

DEVELOPING A PHP-BASED LEGACY APPLICATION GRID SYSTEM

by

LORENZO CAMPANELLI

Advisor

DR. HALA EL AARAG

A senior research project submitted in partial fulfillment of the requirements
for the degree of Bachelor of Science
in the Department of Mathematics and Computer Science
in the College of Arts and Science
at Stetson University
Deland, Florida

Spring Term
2006

TABLE OF CONTENTS

TABLE OF CONTENTS-----	i
ABSTRACT-----	1
1. INTRODUCTION-----	1
2. BACKGROUND-----	2
2.1 Traditional Distributed Computing v. the Grid System-----	3
2.2 Challenges-----	4
2.3 Advantages of Grid Systems-----	6
3. RELATED WORK-----	7
3.1 Deployment-----	8
4. PHP IMPLEMENTATION-----	10
4.1 PHP Limitations-----	10
5. RESULTS-----	11
6. CONCLUSION-----	12
7. REFERENCES-----	13

Abstract

A big problem with the emerging distributed Grid technologies is that each proposed implementation carries unique design and interface features that do not necessarily conform to an accepted standard. This lack of a presiding standardized model for useful Grid systems has created a level of uncertainty and complexity with respect to practical deployment of systems outside of academia. While implementation of emerging systems within their respective research domains may seem successful, it still leaves questions open about how practical these competing designs are to existing legacy applications. This project proposes a PHP-based Grid system with the goal of providing a practical and rapidly deployable system which maintains compatibility with existing legacy applications.

1. Introduction

The Internet was created from the need for large computing centers to exchange data through a common medium. As this concept evolved, many extensions and layered technologies allowed for smaller autonomous domains to join, which caused the massive amount of intra-domain computer communication that exists today. With these rapid additions of autonomous systems, the Internet now contains a vast amount of computer resources that have largely aggregated into inefficiently used autonomous groups. This is especially the case within any common institutional computer lab, where many workstations remain idle most of the work day. The Grid computing concept exploits this massive infrastructure of the Internet with the intent to provide otherwise unused computing resources to applications that need them. Demands for such massive resources of a Grid System originate from users who typically deal with large data sets, complex equations, and scarce equipment, especially when each respective user's local supply is inadequate. For example, bioinformatics, climate modeling, financial modeling, and Genetics storage are just some of the common *grand challenge* applications that work well with the Grid Computing concept. Because most of these applications involve collaborative research at a global scale, the distributive structure of a

Grid System coincides perfectly with the same cooperative nature of each participating domain. As these participating domains provide computing resources, the resulting Grid System becomes a utility that allows for practical delivery of computing services to areas with high demand. This in essence requires a Grid System to act as the platform from which a large scale virtual computer utility can exist.

2. Background

Delivering such a platform can become difficult when attempting to manage the various commodities of a Grid System. For example, a Grid System design can quickly become too complex for implementation when attempting to cover processor, storage, and instrument management. Processing management focuses on dynamically providing capacity through *CPU scavenging*, which searches for otherwise idle CPUs. Maintaining proper CPU load balancing at a large scale, under a multi-user environment can become one of the most challenging design tasks for a Grid System. Sharing data and providing storage on a vastly distributed data set, and sharing scarce scientific instruments through remote operation can also become management nightmares. Because the rate of design complexity increases quickly, many Grid Systems limit the platform interface to a particular function which can be focused on a specific user group. A perfect example of this is the *SETI@Home* distributive project, one of the most successfully deployed distributed systems to date. *SETI* software is deployed through a simple screen saver which can quickly configure an otherwise idle personal computer into a contributing processing resource that exclusively handles the analysis and exchange of work-units from an extra-terrestrial radio research center. In this case, the research center becomes the focused user group that accesses a large amount of donated cycles for a specific use.

The simplistic approach to this Grid-like system allowed *SETI* to overcome the delays and complications of accompanying broader requirements. Since proving to be a major success, many similar research projects have followed *SETI@Home* with equivalent distributive platforms.

However, such proprietary platforms tend to steer too far away from the generally envisioned Grid Computing concept. Many argue that keeping Grid computing platforms closed from a wider range of applications and user groups hinder the development of a true Grid Computer System. By preventing extension and diverse user group access, a distributive platform can severely restrict and divide capacity, which can add to the already inefficient use of idle computer resources. In effect, such proprietary platforms are more like continuations of traditional distributed systems, such as clusters, which lack the innovation of Grid System ideas.

2.1 Traditional Distributed Computing v.s. The Grid System

Both traditional distributed systems and Grid Systems serve many of the same fundamental purposes, such as concurrent computing, and high-performance computing. Many of these similarities occur because Grid Systems extend the concept of distributed computing with a more global mindset. A traditional distributed system, like a cluster, involves closely coupled and homogeneous components, while Grid Systems attempt to join loosely coupled, heterogeneous components on a much larger scale. Also, a Grid System focuses more on crossing autonomous system boundaries with the intent to export access to foreign domains in need of such resources. This focus is very similar to electric power grid operations, from which the name *Grid* originated, in which autonomous power plants import and export electricity throughout the grid to other markets and user

groups. In this regard, a Grid Computing System introduces the public utility concept to deliver resources for distributive application.

By adopting the utility infrastructure characteristic, a Grid System becomes the superset of traditional distributed implementations. In other words, all traditional distributed systems become subsets of the global Grid System. Now traditionally isolated clusters can be joined together on a collaborative utility with greater efficiency.

2.2 Challenges

Although a Grid System is conceptually more efficient than any subordinate cluster or other traditional distributive system, it faces many challenges during actual implementation. One of the most immediate challenges involves maintaining compatibility with various components from a very diverse computing landscape. If considering the additional global scale factor of an advanced Grid design, the compatibility requirements of all participating resources can become overwhelming to support. This is especially challenging when managing resources in an autonomous manner, so that a user of the Grid will feel as though his or her interface drives a single virtual system. Encapsulating heterogeneity without compromising processing performance is critical to a useful Grid design [1].

An additional challenge of Grids is its operation upon unreliable, delayed, and low bandwidth connections which are typical on an Internet infrastructure. Unlike traditional systems, Grid distribution must have extended error handling and recovery that takes into account a greater component failure rate due to these link characteristics. Even further connection faults can occur from the distributed administration factor, in which firewall and domain policy can unexpectedly block access from other participating domains on the Grid.

2.2 Search for a standard

Unfortunately, many competing designs toward Grid interoperability have prevented a standard from emerging that clearly defines the framework from which to build Grid Systems. As there are large numbers of projects around the world working on developing Grids for different purposes at different scales, several definitions of how to build Grid Systems exist [2]. For example, virtually all the major computer software and services companies have a respective proprietary Grid system. Oracle, Sun, IBM, and HP all have unique *middleware* products that encourage application development to match their individual interface. However, some Grid System development kits attempt to conform to the emerging *web services* standards that help define a more common Grid System interface. Helping the promotion and acceptance of *web service* based Grid Systems is an industry consortium named The Global Grid Forum. The GGF is leading the effort toward generally accepted framework standards. The GGF is composed of various members, both from academia and enterprise, who collaboratively develop accepted designs for Grid Systems. For example, the GGF has successfully defined standards like the *Open Grid Services Architecture* (OGSA) which is heavily based on *web services* protocols. These GGF endorsed standards are then implemented through the *Globus Alliance*. The *Globus Alliance* develops the *Globus Toolkit* which provides a platform from which to build grid-enabled applications. These are just some of the rapid developments being made quite recently within the community, of which the OGSA v1.0 being completed as of January 2006 [3]. These premature developments exist because the idea for a Computational Grid has appeared relatively recently with respect to earlier related ideas in parallel and distributed computing research [4].

Even with the increased efforts required for constructing Grids, one can expect a higher return of performance when designed and managed properly. For example, although the Grid depends on unreliable internet communication to utilize components, it compensates for failures by usually being highly scaleable and recoverable if enough component alternatives are available. This is exactly why the design of a Grid system is critical and highly complex, in which the overall performance, availability, and security of the system to the user depends on its ability to properly manage resources.

2.3 Advantages of a Grid System

The potential of a Grid System is supported by many advantages over previously implemented supercomputing architectures. One advantage is that a Grid system provides very inexpensive access to supercomputer capabilities. This allows for a Grid to have superior power over a standalone supercomputer for a fraction of the cost. For example, the [SETI@home](#) [5] Grid System Project has calculated that one of the most powerful supercomputers, the IBM ASCI White, is rated at about 12 TeraFLOPs for \$110 million, while the [SETI@home](#) Grid is rated at about 15 TeraFLOPs and has so far cost around \$500,000. Further savings come from lower operating costs, such as less direct electricity consumption, environmental conditioning costs, and other centralized costs associated with maintaining a single supercomputer.

An additional advantage that comes with Grid Systems is their greater scalability over local supercomputers. For example, because the Grid System is a grouping of loosely coupled components, supplying resources dynamically to application demands is as simple as adding or removing components from the Grid network. Theoretically, a Grid is as scalable as the Internet, allowing as many computing resources needed to join the Grid as the Internet allows. However, a traditional centralized supercomputer is

either too expensive to scale in response to changing demand, or completely limited to the tolerable expansion that was implemented in its initial design.

Furthermore, a byproduct of the distributed nature of a Grid System is the exploitation of existing system resources. Subsystems of the Grid are typically not dedicated to the Grid at all, and only contribute resources when those resources are not in use by their respective administrative domains. Estimates of computer resource utilization by the average enterprise have indicated that up to 90% of CPU cycles remain idle. In many cases, these resources are not properly managed within a traditional network, so applications that require such resources are unable to access them. Even systems that become obsolete to an institution and would otherwise be disposed of may instead be recycled into beneficial components of a larger Grid System. These reasons are what allows a Grid System to efficiently use otherwise wasted or idle computer resources, and instead contribute them to applications that need them.

3. Related Work

The most promising Grid systems proposal is that of Globus, which has been heavily researched and developed into a highly scaleable toolkit. With this toolkit, Grid systems may be deployed with a layered approach by building required applications above the fundamental middleware interface. As a result, these layered systems become services, which can further be managed by other components from the other toolkits. For example, the toolkit's *Unified Resource Information Service* provides real-time information about the underlying metasytem structure and status. Also, the *Globus Heartbeat Monitor* module provides system fault detection and component recovery. This specific module has two sets of API to provide *registration* and *notification* interfaces with client applications [6]. It is Globus' intention to provide such a

metacomputing abstract machine by providing these many interfaces and layered design for developers to build upon [7].

3.1 Deployment

However, this is an example of how a surplus of interfaces and complex design can prove impractical for existing legacy applications outside the efforts of the Globus research requirements. Many of the proposed Grid standards are attempting to accommodate too many use cases in which inconsistent interfaces within the framework exist [8]. For example, some of the adopted web service protocols within components of *Globus*, such as the *Metacomputing Directory Service*, use an altered version of the otherwise standard *Lightweight Directory Access Protocol* (LDAP) protocol to satisfy unique requirements of certain proposed Grid use [9].

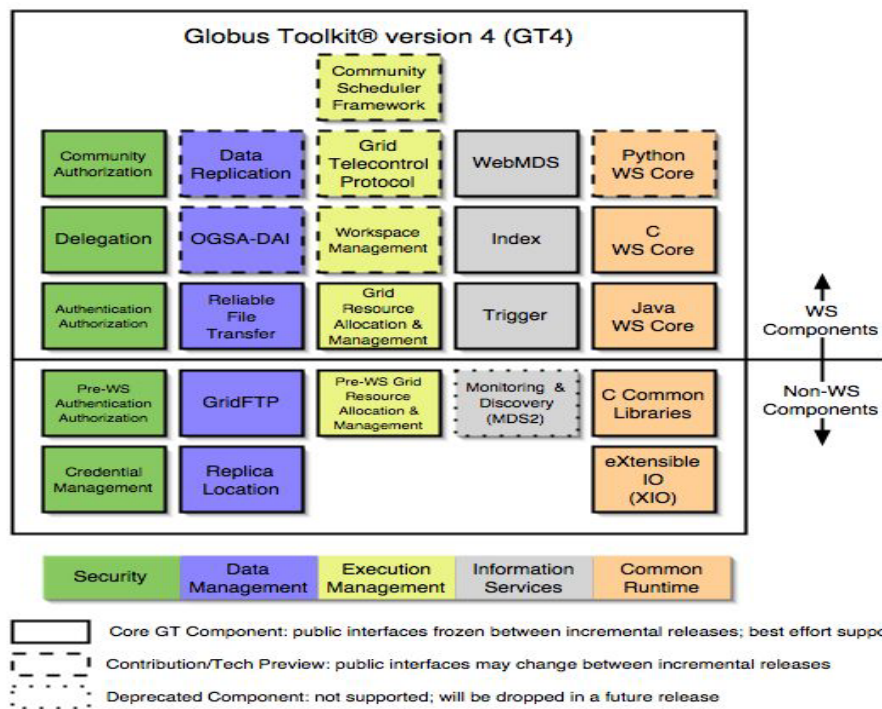


Figure A: The various complex components of the Globus *middleware* toolkit.

Much of the proprietary Grid technology also falls short of being immediately useful “out of the box”. Groups who wish to integrate a Grid System within existing

infrastructure usually need to invest lots of time configuring components and altering vital applications, as well as invest in additional training and consulting. However, those Grid Systems based on existing Web protocols, like HTTP and FTP, are far easier to deploy. This is especially the case with Grid System software that imbeds into a typical web server environment. Since most domains run a dedicated http server, and virtually all workstations carry compatible web browsers, a Grid System can quickly take advantage of the proliferated http protocol and remain compatible with all platforms designed for World Wide Web access through an internet connection.

A web based Grid System is even easier to deploy when composed of popular scripting languages that were designed for web server development. The popular Linux, Apache, MySQL, and PHP/Perl server software combination, also known as a L.A.M.P. platform, perfectly supports potential script-based Grid Systems on top an already proliferated server environment. Not only can these environments exists on Unix-like platforms, but on various other platforms like Windows and Macintosh, through complete *.A.M.P packages. For example, *The Uniform Server* package allows any Windows based OS to quickly become a W.A.M.P. environment with full PHP scripting support. The package is small and portable, which allows even a restricted user to operate the environment within their user space, regardless of typical workstation security policies. In essence, a virtually instant Grid is possible by deploying such packages onto lab workstations which are managed by an institutional web server. Having Grid Systems deployable at such a rapid rate can be beneficial to domain administrators who need to increase the utilization of institutional computing resources, especially by merging idle computer labs into a single accessible grid.

4. PHP Implementation

This project chose to implement a Grid System based on the powerful PHP scripting language for this specific deployment advantage. PHP scripts have a large amount of pre-developed web functions which allow it to quickly provide a reasonable user interface and backend server through any typical web browser. Since there are also shell functions that allow for native execution of applications, PHP scripting also acts as an application interface wrapper that quickly makes existing legacy applications Grid-compatible.

4.1 PHP Limitations

Although using a script based Grid System allows for the quickest deployment, it has several operational limitations. For example, PHP web interfaces are stateless, meaning data cannot easily transfer between each client request, especially when dealing with shell execution. Normally, PHP supports server-stored *session global variables* which allow for variables to transfer between page requests, but these cannot currently store *resource* data types, which are vital to process and file interaction during a shell execution. This directly affects the type of applications PHP is capable of executing. It currently is unable to execute graphical or interactive terminal applications due to this underdeveloped resource support. Therefore, this project was strictly designed for command-line interface applications that were fully supported by the PHP shell commands.

Furthermore, PHP scripts running through the typical Apache module are unable to fork execution requests to allow for a concurrently executed system. However, it was possible to fork execution through the CGI based PHP interpreter. This required having a module script on Apache execute the CGI-PHP interpreter which then loaded the forking

capable script to carry out user shell commands. Application publishing is also limited since HTML forms cannot recursively upload local user folders without the use of a downloaded applet or equivalent client-side software installation.

5. Results

The various PHP limitations introduced an unexpected delay in the development progress of the Grid System. Some limitations caused for the design of the PHP scripts to change rather quickly. For example, the forking limitation required that a separate forking script be designed that returned component output to files. These output files would then be acquired by module scripts. Also, default script execution and page request timeouts caused for several testing defects before such timeouts were disabled. Unfortunately, these hurdles prevented many additional features from being implemented. For example, multi user authentication through a public key method and more verbose status reporting were abandoned towards the end of development. As a result, the PHP grid system as a whole lacks many user interface features expected on a modern *middleware* system.

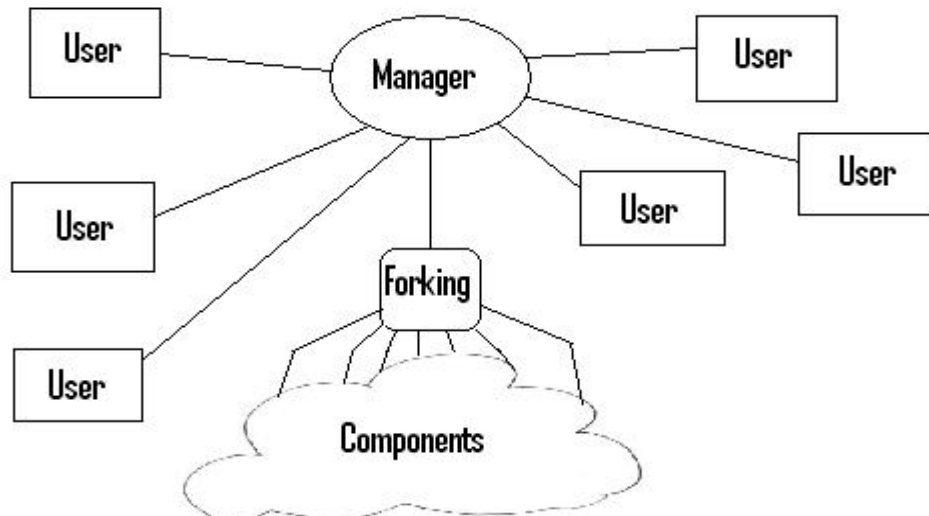


Figure B: Design overview of PHP Grid after forking limitation discovery

However, the popular password hash cracker, *John the Ripper*, was capable of executing through the PHP Grid. The trivial command-line interface of the program, as well as easily configurable settings to allow for specific execution cases allowed for it to distribute and integrate well in the PHP grid.

6. Conclusion

Grid Computing is one of the most actively researched topics within computer science, especially in connection with *web services* and within efforts for modern Internet innovation. Much of the research is done within academia, where the high demand for collective computing resources is most common alongside *grand-challenge* research [10]. Unfortunately, the copious amount of Grid System research has led to many competing definitions and proposed standards. The most promising of Grid System standards is by far those which follow a *web services* based framework. Even though *web services* themselves are still emerging standards, they reflect the most promising future for extending the already proliferated web system into a rapidly deployable *Global Grid*. While extensive Grid System middleware is being standardized, like the Globus toolkit, other temporary solutions can be quickly deployed to exploit the many idle computing resources that exist on the Internet today.

REFERENCES

- [1] Ian Foster. "What Is the Grid? A Three-Point Checklist." *GRIDToday*, 1(6) (22 July 2002). Available: <http://www.gridtoday.com/02/0722/100136.html>.
- [2] R. Buyya, S. Venugopal. A Gentle Introduction to Grid Computing and Technologies. *Computer Society of India*, 19 July 2005.
- [3] H. Kishimoto, J. Treadwell. Defining the Grid: A Roadmap for OGSA Standard, Version 1.0. *Global Grid Forum*, 16 September 2005.
- [4] Ian Foster, Carl Kesselman. "Computational Grids," *The Grid: Blueprint for a New Computing Infrastructure*. San Francisco: Morgan-Kaufman, 1999.
- [5] SETI2Home: Technical News, 8 December 2005. Available: <http://setiathome.ssl.berkeley.edu/>.
- [6] Craig Lee, Paul Stelling. "A Fault Detection Service for Wide Area Distributed Computations." *Proc. 7th IEEE Symposium on High Performance Distributed Computing*, (1998): 268-278.
- [7] Ian Foster, Carl Kesselman. "Globus: A Metacomputing Infrastructure Toolkit." *International J. Supercomputer Applications*, 11(2) (1997): 115-128.
- [8] Leon Erlanger. "Distributed Computing: An Introduction." *PC Magazine* (4 April 2002). Available: <http://www.extremetech.com/article2/0%2C1697%2C11769%2C00.asp>.
- [9] Steven Fitzgerald et al. "A Directory Service for Configuring High-Performance Distributed Computations." *Proc. 6th IEEE Symposium on High-Performance Distributed Computing*, (1997): 365-375.
- [10] Arnold Bragg, Harry Perros. An Architectural Framework and Tool for Modeling Grids, MCNC Research and Development Institute, April, 2005. Available: http://www.cost285.itu.edu.tr/tempodoc/TD_285_05_01_Arnold_Bragg.pdf.