessary to define the supported platforms and to provide the algorithm binaries and a configuration file that specifies the algorithm's parameters. When a user installs a new algorithm, it is dynamically integrated to the environment. Based on the configuration file, CSBase builds the user interface through which the algorithm's parameters are provided, with no need for recompilation. Figure 1 shows the algorithm execution interface and an example of a dynamically built parameter interface.



Algorithm: Teste
Version: 1.0.0
Description:
Host: ☐ Manual Host Selection
gridtec
Node: ☐ Manual Node Selection
Command Description:
☑ Send notification by mail

Input Parameters
Sleeps: 10
Sleep time (s): 5

Log parameters
Log file:

**Figure 1. Algorithm Execution Interfaces**

Users can monitor CPU and memory usage in the execution hosts. The monitoring information collected from those hosts can be presented both textually and through several types of graphs and charts, as shown in Figure 2. Clusters and host groups can be monitored as a whole. Based on the presented information, the user can select an adequate machine for the execution of an algorithm. This selection can also be delegated to the system, which will base its decision on the monitoring information gathered from the execution hosts.

The user of a CSBase instance organizes her working area by creating *projects*: hierarchical structures that store related user files. By arranging her work in projects, the user is able to define different working spaces. A user project can be either private to its creator or shared by different users. CSBase also offers a publishing facility through which an individual user's file can be made available to other users.

An important feature of the framework is its access control mechanism. Within a CSBase instance, storage areas, algorithms, and execution hosts are protected against unauthorized access. User access rights are defined by specific permissions; the system administrator is responsible for creating those permissions and assigning them to individual users or user groups. Through the system administration interface, the administrator can perform tasks such as adding new users to the system, creating and assigning access permissions, and allocating storage space to individual projects.

A CSBase instance offers most of its facilities through customized *applications*, which are incorporated in the user interface. Unlike algorithms, applications are interactive programs that run within the CSBase instance, and thus can use all its internal functions and resources. This means that an application can access and modify user projects and their contents, start the execution of a specific algorithm, or make use of any other internal service.

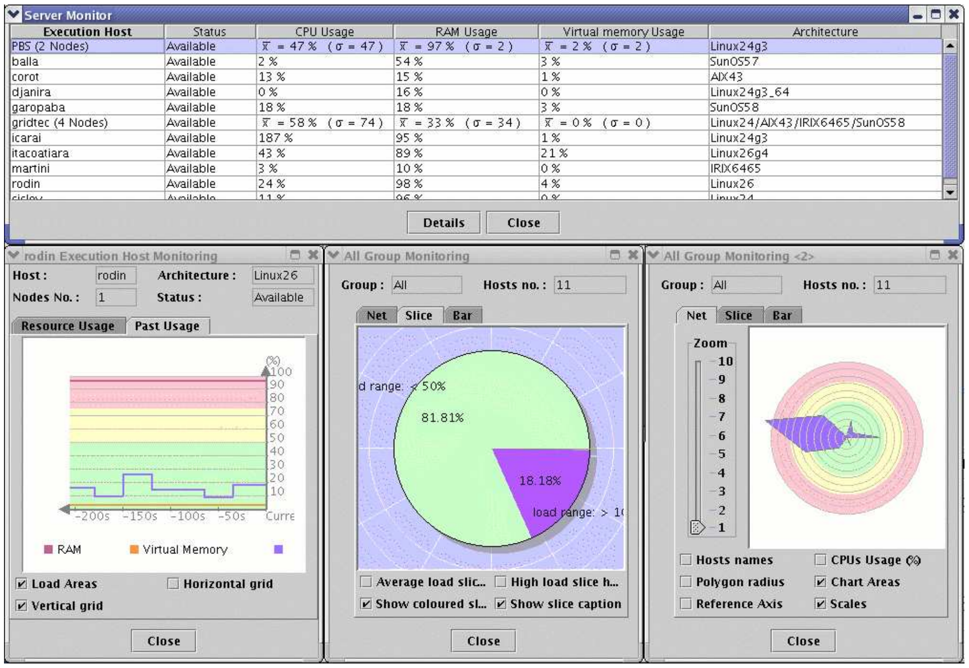
The algorithm execution facility described earlier (Algorithm Executor) is an example of an application the CSBase framework provides to all its instances. The framework also provides other applications, such as a simple notepad, an HTML viewer, and file compression (ZIP) facilities. These applications access files in the user's project area.

The Algorithm Flow Constructor is another useful application provided by the CSBase framework. This application allows users to specify execution flows that combine the sequential and parallel execution of several algorithms. An algorithm flow can be executed on a remote host like a regular algorithm, with the adequate connection of its components' output and input streams.

## 2.1. Architecture

A computational grid can spread over several geographic locations. Each of them maintains a CSBase Server, sharing resources locally. A central CSBase Server is responsible for administrative tasks, maintaining a global administration data repository. All CSBase Servers have an equivalent set of functionalities and each one maintains a user database replica that is synchronized with the central CSBase Server database. The user database replication permits user authentication even when the connection to the central server is unavailable.

Figure 3 shows a general view of CSBase architecture in



Server Monitor

| Execution Host | Status | CPU Usage | RAM Usage | Virtual memory Usage | Architecture |
|---|---|---|---|---|---|
| PBS (2 Nodes) | Available | $\overline{x}$ = 47 % (σ = 47) | $\overline{x}$ = 97 % (σ = 2) | $\overline{x}$ = 2 % (σ = 2) | Linux24g3 |
| balla | Available | 2 % | 54 % | 3 % | SunOS57 |
| corot | Available | 13 % | 15 % | 1 % | AIX43 |
| djanira | Available | 0 % | 16 % | 0 % | Linux24g3_64 |
| garopaba | Available | 18 % | 18 % | 3 % | SunOS58 |
| gridtec (4 Nodes) | Available | $\overline{x}$ = 58 % (σ = 74) | $\overline{x}$ = 33 % (σ = 34) | $\overline{x}$ = 0 % (σ = 0) | Linux24/AIX43/IRIX6465/SunOS58 |
| icarai | Available | 187 % | 95 % | 1 % | Linux24g3 |
| itacoatiara | Available | 43 % | 89 % | 21 % | Linux26g4 |
| martini | Available | 3 % | 10 % | 0 % | IRIX6465 |
| rodin | Available | 24 % | 98 % | 4 % | Linux26 |

rodin Execution Host Monitoring

Host : rodin Architecture : Linux26
Nodes No. : 1 Status : Available

All Group Monitoring

Group : All Hosts no. : 11

All Group Monitoring <2>

Group : All Hosts no. : 11

**Figure 2. Monitoring Execution Hosts**

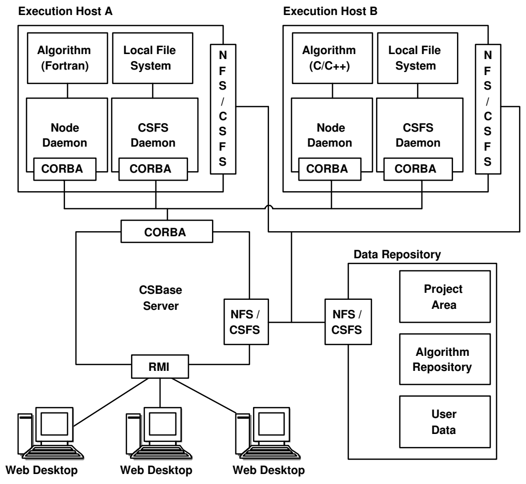


**Figure 3. CSBase Architecture**

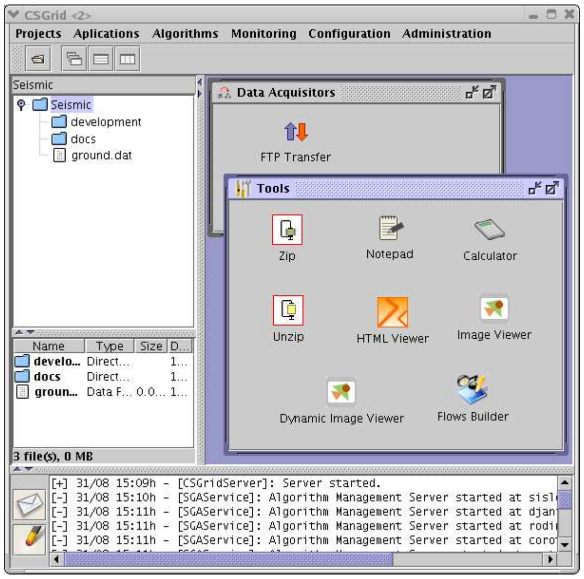
a single geographic location. The CSBase Server is composed by a set of services. The Administration Service, for example, is responsible for maintaining user-related data. The Project Service allows the remote access to the project area. There is also the Algorithm Management Service, E-Mail Service, HTTP Service, and Login Service, among others.

The Data Repository is a disk area where the main server stores the user database, user project data, and the algorithm repository. User database comprises user data and permissions, stored in regular files. The project area stores a directory to each user project, holding regular files and directories, as well as description files and general information. The algorithm repository constitutes a hierarchical structure containing its documentation, a binary code for each execution platform, and a configuration file.

The Node Daemon runs in each algorithm execution host. It has a CORBA interface that allows requesting the execution of an algorithm and gathering monitoring information on hosts and algorithms. Upon activation, the Node Daemon registers itself to the CSBase Server. Regularly, a CSBase Server requests host status information to the Node Daemon. When a user requests the execution of an algorithm, the CSBase Server forwards the request to the selected execution host and starts monitoring the process until completion (success, fail, or interruption). Node Daemons run on Linux, AIX, SunOS, IRIX, Windows and on PBS [10] clusters. On a PBS cluster, a Node Daemon runs only on the server and uses the PBS infrastructure to request and monitor process execution as well as to collect information on the use of host resources.

Considering that traditional distributed file systems, such as NFS, are not always suitable to the grid computing environment [4], CSBase provides a data management infrastructure whose main purpose is to allow binary and data files staging [2]. CSFS offers a remote interface to the execution host's local file system. It is also a daemon process and is collocated to every Node Daemon, allowing binary and data files staging between the hosts and the data repository. A remote client is able to contact the daemon to create new files and directories, as well as to request a copy operation among different hosts. CSFS is an overlaid file system designed for grid systems.

The Web Desktop is the end-user interface of CSBase. It consists of a Java applet that is automatically downloaded and started when the system's URL is accessed. The typical GUI looks like the desktop shown in Figure 4. On the left is a hierarchical tree representing an open project and a table containing file details of the selected directory. Although the directory tree is similar to a regular file system, it consists of remote files instead of local ones. On the right, the installed applications are shown and can be executed, possibly manipulating remote files within the project area. At the bottom, there is a text area used to communicate with other users and to notify system events, such as to indicate the completion of remote processes and the addition of new Node Daemons.



| Name | Type | Size | D... |
|---|---|---|---|
| develo... | Direct... | | 1... |
| docs | Direct... | | 1... |
| groun... | Data F... | 0.0... | 1... |

```
[+] 31/08 15:09h - [CSGridServer]: Server started.
[-] 31/08 15:10h - [SGAService]: Algorithm Management Server started at sisl
[-] 31/08 15:11h - [SGAService]: Algorithm Management Server started at djan
[-] 31/08 15:11h - [SGAService]: Algorithm Management Server started at rodi
[-] 31/08 15:11h - [SGAService]: Algorithm Management Server started at coro
```

**Figure 4. An Example of the Web Desktop**

## 2.2. Applying CSBase

CSBase is an object-oriented framework that developers can customize to build new client/server systems by inheriting and instantiating classes in the framework. It is also a run-time environment that provides the CSBase instances with the basic functionalities of a grid.

One of the most important goals of CSBase is to facilitate the development of specific domain applications and to enable their integration to the grid. The end-users of these applications can thus benefit from all CSBase built-in facilities, customized to their own needs.

The core of the CSBase framework is an infra-structure layer for client/server foundation. At the client side, CSBase provides classes to develop new applications. At the server side, CSBase offers a wide range of services and allows the creation of new ones.

To support the development of an application, the framework comes with a collection of classes that create standard frames, locate available services, handle notifications and events and many other facilities. An abstract class, `Application`, defines hotspots for integrating concrete classes to the Web Desktop. Based on configuration files, the `Application Manager` component instantiates and installs applications in the Web Desktop without the need to modify or recompile CSBase. The configuration file of an application defines the concrete class, icons, language and other resources used by this application. Another configuration file defines the list of applications to be installed in a CSBase desktop.

Domain-specific applications may also require specialized services. At the server side, CSBase can be specialized to provide new services. A CSBase service must implement its specific remote interface and extend the `Service` abstract class, which provides common facilities, such as service initialization and event logging.

Besides implementing the service itself, the developer must also define a permission class that controls the access to that service. As previously mentioned, each user holds a set of permissions that enables him to perform operations within each service.

The CSBase security model already defines permission classes specifying common access policies. One of these classes allows the creation of name/value pair attributes for each permission. This facilitates the dynamic customization of permission classes based on service needs. For instance, when a user selects a node for executing an algorithm, the Node Manager Service checks if this user holds a permission with the name of the machine as attribute.

CSBase services are accessed through a security-aware proxy. When the user logs in, the `ServiceManager` instantiates a proxy to each service and returns their remote references to the client. Further remote calls are handled by the proxy, which forwards them to the actual service implementation along with the identification of the user requesting the operation. The service implementation can than check if the source of a request is a user who has an adequate permission.

Finally, the CSBase instance developer must implement the abstract class `Server` responsible for creating the services available to the system. Currently, this must be done programmatically but the next CSBase version will use a configuration mechanism to create and install the services dynamically.

CSBase's extensibility is crucial to ensure the integration of new applications and services to the core environment. Requirements coming from different systems that already use CSBase demand flexibility as an important design decision. On the other hand, the CSBase development team faces the challenge of adding new functionalities to the CSBase framework without impacting the development of CSBase instances. The next section presents three systems that apply CSBase to build their own grid workbench. It shows different scenarios and requirements of CSBase usage.

## 3. Framework Instances

This section presents three examples of CSBase instances: WebSintesi, InfoPAED, and CSGrid. WebSintesi and InfoPAED are maintained by Tecgraf/PUC-Rio and Petrobras (Brazilian Oil Company). CSGrid is a research project developed at Tecgraf/PUC-Rio and supported by RNP (National Education and Research Network) through the GIGA project [11].

The WebSintesi project integrates applications that support the analysis and processing of seismic data. These applications demand a high processing power, handling seismic data files that comprise hundreds of gigabytes.

One example of these applications is TrackViewer, a graphical application that manipulates seismic and well-drilling data combining them into a single file that will serve as the input for a seismic inversion algorithm [3]. Figure 5 shows a snapshot of the TrackViewer application[2].

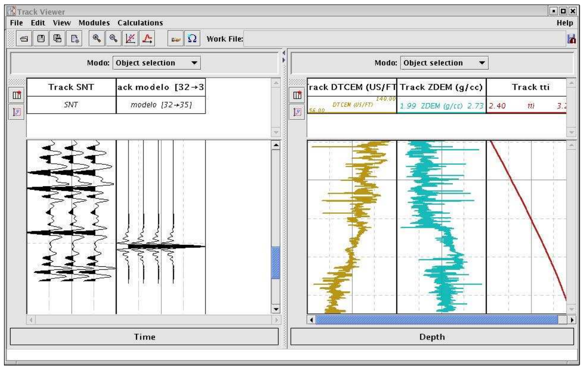


**Figure 5. The TrackViewer Application**

Another example of application integrated in WebSintesi is VGE, an application that retrieves seismic and well data from different storage systems, making the data origin transparent to the end-user. The user can select the data through two distinct interfaces, a hierarchical data tree or a georeferenced interface. The latter consists of a geographical map from where the user can locate which well or seismic data she wants to retrieve. All retrieved data are stored in her project area.

[2]The data presented here are fictitious.

We have developed around 15 services that specialize the CSBase framework to WebSintesi. One of them is the OpenSpirit Service, that uses the OpenSpirit integration framework [9] to access data from any datastore with an OpenSpirit Data Connector and make them available to the user's project area. Through another service, WebSintesi has access to a Tivoli Storage Manager [14], to mount tapes and download files into the project area.

WebSintesi's first version was installed about two years ago. Nowadays it has more than 1,000 users, about 30 algorithms, and several Node Daemons installed in different execution platforms. It has a central server located in Rio and 6 servers located in different Brazilian states. Connected to the central server are 8 Node Daemons, running on 5 multiprocessed IRIX machines, 1 SP2 with 16 nodes, and 2 beowulf Linux clusters running PBS, one with 1,008 processors and the other with 250 processors.

The InfoPAED project groups algorithms for contingency recovery, for example processing information about oil platform balance and stability. Differently from WebSintesi, InfoPAED has very few services. Its first version was installed a few months after WebSintesi. Currently it has one central server, about 15 users, 15 algorithms, and 71 Node Daemons, running on 70 Windows machines and 1 AIX (Regatta).

CSGrid is another instance of CSBase. It is a research project supported by RNP (National Education and Research Network) through the GIGA project [11]. CSGrid is a grid workbench for scientific applications in advanced networks and is about to become the first multi-institutional usage of CSBase.

As part of the CSGrid environment, we will provide facilities to build scientific applications that need support to distributed visualization. These facilities are to be based on different strategies for image rendering and load balancing among CSBase nodes, thus maximizing computational resource usage [1]. CSGrid can also be a guide for people interested in extending CSBase and is available for download at `http://www.tecgraf.puc-rio.br/csbase/csgrid`.

Besides the three examples presented in this section, there are two other CSBase instances under development at Tecgraf.

## 4. Related Work

Globus [8, 6] comprises a set of services that can be independently used to build distributed and heterogeneous infrastructures. Globus, as CSBase, implements the main components to create a computational grid environment, i.e. security, data management, resource management and information service. These components can be joined to create new applications in the grid. A set of development packages called Commodities Grid Kits [15] makes the grid applications development easier through the use of high-level frameworks.

Globus user files are stored using a regular account, which is determined by a mapping procedure from user identities to user accounts. In CSBase, all files are stored using a single account and the permissions scheme allows a better granularity control over data privacy and sharing possibilities between users.

Globus has been developed to address different kinds of applications and services in the grid, and therefore it presents a complex and extensive interface. The design and implementation of CSBase has been driven by the needs of specific applications, providing a lightweight system for grid computing with specialized interfaces. In addition to middleware services, CSBase also offers facilities to build customized grid systems that provide their end-users with a working environment integrated with the grid.

Condor [12, 13] is a resource management system for computing intensive jobs. As well as CSBase's algorithms, Condor's jobs must be able to run in batch mode. It aims at using idle resources available in clusters or desktop workstations. Submissions are distributed so each host maintains a job queue and there is no central server to orchestrate jobs.

Besides the command-line applications, Condor also provides a set of APIs that allows the development of job-management applications [5]. In contrast, CSBase provides high-level services, allowing developers to create applications in a faster and simpler way, for example by customizing provided components.

## 5. Final Remarks

In this paper, we presented CSBase, a framework that helps the development of customized grid environments. Like other grid infrastructures, CSBase provides support for remote task execution, task monitoring and resource management, among other common features of grid middleware. However, CSBase differs from other infrastructures in the facilities it provides to develop customized grid environments, composed by specialized services and end-user applications. In this way, a CSBase instance provides its users with an integrated grid workbench within which they can easily access grid resources.

We have already applied CSBase in some projects and, based on this experience, we can confirm that an end-user working environment integrated with the grid is a very important feature to enable the effective use of grid resources.

Currently, we are investigating the evolution of CSBase in many directions. We plan to expand the capabilities of the Algorithm Flow Constructor with a better support for grid orchestration. We are also working on the improvement of our scheduling mechanism to take into account data transfer overheads between machines.

As part of CSGrid environment, we are investigating mechanisms that contribute to the development of coupled parallel applications using our distributed rendering system [1]. In this regard, our goal is to adapt this system to be used in the CSGrid workbench.

We also plan to provide CSBase with the capability to work with decentralized administration. This feature will allow the use of CSBase in environments with multiple administrative domains.

## References

[1] F. R. Abraham, W. Celes, R. Cerqueira, and J. L. Elias. A Load-balancing Strategy for Sort-First Distributed Rendering. In *Proceedings of SIBGRAPI 2004*. IEEE Computer Society, Oct. 2004.

[2] J. Bester, I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke. Gass: A data movement and access service for wide area computing systems. In *Sixth Workshop on I/O in Parallel and Distributed Systems*, May 1999.

[3] N. Bleistein, J. Cohen, and J. Stockwell. The mathematics of seismic imaging, migration, and inversion. Springer, 2000.

[4] B. Callaghan. *NFS Illustrated*. Addison-Wesley, 1999.

[5] Condor Team. *Condor Version 6.7.10 Manual*. University of Wisconsin-Madison. `http://www.cs.wisc.edu/condor/manual/v6.7/`.

[6] I. Foster. A Globus Toolkit Primer. `http://www.globus.org/primer`, 2005.

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

[8] Globus. `http://www.globus.org`, 2003.

[9] Openspirit. `http://www.openspirit.com`, 1997.

[10] Torque resource manager. `http://www.clusterresources.com/products/torque/`, 2001.

[11] GIGA Project. `http://www.giga.org.br`, 2005.

[12] T. Tannenbaum, D. Wright, K. Miller, and M. Livny. Condor – a distributed job scheduler. In T. Sterling, editor, *Beowulf Cluster Computing with Linux*. MIT Press, October 2001.

[13] D. Thain, T. Tannenbaum, and M. Livny. Condor and the grid. In F. Berman, G. Fox, and T. Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., May 2003.

[14] Tivoli storage manager. `http://www.tivoli.com`, 1999.

[15] G. von Laszewski, I. Foster, J. Gawor, W. Smith, and S. Tuecke. CoG Kits: A Bridge between Commodity Distributed Computing and High-Performance Grids. In *ACM Java Grande 2000 Conference*, pages 97–106, San Francisco, CA, June 2000.