A lightweight interface to local Grid scheduling systems

A dissertation submitted to the
Faculty of Science
at the University of Cape Town
in fulfilment of the requirements
for the degree of
Master of Science

By
Christopher Parker

Supervised by
Dr Hussein Suleman

· 26 May 2009 ·

Contents

List of Figures

List of Tables

Abstract

Many complex research problems require an immense amount of computational power to solve. In order to solve such problems, the concept of the computational Grid was conceived. Although Grid technology is hailed as the next great enabling technology in Computer Science, the last being the inception of the World Wide Web, some concerns have to be addressed if this technology is going to be successful.

The main difference between the Web and the Grid in terms of adoption is usability. The Web was designed with both functionality and end-users in mind, whereas the Grid has been designed solely with functionality in mind. Although large Grid installations are operational around the globe, their use is restricted to those who have an in-depth knowledge of its complex architecture and functionality. Such technology is therefore out of reach for the very scientists who need these resources because of its sheer complexity. The Grid is likely to succeed as a tool for some large-scale problem solving as there is no alternative on a similar scale. However, in order to integrate such systems into our daily lives, just as the Web has been, such systems need to be accessible to "novice" users. Without such accessibility, the use and growth of such systems will remain constrained.

This dissertation details one possible way of making the Grid more accessible, by providing high-level access to the scheduling systems on which Grids rely. Since "the Grid" is a mechanism of transferring control of user submitted jobs to third-party scheduling systems, high-level access to the schedulers themselves was deemed to be a natural place to begin usability enhancing efforts.

In order to design a highly usable and intuitive interface to a Grid scheduling system, a series of interviews with scientists were conducted in order to gain insight into the way in which supercomputing systems are utilised. Once this data was gathered, a paper-based prototype system was developed. This prototype was then evaluated by a group of test subjects who set out to criticise the interface and make suggestions as to where it could be improved. Based on this new data, the final prototype was developed firstly on paper and then implemented in software. The implementation makes use of lightweight Web 2.0 technologies. Designing lightweight software allows one to make use of the dynamic properties of Web technologies and thereby create more usable interfaces that are also visually appealing. Finally, the system was once again evaluated by another group of test subjects. In addition to user evaluations, performance experiments and real-world case studies were carried out on the interface.

This research concluded that a dynamic Web 2.0-inspired interface appeals to a large group of users and allows for greater flexibility in the way in which data, in this case technical data, is presented. In terms of usability— the focal point of this research— it was found that it is possible to build an interface to a Grid scheduling system that can be used by users with no technical Grid knowledge. This is a significant outcome, as users were able to submit jobs to a Grid without fully comprehending the complexities involved with such actions, yet understanding the task they were required to perform. Finally, it was found that the use of a lightweight approach in terms of bandwidth usage and response time is superior to the traditional HTML-only approach. In this particular implementation of the interface, the benefits of using a lightweight approach are realised approximately halfway through a typical Grid job submission cycle.

Acknowledgments

There are many people who affect our daily lives, and they all contribute in different ways to keeping us happy (both emotionally and monetarily) and sane, giving us strength when we have none and ideas when the thought process suffers from short periods of academic drought. It is these people that I would to take the opportunity to thank here...

My family : I would like to thank my parents for putting up with me over the past two years. There have been times where this project has provided much enjoyment, much headache and much stress. They have been a constant source of encouragement and support over this time and I thank them for that.

My friends : For keeping me entertained during this time, making me laugh when servers crash and work is lost and making me forget about the impending repair period when such mishaps occur.

My supervisor : Dr Hussein Suleman for always having so many ideas and opinions along the way, for providing me with the tools I needed to complete this project, for always making time to see me even during the busy periods and finally for always providing sound feedback on the work I had been doing. Thank You!

To Francois Grey and Ben Segal : For providing me with the wonderful opportunity to host the Africa@Home project at UCT and for assisting me during my time spent at CERN in Switzerland.

The National Research Foundation : For providing me with the funding so vital during any degree. I would also like to once again thank Dr. Suleman for providing me with extra income in the form of laboratory duties during this research period.

The experts and test subjects : Finally I would like to thank the scientists and test subjects who graciously offered their time to assist with various facets of this research over the course of the past two years.

List of Acronyms

AJAX
: Asynchronous JavaScript and XML

API
: Application Programmer's Interface

BSD
: Berkeley Software Distribution

CA
: Certificate Authority

COTS
: Commodity Off The Shelf

CPU
: Central Processing Unit

CSS
: Cascading Style Sheet

DHTML
: Dynamic HyperText Markup Language

DOM
: Document Object Model

DRS
: Data Replication Service

FTP
: File Transfer Protocol

GCB
: Generic Connection Broker

GRAM
: Globus Resource Allocation Manager

GridFTP
: Grid File Transfer Protocol

GSI
: Globus Security Infrastructure

GSML
: Grid Service Markup Language

GT4
: Globus Toolkit 4

GUI
: Graphical User Interface

HPC
: High Performance Computing

HTC
: High Throughput Computing

HTML
: HyperText Markup Language

IBM
: International Business Machines

IPC
: Interprocess Communication

JSDL
: Job Submission Description Language

KWIPS
: Kilo-Whetstone Instructions Per Second

LSF
: Load Sharing Facility

MCS
: My-Condor Submit

MDS
: Monitoring and Discovery Service

MIMD
: Multiple Instruction Multiple Data

MPI
: Message Passing Interface

MW
: Master/Worker

MWIPS
: Millions of Whetstone Instructions Per Second

NFS
: Network File System

OSCAR
: Open Source Cluster Application Resources

PBS
: Portable Batch System

PC
: Personal Computer

PSDL
: Parameter Sweep Description Language

PVM
: Parallel Virtual Machine

RFT
: Reliable File Transfer

RLS
: Replica Location Service

SETI
: Search For Extra-Terrestrial Intelligence

SGE
: Sun Grid Engine

SIMD
: Single Instruction Multiple Data

SOA
: Service Oriented Architecture

SOAP
: Simple Object Access Protocol

SRB
: Storage Resource Broker

UNIX
: UNiplexed Information and Computing System

URL
: Uniform Resource Locator

W3C
: The World Wide Web Consortium

WCG
: World Community Grid

WS
: Web Services

WSRF
: Web Services Resource Framework

XHR
: XMLHttpRequest

XHTML
: Extensible HyperText Markup Language

XML
: Extensible Markup Language

Chapter 1

Introduction

"I think there's a world market for about 5 computers."
Thomas J. Watson, Chairman of the Board, IBM, circa 1948

The next great revolution in science is thought by many to be the concept of Grid computing. Just as the Internet has revolutionised the way in which people communicate and exchange information, the Grid promises to solve some of the greatest challenges known to mankind by making available an immense amount of computational power to solve so-called grand challenge problems. Although the initial vision for the Grid has not yet been fully realised, research in the field has led to a number of advances that have changed the way in which scientists approach problem solving. By combining the power of thousands of computing elements in a secure manner, problem solving and data storage at a previously unimaginable and impossible scale has been realised, an outcome far removed from the way in which people were thinking at the dawn of the computing era.

In parallel to research in Grid computing technology, use of the World Wide Web has grown significantly. The move from Web 1.0 to Web 2.0 has seen the advent of many new types of Web applications, as well as an increase in the use of the browser as an engine for running complex applications previously limited to the desktop. Furthermore, core Grid middleware has transitioned from a monolithic architecture to a component-based Web-service model. This approach has made the development of new Grid services simpler and allows for a more flexible architecture capable of growth in a rapidly expanding field. This transition has made it possible for the Grid to take advantage of browser improvements and Web 2.0 technology as a medium for users to interact with the Grid. Unfortunately, the majority of research in the field of Grid computing has focused on the development and improvement of core Grid middleware. Although this research is vital, the Grid is not as accessible as it could be, thereby threatening its future as potential users revert to traditional scientific computing platforms in an effort to avoid the steep learning curve. In order to overcome the usability problem currently plaguing Grid technology, this research has focused on the development of a Web-based, Web 2.0-inspired system capable of abstracting away the complexity of Grid systems.

This chapter will motivate why this research is important as well as outline the objectives of the research project. Furthermore, the scope and limitations to which the project is constrained are also presented. Finally, an outline for the dissertation is provided.

1.1 Motivation

As already mentioned, Grid computing middleware is inherently complex due to the nature of large-scale distributed systems. Furthermore, Grid middleware makes use of low-level command-line utilities in order to create, launch and monitor jobs on the Grid. The use of such utilities, however, requires extensive knowledge of low-level Grid operations in order to know when, and in which context, certain utilities need to be used. An example of a simplified Grid creation, submission and monitoring procedure is outlined below for a typical Grid job, assuming a local scheduling system is used. A more detailed version of this procedure is presented in Chapter 3, Section 3.3.2.

1. Using a shell, obtain information on the status of the Grid.
2. Obtain information on the available resources on the Grid in terms of operating systems and architectures.
3. Write and compile, or use an existing, Grid application taking the above information into account.
4. Use a reference manual to describe the job in terms of Grid resources by creating a submission file.
5. Transfer input files if needed.
6. Submit the job using the Grid job submission utility.
7. Obtain status information on the Grid job just submitted.

In this example, no security mechanisms in terms of authentication and authorization have been taken into account. Typically, users would have to generate and supply their own certificates in a production Grid environment in addition to the procedure outlined above. The generation and use of such certificates is not always fully understood by users, leading to the blind use of such certificates [Foster et al., 2001]. This complexity is hampering the uptake of Grid technology, since many scientists either do not have the necessary Grid-specific knowledge needed to understand and execute all Grid processes or do so at the cost of a steep learning curve.

Throughout the course of this research, two main motivators for the enhancement of Grid usability have been identified. These are listed below:

1. Computational problems are getting larger and are requiring the use of large utilities such as Grids in order to solve them. An expanding need for such systems therefore highlights the importance of usability research in this area.
2. Creation of improved interfaces should lead to a change in the way Grid software is written which will make building future interfaces simpler. This is currently not the case.

The success of Grids relies on people using them to solve problems. If Grid infrastructures are not being used effectively due to a lack of usability, it is possible that the future of Grids will be short lived. It is therefore important that a suitable mechanism is found to make Grids more accessible. This research project therefore looks at one way of solving the Grid usability problem by designing and implementing a Web interface to such a system. Although Web interfaces to Grid computing systems do exist, the approach taken in this research is believed to be novel due to the lightweight, Web 2.0-inspired approach taken, along with its comprehensive multi-faceted evaluation.

1.2 Objectives

This chapter has alluded to some of the ways in which the Grid usability problem has been approached during this research project. However, in order to clearly outline the objectives of this research, each objective will be presented in this section along with an accompanying discussion on the way in which the particular objective has been met. The importance of each objective with respect to the outcome of the research project at large is shown by ordering the discussion of these objectives from the most to least significant.

1.2.1 Determine if a Web interface to a local Grid computing system can be both functional and usable

When designing any piece of software that one expects people with differing skill levels from multiple disciplines to use, usability is a key factor. Since the interface built as part of this research was envisioned for use by scientists without Grid-specific knowledge, the way in which the interface is displayed to users and the way in which tasks are represented was a key design consideration. A variety of techniques were used to achieve this goal, however, one worthy of mentioning at this point is the AJAX-based design approach. Over the past few years a steady increase in the number of sites utilising techniques synonymous with Web 2.0, and in particular AJAX, has been observed [Jazayeri, 2007]. A Web interface designed using this technique, when used correctly, enhances usability by decreasing request turnaround time, creating a dynamic workspace and also providing a structured, flow-oriented and uninterrupted user experience. Another important mechanism to building an interface with a high degree of usability, is to involve the end-user in its design. Therefore, throughout this research, the user forms a key part of the design process.

1.2.2 Determine if computer-literate non-Grid experts are able to make use of the interface

Without adequate evaluation, it is not possible to determine whether the system actually meets the initial objectives. It was therefore decided that three different evaluations would be performed at various stages of the research, namely validation of completeness, user evaluation and finally performance evaluation. The results of each of these evaluations are then used to draw conclusions as to the success of the research project and as to whether the research questions have been answered satisfactorily.

1.2.3 Determine whether lightweight, Web 2.0 techniques live up to claims of increased usability, speed and decreased response time

The term lightweight can have different meanings depending on the context in which it is used. In this context, the term lightweight is used to refer to an interface that, once loaded into a Web browser, responds quickly to user requests, is able to display large datasets in a scalable fashion and is able to render a large amount of data in a short amount of time with little overhead on the user's system. Furthermore, a lightweight interface has economies of scale in terms of bandwidth usage. Since user's request for data result in a transfer of only the specific portion of data requested for the particular operation, overall bandwidth consumption is reduced. The interface built as part of this research was therefore designed to be lightweight.

1.2.4 Ensure that the interface is extensible by allowing for inclusion of different schedulers as "plug-ins" or components

A major downfall to many traditional software systems is caused by their tightly-coupled nature. It is for this reason that a shift has been seen in a move to component-based, modular application

design methods. Such methods increase the maintainability of source code as well as promote reuse of such software components. Another major reason for creating a modular system is to allow for the easy addition of new components as they become available. It is for this reason that the Web interface assumes a modular approach in its inclusion of scheduling systems. A scheduling system is a system that schedules Grid jobs at the local or organisational level. Such systems submit, monitor and control Grid jobs and therefore take the burden of managing large tasks off the user. Such scheduling systems are discussed in more detail in Chapters 2 and 3 respectively; however, for now the details of such systems are not of great importance.

Since most scheduling systems use a similar structure, the components tend to look very similar, thereby making the inclusion of a new scheduler a relatively simple task. Furthermore, a modular approach allows the Web interface to support multiple concurrent scheduling systems, thereby making it particularly suited for use in organisations where different departments make use of different scheduling systems. By providing a standard interface to multiple scheduling systems, that under normal circumstances would need to be individually mastered, users can focus on the task at hand without having to deal with the specifics of the individual systems.

1.3 Scope and Limitations

Since the field of Grid computing is vast, this research has focused on local resource managers (or schedulers) as an abstraction of a large Grid environment. In other words, the core of the Grid or the Grid middleware has been excluded from the Web interface built during this research. In practice, a user would create a Grid job and submit it to a scheduling system via some sort of Grid middleware suite. Although this research has not focused on such Grid middleware exclusively, the approach and results are applicable to large-scale Grids as well. Since Grid middleware acts as a communication medium between virtual organisations, and acts as a global resource monitoring system, one can think of schedulers as performing the same operations as Grid middleware, but at the local level.

1.4 Dissertation Outline

Figure 1.1 provides a graphical representation of some of the chapters that make up this this dissertation. The chapters in this dissertation appear in chronological order and outline the research conducted during each iteration of this research project.

1.4.1 Chapter 2 : Background

This chapter provides an overview of the relevant literature on distributed computing technologies (including Grids), usability of Grids as well as Web technologies and provides a snapshot of the state of the art in Grid computing and related technologies at the time that this dissertation was written. All subsequent chapters draw upon the literature outlined in the background chapter.

1.4.2 Chapter 3 : Condor

The Web interface built during this research makes extensive use of Condor as the primary underlying scheduling system. During the initial stages of this project, the Condor scheduler was used to build a test Grid and formed the basis for many decisions made in terms of the design of the interface. This chapter discusses the fundamental underlying architecture of the Condor system as well as its main features and benefits in the context of Grid computing.

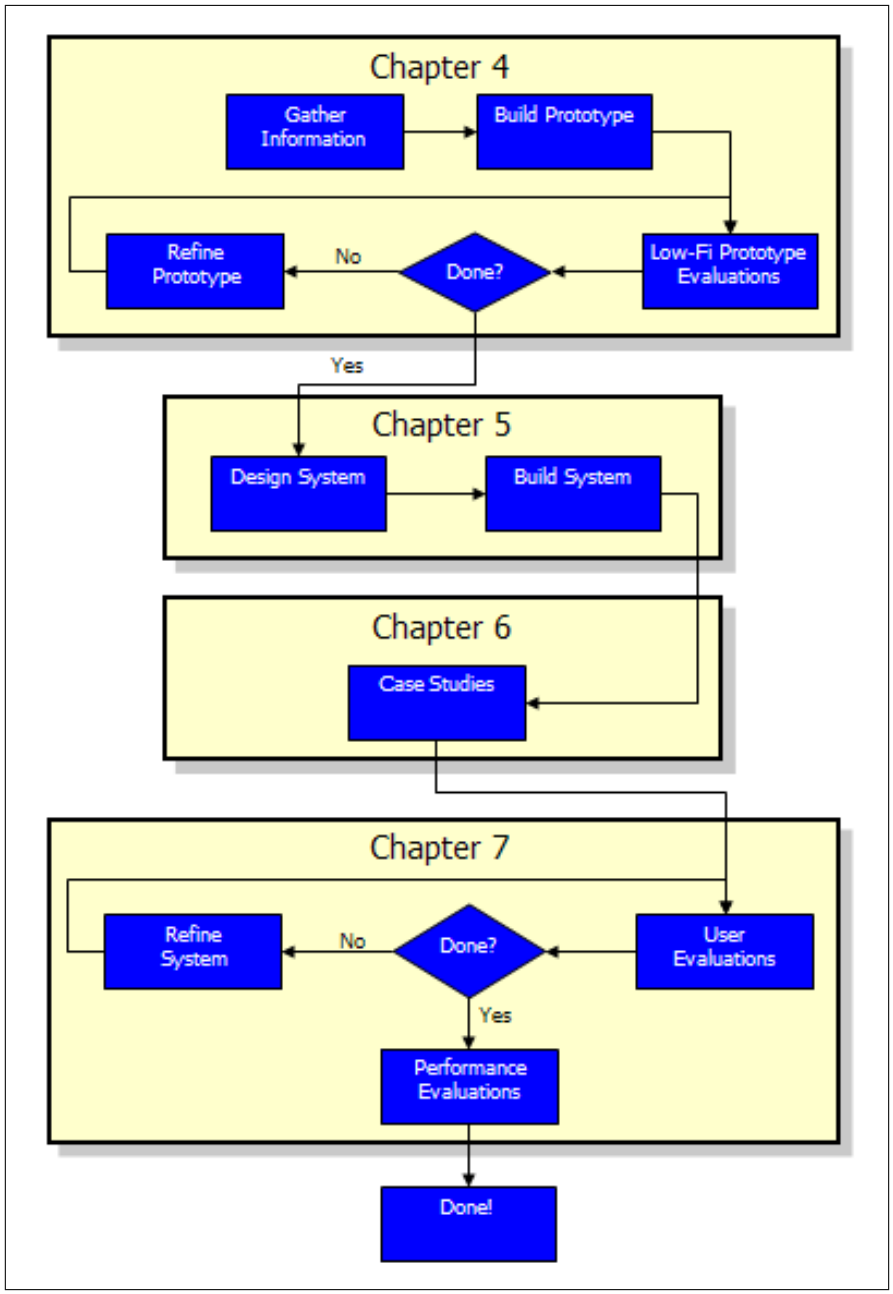

Figure 1.1: Graphical view of chapter content as a flowchart of tasks

1.4.3 Chapter 4 : Prototype Evaluations

This chapter reports on the process followed to develop and to evaluate an initial system design. The chapter discusses interview sessions held with scientists that helped construct a prototype Web interface as well as the first round of user evaluations that were conducted to evaluate this prototype.

1.4.4 Chapter 5 : Infrastructure & Design

The findings and conclusions drawn from the user evaluations discussed in Chapter 4 were used to build the Web interface upon which the rest of this research is based. Chapter 5 discusses the methodologies, tools and design approaches taken during the development of the interface as well as provide an overview of a test Grid constructed with which to test the interface. Furthermore, this chapter provides an in-depth discussion of each interface component as well as the architecture of the underlying system.

1.4.5 Chapter 6 : Case Studies

After the initial design, evaluation and implementation phases of this research, the final system was evaluated in terms of feature completeness. Chapter 6 outlines a set of case studies which were conducted in order to demonstrate that the interface functionality is able to support increasingly complex Grid jobs. Five case studies as well as the results from each of these studies are presented in this chapter.

1.4.6 Chapter 7 : Evaluation

Once the implementation and completeness validation were finalised, the Web interface was re-evaluated. Due to the importance of usability in this research, it was deemed necessary to conduct a second round of user evaluations. The way in which these evaluations was conducted along with a statistical analysis of the results of these evaluations are outlined in the first part of this chapter. The second part outlines a set of performance evaluations specifically focusing on the performance of the AJAX-based design approach as opposed to traditional development techniques.

1.4.7 Chapter 8 : Conclusion

To conclude, a discussion of how each of the objectives set out in this chapter have been achieved, is discussed. Furthermore, a brief reflection on the process of conducting this research is presented, followed by an overview of possible future work.

Chapter 2

Background

2.1 Introduction

Grid Computing is a relatively new area in High Performance Computing (HPC) and has been around since the mid-90s. Although the concept of a Grid has been defined as such for much of this time, only in the last couple of years has Grid computing software been getting some attention as a viable alternative to the traditional cluster computing paradigm of HPC. As the field of Grid computing has matured, more research has been done in this area and has subsequently resulted in a number of tools being made available. Furthermore, with the steady growth of Grid computing infrastructures, many vendors have built front-ends to their software that allow users to make use of Grid resources from a high-level, abstract interface that would make the use of such resources far easier and more convenient.

This chapter begins by giving an overview of current distributed computing technologies, and then discusses the differences between traditional parallel computing and Grid computing paradigms. A discussion of scheduling software that lies at the core of many Grid installations is followed by an overview of Grid job submission mechanisms and standards. This chapter concludes with a short overview of existing Web technologies and how they can be used in order to enhance usability of Grid computing systems.

2.2 Distributed Computing Technologies

The notion of parallelism, in some form or another, has existed since digital computers were first conceived. Advancements in speed and efficiency of computing equipment over time can be partially attributed to the incorporation of parallelism into the core architecture of both hardware and software components. Over the last decade, however, it has become clear that single system parallelism will not and cannot continue to improve at the same rate as has been the case over the past 50 years, mainly due to the laws of physics [Moore, 1965]. Furthermore, scientific problem sizes have expanded rapidly and have resulted in large quantities of data being generated and therefore needing to be processed. Some hardware manufacturers have temporarily overcome these limitations by, for example, simply adding additional cores to processors. However, such approaches will soon suffer from practicalities involving power consumption and heat emission. Agarwal [Agarwal & Levy, 2007] points out that current processors use tens of watts of energy; however, even if this could be reduced to one watt per processor, the power consumption of thousand core processors, which he predicts will be available by the start of the next decade, will be unsustainable. The current trend to add more and more cores to processors therefore does not seem to be scalable in the long-run.

The growing need for more computational power as well as the search for possible alterna-

tives to the physical limitations plaguing non-parallel systems, has led to the design of parallel computing tools that can meet the demands of mostly large scientific computations as well as large amounts of data that need processing. These tools include APIs that allow multiple computational elements to communicate with one another and thereby led to the natural parallelism obtained when harnessing the power of large numbers of such elements. The remainder of this section will look at the difference between the more traditional cluster-based parallel computing paradigm and the "newer" Grid computing paradigm, as well as look at some batch scheduling software that is widely used in both Grid and cluster installations.

2.2.1 Cluster Computing

A cluster computer is a computer that is made up of many computational devices which work together to solve problems in unison [Foster & Kesselman, 2004]. Most scientific problems that require large amounts of processing to be done have relied on such clusters for many years in order to reduce the time taken to obtain results. The computational devices which make up a cluster can range from commodity off the shelf computers (COTS) to high-end server equipment. Cluster nodes, the devices which make up a cluster, are usually located in close proximity to one another as applications that run on clusters are often fine-grained and therefore require a lot of intra-cluster communication. Cluster nodes are usually connected to each other by low-cost interconnects such as Ethernet and are typically made up of between 16 and 64 nodes [Foster & Kesselman, 2004]. Other important characteristics of cluster computers are that they are always dedicated and are therefore not multi-purpose devices and that they are usually, but not always, owned by one person, department or organisation. Single ownership is an important characteristic since security considerations and authorization issues are less complicated and therefore easier to deal with than Grids, for example.

Administration

In order to effectively make use of a cluster, one requires more than just the hardware. Good clustering tools are important in order to efficiently manage the underlying hardware. There are many tools available to manage clusters, two of the most popular being ROCKS [Papadopoulos et al., 2003] and OSCAR [Scott, 2001]. These tools allow administrators of clusters to execute functions across all cluster nodes. For example, these functions can include installation of new software, updates to existing software, batch rebooting or shutting down of cluster nodes as well as a wide variety of other functions. These tools also usually support many different operating systems and are installed as an added layer on top of such operating systems to facilitate administration from a higher level, in many cases making use of lower level operating system commands.

Clusters also can be setup manually, and therefore not rely on high level tools. However, this is non-trivial and typically used for specialised installations. In addition to the cluster management software, many clusters make use of some kind of scheduler. A scheduler is a software system that ensures that jobs are matched to nodes in an efficient manner, and usually serves as a load-balancing tool to ensure that under-utilised nodes are allocated new work and overloaded nodes are not dealt more work [Litzkow et al., 1988]. Some schedulers can migrate work from severely overloaded nodes to underutilised nodes, but this is a non-trivial task and therefore many schedulers do not support such operations. Scheduling of jobs to cluster nodes is discussed in Section 2.2.5.

Job management

Clusters are frequently used to run jobs which have high communication requirements. Such jobs, when split up, require data to be passed from one machine to another. In order to create

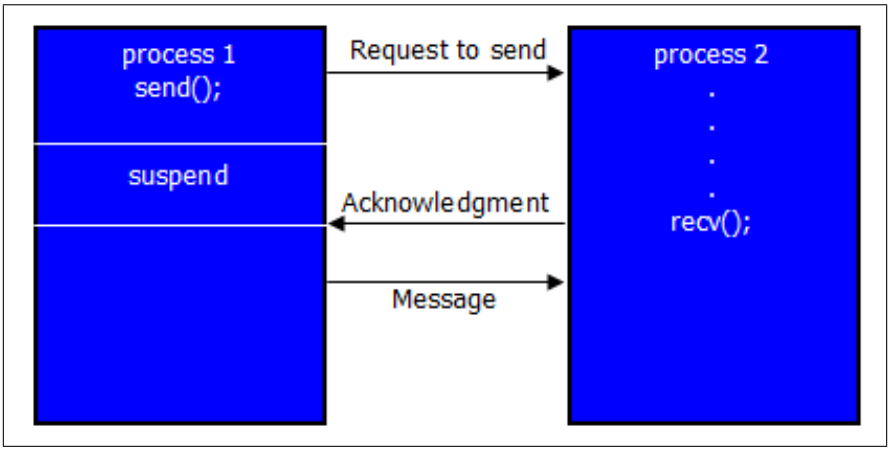

Figure 2.1: Message Passing Example

such jobs, a parallel programming language or toolkit is required. There exist many such systems; two of the more popular are MPI (Message Passing Interface) [Message-Passing Interface Forum, 1997] and PVM [Sunderam, 1990]. MPI consists of a set of routines that programmers can call from within their programs. These calls essentially allow a programmer to break up a program into independent parts or processes and specify which parts are to get run on slave nodes and which part is the master process. Once MPI has distributed these processes to multiple nodes, messages containing data structures are passed between processes using `send();` and `recv();` calls (see Figure 2.1), hence the reference to message passing. The contents of these messages could contain control signals or data needed by other processes and can be sent synchronously or asynchronously. MPI is, unfortunately, notoriously complex to use and therefore used only when much interprocess communication is required. PVM (Parallel Virtual Machine), on the other hand, is different from MPI as it abstracts multiple separate computational devices connected by a network into a single virtual machine. Once PVM is installed on a cluster, the programmer makes use of the PVM library to call routines that facilitate the cooperation between tasks.

Taking the PVM virtual machine approach further, the MOSIX [Bar, 2003] designers took a "fork and forget" approach to cluster computing. Although MOSIX does not support explicit interprocess communication, it does provide an API which can perform transparent load balancing over a set of machines. Communication between processes is made possible by System V IPC as if such processes were running on a single machine.

In the context of cluster computing, a final concept that needs to be discussed is the Master-Worker (MW) model [Linderoth et al., 2000] of parallel programming. This model is an abstract model that can be implemented with many of the technologies mentioned so far. In this model, the Master process is responsible for the algorithm control, whereas Worker processes run computationally intensive jobs on remote machines. The Master process can decide to provide a Worker with more work, or if there no work, generate one result from results each Worker has returned. The MW approach adheres to the four main properties that parallel metacomputing tools should have [Linderoth et al., 2000], namely:

- Programmability : Existing code should be be integrated with the system easily
- Adaptability : The system should adapt dynamically to the heterogeneous execution environment
- Reliability : The system should react correctly to execute elements failing

- Efficiency : The resources should do useful work over long periods of time

2.2.2 Volunteer Computing

Volunteer computing has been given many names over the years including cycle stealing, cycle scavenging, CPU scavenging, Internet computing and now even Grid computing [Anderson & Fedak, 2006]. All but the last, Grid computing, accurately summarise the processes involved in volunteer computing. It is important to note that volunteer computing is not Grid computing, although it is referred to this way by initiatives such as the World Community Grid. Section 2.2.4 gives an overview of Grid computing and, from this, the differences between Grid and volunteer computing can be inferred. Volunteer computing is a way in which ordinary people can help to solve grand challenge problems by enabling their computers to download small datasets and executables from a secure server. The results of these computations are returned to the server for verification and the user then obtains credit for his/her efforts. There are many volunteer computing projects available today and some of these are mentioned in the sections to follow.

BOINC

In order to facilitate the need for volunteer computing over the public Internet, the Berkeley Open Infrastructure for Network Computing [Anderson, 2004] (BOINC) was created. A BOINC core-client makes it possible for an ordinary person at home to connect to a volunteer computing project and then download the executable and input files needed to run the computation. The first project to use BOINC was the SETI@Home project [Anderson et al., 2002]. This project has one of the largest user bases of all volunteer computing projects and is dedicated to finding signs of life in other parts of the universe.

In order to set up a BOINC project, one installs a BOINC-enabled server, ports a project to the BOINC platform so that workunits can be distributed to clients and do some advertising in order to recruit volunteers. Drawing in volunteers is the most challenging part of the process as people need to believe that the computation they will be performing is for a worthy cause.

Another project that makes use of the BOINC platform is the World Community Grid (WCG)[1]. Although the WCG is in fact not a Grid, it also has a large user base since it strives to solve problems that are critical to the survival of the human race. Another strength of this project and why it is so popular among volunteers, is the credit accumulated over time. With many volunteer computing projects, credit stops accumulating when a project shuts down. With the WCG, however, credit continues to rise as new projects are added on a continual basis. This means that volunteers are kept satisfied with a continual supply of credit and the WCG can benefit from an ever expanding user base without having to search for more users with each new project.

Volunteer computing is an enabling technology to assist with solving problems that require immense computational power. However, there are many considerations that must be addressed before one embarks on such a project. Practical aspects such as bandwidth, computational power and storage capacity of the server are important when planning to support a large number of volunteers, as quality of service is important. Furthermore, the type of application that is going to be run needs to carefully be decided upon as volunteers tend to rank projects by significance. In other words, projects dedicated to humanitarian problems are more likely to be adopted by volunteers than problems that benefit commercial or government agencies.

[1] http://www.worldcommunitygrid.org

2.2.3 Cloud Computing

As computational problems become more complex, an increasing need for more processing power has been observed. This chapter has so far discussed two distributed computing techniques that enable such problems to be solved, namely cluster and volunteer computing. However, as problem complexity increases, so too does the need for more computational power, often leading to demand outweighing supply of the computational resources. In order to solve such problems it is either necessary to invest in more cluster equipment— a costly option, or make use of volunteer computing. The latter option is, however, a time consuming process as volunteers need to be enticed to volunteer their CPU time. Furthermore, not all science problems can be tailored to a volunteer computing platform due to problem constraints on IPC, for example. Due to the aforementioned problems, the concept of cloud computing [Delic & Walker, 2008] has been introduced. This "new" distributed computing paradigm allows users with computationally intensive problems to rent CPU hours from a third party. Typically, the third party would have thousands of machines in a datacenter and will assign them to a client based on the clients needs. The client is then billed for the amount of CPU time consumed. Companies such as Amazon[2] have implemented such systems in the form of their Elastic Compute Cloud (EC2)[3] service. This service makes available thousands of machines with different hardware configurations that are linked up to a large storage system, the Amazon Simple Storage Service (S3)[4]. Users contact Amazon, request a certain number of compute elements and are then able to perform processing on what is effectively a huge cluster computer. Users have complete control of the machines they are allocated and build an operating system image for use in the cloud environment. Such third party services have many benefits in terms of investment into computing elements on the part of users or institutions. It is no longer necessary for individuals or institutions to invest in large cluster installations thereby decreasing the monthly expenditure in terms of electricity, administration staff, etc.

2.2.4 Grid Computing

The concept of Grid computing, in theory, is not very different from that of cluster computing. Both Grid and cluster computing paradigms consist of computational devices that collaborate to solve a single problem. The major difference, however, becomes apparent when one compares the characteristics of clusters to those of Grids. It was mentioned in the previous section that some important characteristics of cluster computers are that they are dedicated and under single ownership. With Grids, however, this is usually not the case. Grid resources are usually owned by multiple parties and are therefore on loan for others to use— an important property of a Grid. Since Grid resources are only on loan, a Grid operates differently to a dedicated cluster as resources disappear and reappear frequently. That said, the practical challenges facing Grids are far removed from the theoretical view where Grids merely seem to be another type of cluster architecture. A Grid then can be defined as an interconnected system of heterogeneous computational devices under distributed ownership, usually spread over large geographical areas and connected by public or private communication links [Foster & Kesselman, 2004]. This definition is by no means complete, but attempts to emphasize the major differences between clusters and Grids as they apply in this thesis. Some of the main differences between Grids and clusters are summarised in Table 2.1.

Grid computing has come about out of the necessity to solve larger and larger computational problems that clusters are simply not able to do in any reasonable time period. In order to solve such complex real world problems, Grid computing was envisioned to be a possible solu-

[2] http://www.amazon.com/
[3] http://aws.amazon.com/ec2/
[4] http://aws.amazon.com/s3/

Grid	Cluster
Distributed Ownership	Single ownership
Widely distributed	Close proximity
Low to high speed communication	High speed communication
Non-dedicated	Dedicated
Unreliable	Reliable
Public communication links	Private communication links
Coarse grained applications	Fine grained applications

Table 2.1: Grid vs. Cluster computing

tion. The number of tools available to set up Grid computing environments or even volunteer computing platforms have increased over the years— however, some are more prevalent than others. The Globus Toolkit [Foster, 2006], discussed in the following subsection, which has been under development for more than ten years, is considered to be the de facto standard for Grid computing. Tools such as Globus allow local schedulers located at distributed locations to be "connected" to one another and thereby allow jobs to be submitted to the clusters they control from remote locations. These schedulers are an important component in Grid environments and will be discussed further in Section 2.2.5.

Types of Grids

Grid computing has been incorporated into many different computational environments. This section gives an overview of the various ways in which Grid computing is used today and looks at some of the benefits of utilising Grids in these diverse environments.

The first type of Grid, collectively called an organisational Grid [Abbas, 2004], can be split up into three sub-types, namely, departmental Grids, enterprise Grids and extraprise Grids. The differences among these Grid sub-types, although they belong to one organisation, has to do with the level at which they are used. Departmental Grids are used by a small group of individuals usually working on a single project, whereas an enterprise Grid is used by all departments within an organisation. Extraprise Grids, on the other hand, are utilised by a single organisation or its partners. Organisational Grids are one of the most common types of Grids present in large companies, as all computational resources belonging to such a Grid are deemed to be secure as they are under control of the company itself.

The next type of Grid is known as a global Grid [Abbas, 2004]. The main distinction between a global Grid and an organisational Grid is that the resources present in a global Grid are under distributed ownership. Organisations that make use of such Grids do so with the knowledge that their information is being sent over the public Internet.

A desktop Grid [Abbas, 2004], the next type of Grid to be considered, is a computing infrastructure where a number of organisational computers are utilised in a volunteer computing-like fashion in order to perform part of a larger calculation. The debate rages as to whether a desktop Grid is in fact a true Grid or simply a privately owned volunteer computing infrastructure. Since any Grid can be deemed to be a volunteer computing infrastructure due to its architecture, the converse not necessarily being true, the distinctions between a public Grid and private Grid must be made clear. A production Grid connects resources securely and with appropriate credentials, whereas a desktop Grid can be considered to merely be a subset of a production Grid. The reason for this distinction is due to the fact that privately owned "Grids" do generally not need this secure communication and do not make use of production Grid middleware such as

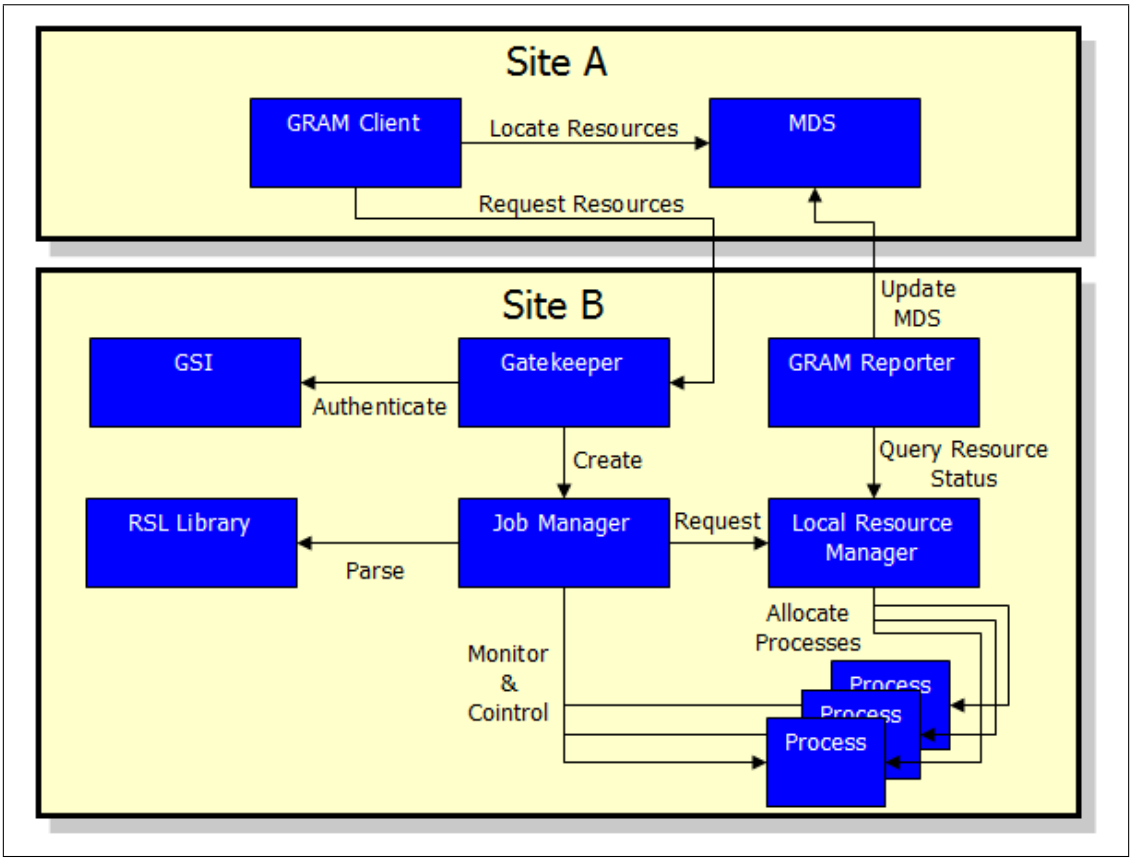

Figure 2.2: Globus GRAM workflow

the Globus Toolkit [Foster, 2006]. For the sake of clarity, however, desktop and public volunteer computing projects will be referred to as Grids.

The last two types of Grids are data and utility Grids [Abbas, 2004]. Data Grids are ordinary Grids that need access to and process large quantities of data. Utility Grids are simply compute resources that get "sold" to companies or individuals for a period of time. This type of Grid is common for companies or individuals that do not have the resources to create their own Grids or do not have the funding to purchase large clusters. The term "Utility Grid" has, however, been overtaken by the term "Cloud Computing", although both are valid.

The Globus Toolkit - theory and practice

The Globus Toolkit[5] is a set of software components that have the ability to join multiple remote computing facilities together across public as well as organisational networks and over large distances, securely [Foster, 2006]. This software allows people around the world to share both data and computing power thereby benefitting all parties involved. Although the Globus Toolkit is feature rich, the toolkit by itself is not a complete Grid solution. As its name implies, the Globus Toolkit is merely a set of tools that allows a Grid to be constructed from multiple independent clusters or existing Grids by making use of Globus services. The toolkit is therefore a glue technology that allows distributed resources to communicate with one another in a secure way. The Globus Toolkit is geared towards security and therefore has functionality built in to ensure that communications between entities connected by Globus is secure. In addition to security, Globus consists of many other services that are vital to Grid functionality. These services include: a Monitoring and Discovery Service (MDS) that keeps a record of the resources available to the

[5]http://www.globus.org/toolkit/

Grid; Grid Resource Allocation and Management (GRAM), which allocates resources to jobs submitted; and Reliable File Transfer (RFT) and GridFTP, which transfer files from one system to another quickly, efficiently and reliably [Foster, 2006].

Grid middleware, particularly in the case of the Globus Toolkit, is predominantly composed of a large set of services. Foster [Foster, 2005] argues that since Grids are inherently distributed systems that communicate by exchanging messages from heterogeneous hardware and software platforms, a service oriented architecture (SOA) makes the most sense since this technology has been designed to facilitate interoperability among such systems. Therefore, with the launch of the Globus Toolkit Verson 4.0 (GT4), Globus incorporates the Web Services Resource Framework (WSRF) [Sebu & Ciocarlie, 2006]. This framework describes a model by which so-called next generation Grid services can be built. Traditional Web Services have been designed in such a way as to be stateless which means that the Web server does not keep state related to any service invocation. The Globus designers realised that this was not feasible for use in a Grid environment, and hence WSRF, or stateful Web Services, was born. This model separates the state of the service request into what is known as a WS-Resource. This resource can then be accessed by making use of a WS-Resource-qualified endpoint reference. The move to incorporate state into Web services has therefore resulted in a more powerful inter-component communication framework and has changed the level at which Grid components can interoperate. Since the Globus toolkit is considered to be somewhat of the standard in Grid computing, the services of which it is made up will now be presented in more detail.

Since the Globus Toolkit has been designed primarily to connect geographically separate Grids together, it has no way of scheduling jobs. This task is left up to third-party tools such as those that will be discussed in Section 2.2.5. Although Globus does not support job scheduling itself, it does provide a way in which to submit jobs to a Grid by making use of the Globus GRAM service— see Figure 2.2 [Czajkowski et al., 1998]. In order to submit a job to a Grid, a GRAM client calls upon the MDS to find resources that are available to run the job and then calls the gatekeeper in order to perform user authentication, determine a local user name for the remote user and start a new job manager. If authentication is successful, the job manager will try to allocate resources by negotiating with the underlying resource manager. The job manager is also responsible for monitoring the state of the new process. Once a new job has been created, the local resource manager takes over and completes its execution. By performing the above operations, the Globus toolkit can be used to connect to local resource managers that are distributed throughout the world.

Since the submission of jobs to Grids usually requires input and output files as well as executables to be transferred from the submission host to a remote machine, Globus contains a number of data-centric services that move and monitor data. In order to copy or move data from one location to another using the Globus Toolkit, tools such as GridFTP or the Globus Reliable File Transfer (RFT) service can be used [Foster, 2006]. Since GridFTP is simply a software tool and not a service, it is not recommended for use with large file transfers as the client must have an open connection to the server at all times[6]. RFT on the other hand, being a service, has the ability to be called from within custom applications as it has an API which exposes its functionality. So far, the ability to transfer data has been discussed. However, one of the main problems faced by any distributed computing system is the availability of such data. Since a number of users might need access to large datasets, it is in their best interest to make copies of the data on their local systems in order to speed up computation. In order to facilitate the data replication process, Globus incorporates a service called the Replica Location Service (RLS). This service keeps record of where replicas are located and can, in the event of one data

[6]http://www.globus.org/toolkit/docs/4.0/data/key/

source becoming unavailable, point a user to an operational data provider. One can, however, argue that file transfer and replication are similar operations, therefore Globus has in place a service known as the Data Replication Service (DRS). This service combines the RFT service with the RLS to provide a service that enables users to move data and add an entry to the replication service in one operation.

So far, the concepts of job scheduling and data management have been discussed. However, in order for any of these operations to take place, a mechanism is needed to determine where the resources where such jobs are to be run are located, if they are operational and if they are willing to receive jobs at the present time. To this end, the Globus Toolkit includes the Monitoring and Discovery Service (MDS), already mentioned above [Foster & Kesselman, 2004]. This service maintains a list of Grid resources, resource status and also monitors such resources. The MDS is comprised of two sub-services which provide differing levels of functionality. The Index service aggregates all resource information into a central repository and the Trigger service executes actions if certain data harvested from Grid resources return true for a particular predefined rule. Other Globus services make use of the MDS on a regular basis in order to obtain up to date information on the status of the Grid.

Other than the services mentioned above, Globus has features built-in to ensure that communication between Grid entities is secure. Since Grid resources are owned by multiple parties as well as the fact that numerous people make use of these resources, the integrity of these systems must be ensured. Since these security mechanisms are out of scope for this thesis, only a broad overview will be presented. In order for secure communication to take place between Grid entities, Globus has a built-in security mechanism known as the GSI (Grid Security Infrastructure)[7]. The GSI makes use of public key cryptography for secure communication between Grid entities. The use of a certificate authority (CA), that can be installed along with Globus, ensures that certificates are authentic. An external CA may also be used if an organisation already has one in place, but this is optional. Once a connection is established between two parties, GSI falls away and unencrypted communication takes place. The reason for this is that the overhead of encryption is too large for frequent communications— however, GSI does incorporate a mechanism whereby integrity can be ensured. This system, referred to simply as communication integrity, does lead to some extra overhead, but not as much as is the case with encryption.

Grid middleware, such as Globus, is composed of a multitude of different services and tools that facilitate remote execution, data management and security. Since these tools are heavily interconnected, the complexity of the entire system is increased. A discussion as to why this is the case and the implications thereof is presented in Section 2.4.

In order to complete the discussion on Grids, it is necessary to discuss the role that job scheduling and schedulers play in Grid computing environments. These schedulers will be discussed in the next section.

2.2.5 Scheduling

Scheduling ensures efficient, scalable and prioritised scheduling of processes to computational resources in a way that is fair to all entities wanting to execute some process (human or otherwise).

This section will give an overview of some HPC-specific schedulers that have become popular in both cluster and Grid computing environments and that have been used or considered during this research.

[7] http://www.globus.org/toolkit/docs/4.0/security/

Condor

The Condor Project[8] [Butt et al., 2003] is a research project run by the University of Wisconsin at Madison and aims to design a system that pools resources from a wide range of heterogeneous distributed computing resources under distributed ownership. The Condor system is a job scheduler that can distribute submitted jobs to any available machine that meets a set of job-specific requirements on the Condor network. The fact that Condor can utilise resources that are under distributed ownership means that, unlike other parallel schedulers, it can operate in a Grid environment. It does this by means of its flocking mechanism (see Chapter 3). If Condor had only supported the scheduling of processes to dedicated resources or clusters, for example, it would simply be another clustering tool.

The Condor scheduling system is discussed at length in Chapter 3.

IBM LoadLeveler

The IBM Tivoli Workload Scheduler LoadLeveler[9] is a batch scheduling system that is similar in functionality to Condor. This system, like Condor, is installed on multiple workstations or cluster nodes in order to build a pool to which to submit jobs. This system, however, does not have any notion of flocking like Condor, but is interoperable with the Globus Toolkit, thus allowing one to build a Grid infrastructure by using these two tools together. Many of the features of LoadLeveler are similar to those of Condor and will therefore not be elaborated upon. However, a feature that Condor lacks, namely accounting, is built into LoadLeveler. Although LoadLeveler is commercial software, an academic licence can be granted for research purposes, as was the case for this research.

Other workload managers

Besides Condor and LoadLeveler, there are a number of other batch schedulers available. A more well known one is Grid Engine from Sun Microsystems, which is freely available from their website[10]. The Portable Batch System (PBS)[11], is also available as a free download, although this version is no longer supported— alternatively, a fee can be paid for the commercial version. Another well known but non-free scheduler is the Platform Computing LSF (Load Sharing Facility) provided by Platform Computing[12]. Again, this system is very similar in functionality to systems such as Condor, but provides a level of technical support in the form of a licence agreement.

There are many batch schedulers available. However, many are commercial software that require licences for the machines they will be installed on. For the purposes of this thesis, only Condor and LoadLeveler will be investigated, although in theory any scheduler could be utilised.

Multi-Scheduler Environments

Many institutions are split up into departments where such departments are usually, but not always, responsible for their own IT infrastructures and, in particular, cluster setup. This usually means that within an organisation, many different schedulers might be present depending on the needs of the particular department. Apart from heterogeneous hardware, an organisation may also now be faced with a heterogeneous mix of schedulers as well, which makes the construction of an organisational Grid complicated. The main reason for these complications lie with the

[8] http://www.cs.wisc.edu/condor/
[9] http://www.redbooks.ibm.com/abstracts/sg246038.html
[10] http://www.sun.com/software/Gridware/
[11] http://www.openpbs.org/
[12] http://www.platform.com/Products/Platform.LSF.Family/

schedulers themselves. Although many batch schedulers have similar functionality, the way in which they are to be used can differ greatly. Most schedulers, for example, require a submission script to be written detailing the way in which a job is to be submitted. The format of these scripts differs radically from scheduler to scheduler and therefore complicates matters when dealing with multiple schedulers all expecting different syntax. In order to circumvent these issues, a standardised format for job submission is required in order to build tools that allow for a single interface to many schedulers. In Section 2.4.2, an overview of the Job Submission Description Language (JSDL) [Global Grid Forum, 2007] will be presented. This language attempts to provide a standardised way in which to represent jobs on a Grid.

2.3 Distributed Computing Paradigm Comparison

Although Grid and cluster computing are similar in many ways, the problems that they are geared to solve differ greatly. Clusters are good at solving many types of problems ranging from very fine-grained, high intra-cluster communication problems to coarse-grained embarrassingly parallel problems. However, since clusters are expensive to buy as well as expensive to run and maintain in addition to other computational equipment that an organisation might have, cluster sizes are generally small [Underwood et al., 2004]. Grids, on the other hand, are more suited towards coarse-grained problems as Grid nodes can be located far apart and can be connected by slow communication links. As mentioned, a reason that fine-gained applications tend not to scale well on Grid infrastructures is that Grid nodes are not guaranteed to be operational for the entire duration of the application execution. If such a fine-grained application were to lose a critical process to a failed Grid node, the application could end up in an error state and therefore have to be restarted. The success of coarse-grained applications by projects such as SETI@Home and the World Community Grid provide compelling results for the potential use of such application types. These projects have in recent years managed to yield hundreds of thousands of hours of CPU time from volunteers, that would otherwise not have been possible, by making use of a coarse-grained approach to problem dissection.

Another major difference between clusters and Grids is the way in which their performance can be measured. Traditionally, clusters have been evaluated for their peak performance in units such as gigaFlops or teraFlops. Clusters therefore are part of a High Performance Computing (HPC) paradigm, since peak performance is the most important factor. Since the advent of Grid computing, High Throughput Computing (HTC) has been considered as an alternative to HPC [Condor, 2008]. Whereas the HPC paradigm looks at the performance of a cluster at any point in time, the HTC paradigm looks at the performance over a longer period of time. Phrased differently, HPC is geared towards solving fine grained problems where rapid synchronisation between processes is necessary. High throughput applications on the other hand are geared towards solving coarse grained problems where the focus is on maximising the output per minute, hence the volume per unit of time. These two performance measurement metrics are fundamentally different and serve to illustrate a difference between the Grid and cluster computing paradigms.

2.4 Grid Usability and High Level Tools

2.4.1 Introduction

Grid computing middleware is inherently complicated to use, and there are many reasons for this. One of the main reasons is the fact that most Grid middleware has been written over a long period of time by many Grid specialists and therefore little attention has been given to designing software that is usable by non-specialists. Since Grid computing software projects have in the

past been mainly research projects, most Grid tools are low-level command-line based systems with no user interfaces or, at best, poor ones, where users are required to go to a lot of effort to submit a job to the Grid. Even with current scheduling tools, discussed in Section 2.2.5, some effort is required in order to prepare a job for the scheduling process. This section looks at some attempts to make the use of a Grid easier for the typical user and how others have proposed to go about such enhancements.

Bruce Beckles [Beckles, 2005] makes the statement that if a system is not usable by non-specialists then its functionality is irrelevant. This is particularly true in the case of Grid computing and the way in which non-specialists are expected to make use of such software. Since existing Grid middleware has usability issues, Beckles provides two alternatives which would make existing middleware more usable. The first of these options is to develop new software that is not only more usable, but interacts with existing middleware. This approach, although a viable option, is wasteful of resources and one would imagine rarely considered. The second option that Beckles provides is to attempt to refactor existing middleware for usability. Again, for large systems, this approach is not feasible and would be considered by some to be somewhat of a "hack". Much time and effort would be required to code usability into an existing system and could lead to catastrophic bugs. Bruin et al [Bruin et al., 2006] came to the realisationf that the use of a Web portal to a Grid environment is instrumental in enhancing usability and also note that tools built for use with such Grid environments should have usability built in from the outset. Their tool, *my_condor_submit* (MCS) is a command-line based tool that allows a user to submit a job to a Grid and provides a simple mechanism for dealing with Globus, SRB [Baru et al., 1998] and a range of schedulers. From their research, they have noticed that it is unrealistic to work with the raw Globus-like commands and that users prefer a higher level tool.

Distributed computing has always posed much more of a challenge than conventional computing in terms of programmability. With the advent of Grid computing, programming has become even more difficult as Grid resources are generally not homogeneous and are also not necessarily always available on the Grid. Shu et al. [Shu et al., 2005] have also found this to be true and in their paper give an overview of why dealing with excessively low-level APIs make Grid programming difficult. In order to provide an easier approach to Grid programming, they make use of a higher level application description language called Grid Service Markup Language (GSML) where Grid application development is aided by visual tools in an event driven fashion.

Usability of Grid computing infrastructures is becoming more important as many people begin to see the benefits that Grid computing has to offer. In order for people to adopt Grid computing more easily, usability issues need to be considered in order to prevent potential users from being frightened away by these complex systems.

The use of high-level tools for abstracting low-level commands is a common way in which usability can be enhanced. The next section gives an overview of such high-level tools and the way in which they manage to abstract the complexities inherent in Grid computing environments.

2.4.2 Grid Job Submission

In order to deploy a computational job to a Grid, a mechanism is needed to help end-users complete this task in an easy way. Traditionally, desktop software has made use of graphical user interfaces (GUIs) for this purpose, and more recently, Web interfaces have become popular. Grid technology, however, has long suffered from the fact that there have only been low-level command-line based tools for this purpose. Even today, many of the best Grid technologies have very limited or no user interfaces to make the use of such systems easier. This has driven some developers to produce Grid portal tools. This section gives an overview of some Grid frontends

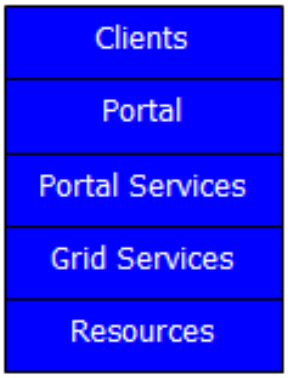

Figure 2.3: Gridport layered architecture

as well as an associated specification that aids in the interoperability between Grid front-ends and Grid middleware.

Grid and Scheduler Front-end's

There are currently numerous Grid portals available that provide high-level access to Grid computing environments. The architecture of such portals typically takes on a layered approach. The Gridport toolkit [Dahan et al., 2004] (see Figure 2.3) the only such toolkit that will be discussed in this thesis, consists of five such layers. The first layer represents the clients or Web browsers. The second layer represents the portal front-end which displays information from lower levels in an appropriate fashion. The third layer, portal services, handles the storage requirements of each user and contains a file management system. This layer is directly responsible for converting the user's request into a Grid specific function call, thereby abstracting the complexities of dealing with low-level Grid commands. The fourth layer contains the actual Grid services to which a user's requests can be mapped by the portal service layer and the final layer represents the resources on which the job will eventually be run. For most modern portals, the reliance on Web services is large and used to provide interoperability between different systems. Most Grid middleware and scheduling software provide SOAP APIs in order to make the development of both low- and high-level tools easier.

In terms of functionality, Grid portals, in general, only offer a basic set of services to end-users. The common features provided include job submission, monitoring and cancellation, as well as file transfer and system status information. In the case of many Grid portals, only certain schedulers are catered for, thus restricting the user-base to whom the portal will be useful. In order to cater for all schedulers, specific modules need to be written for these toolkits, which can be costly.

One final type of interface that deserves some mention is the Grid API. Many such API's exist in order to facilitate submission and monitoring of Grid jobs from within an application itself by way of the API. One such example is the GridSAM API [OMII UK,]. The API allows jobs to be submitted to a wide range of scheduling systems such as Condor, PBS, SGE and many others. This is especially useful for developers not wanting to code their own API's for integration with such schedulers. The GridSAM API was not used in this research, which instead relies on the API's exposed by the various schedulers themselves.

Although the Web interface discussed throughout this dissertation provides many of the same services as provided by existing portals, a number of fundamental differences should be pointed out. One of the most important differences between this interface and existing ones is the type of

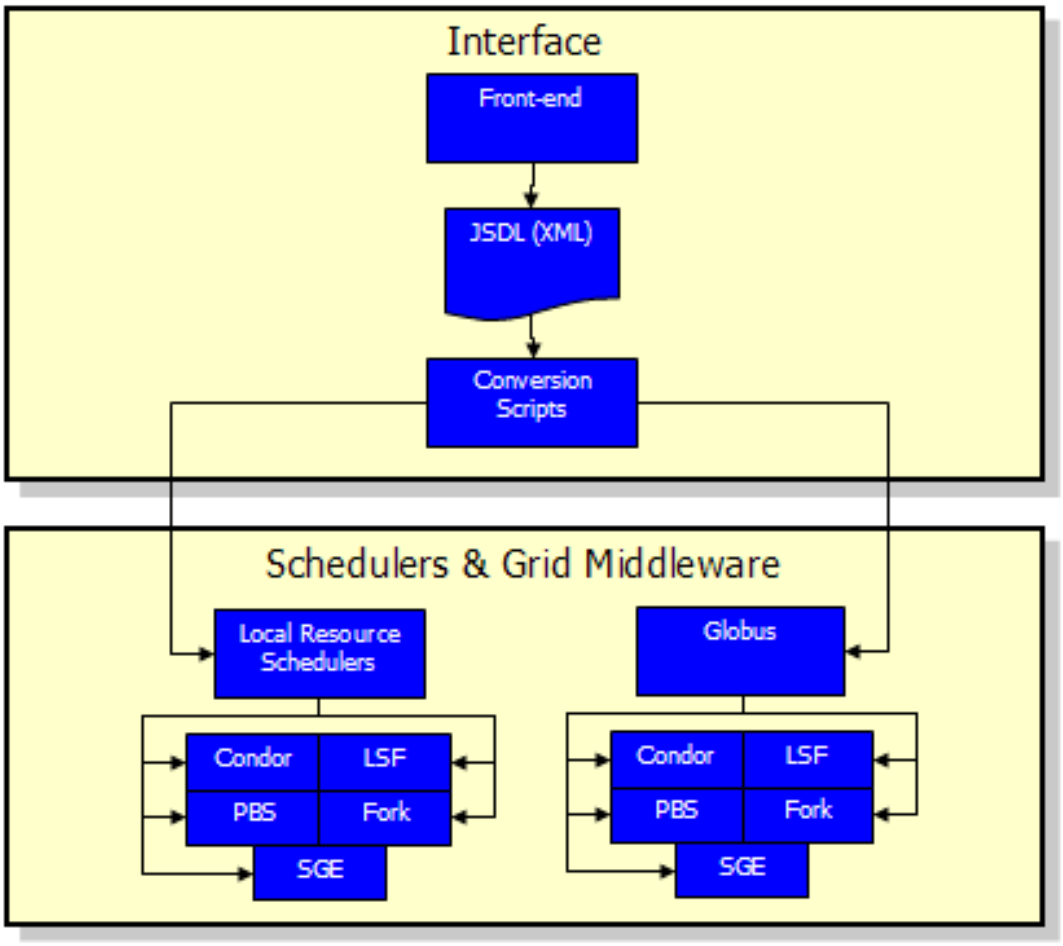

Figure 2.4: Conversion of custom front-end data into system neutral JSDL

application it has been designed for, namely— the parameter sweep application. The interface automates many of the complexities involved in building the parameter runs associated with such applications by use of a dynamic AJAX-based approach, discussed further in Chapter 4. A discussion on parameter sweep applications can be found in Section 3.3.1.

JSDL

In Section 2.2.5, a number of different batch schedulers commonly used in both cluster computing and Grid environments were discussed. It was noted that many of these schedulers had similar features and all accomplished similar tasks, but in different ways. In particular, the way in which jobs are represented in terms of submission scripts was discussed. Since many, if not all, schedulers have such different submission scripts, jobs are submitted in the Job Submission Description Language (JSDL) [Global Grid Forum, 2007], a format or language for describing computational jobs in the form of an XML document.

In order to make use of the JSDL in high level Grid tools, an interface can be designed with input fields such that the contents of those fields are parsed into the JSDL format. Once the JSDL XML document has been constructed, a script can be run that translates the XML document into a file that corresponds to the format of the particular scheduler for which the job is intended (see Figure 2.4). This process not only standardises the high level tool, but also makes management of the script formats easier when new versions of scheduling software become available.

In order to build high-level Grid tools, a discussion of Web technologies and their role in Grid computing will be presented in the next section.

2.5 Web 2.0

Since the early beginnings of the Web, there have been many technologies that have each built upon one another to create more usable interfaces. Since personalised websites were first developed, people have strived to make pages more interactive and intuitive for their visitors. These advances came in the form of JavaScript, where pages then had animation and seemingly dynamic properties embedded within them. Although JavaScript does not have dynamic properties and is a client-side technology, the ability to manipulate objects and produce effects still makes it popular. After JavaScript, developers looked at technologies such as Macromedia Flash [Codling, 2003] in order to make pages not only look impressive, but also be highly interactive, as communication could occur behind the scenes and have the page appear to be dynamic. The latest such technology, dynamic HTML or DHTML, has become popular since this technology does make pages dynamic and thereby enhances a user's experience since only portions of the page get updated at a time. This section aims to give an overview of a new development paradigm that incorporates many of the afore-mentioned technologies and the way in which it could enhance a user's experience.

Web 1.0 to Web 2.0

When the Web first became popular, its main use was as a source of information. In other words, users could enter a URL, visit a site and locate a piece of information they were looking for. The Web, in the early days, was therefore seen as a collection of documents (Web pages) that could be retrieved in order to gain knowledge on a particular topic [Berners-Lee et al., 1994]. From these early beginnings, the Web evolved into an interactive resource where users could sign up for services such as newsletters, create online email accounts and other such value-added services. This period in the history of the Web is known as Web 1.0. Since Web 1.0 was impersonal and merely a large pool of information, a way had to be found for users to create their very own Web that would reflect their lifestyles, would allow them to connect with friends, colleagues and family et cetera [O'Reilly Media, 2007]. In order to achieve this, new types of Web applications, now referred to as Web 2.0, were born [Cormode & Krishnamurthy, 2008]. Web 2.0 is characterised by personalisable websites such as Facebook[13], where users are able to create profiles, change the way their pages appear to the outside world and interact with the site in ways that were not possible with Web 1.0. Technologies such as AJAX, discussed in the next section, as well as DHTML are synonymous with Web 2.0 and are used in many of the wideley used Web applications today.

Since the differences between Web 1.0 and Web 2.0 are somewhat fuzzy, these differences can be illustrated by the way in which people used and consumed Web 1.0 information and how similar information is being used and consumed today. Table 2.2 shows some perceived transitions in the use of Web technology that has taken place from Web 1.0 to Web 2.0 [O'Reilly Media, 2007]. The move from purely content-based systems such as personal websites, to a more collaborative technology such as a blog is indicative of Web 2.0, as it encourages participation in the form of comments that can be left to bloggers. Similarly the philosophy of online encyclopaedias has died out somewhat as these online systems usually require a subscription fee. Since a wiki allows ordinary people to add and update information in it, wikis have become popular as the content

[13] http://www.facebook.com

Web 1.0	Web 2.0
Reading	Writing
Companies	Communities
Client/Server	Peer to Peer
HTML	XML
Personal Websites	Blogs
Online Encyclopaedias	Wikis
Content Management Systems	Content Management Systems
Publishing	Participation

Table 2.2: Perceived transition from Web 1.0 to Web 2.0

is not only always kept up to date, but cover specific or specialist topics, which encyclopaedias cannot.

AJAX

AJAX is short for Asynchronous JavaScript and XML and the term was first coined by Jesse-James Garrett in 2005 [Garrett, 2005]. AJAX represents a new approach to Web development in the sense that AJAX-based Web applications communicate with the Web server asynchronously, thereby making a website dynamic. The main feature of an AJAX-based Web application is its similarity to desktop software due to these dynamic properties. These features increase usability and make such applications more interactive as users do not have to wait for a page to refresh. The typical AJAX application makes use of a number of existing technologies including: DHTML for event-driven components; HTML/XHTML, CSS, HTTP, server-side scripting, JavaScript, XML/Document Object Model (DOM); and, most importantly, the XMLHttpRequest object which allows for asynchronous communication [Mahemoff, 2006]. Although the technologies of which AJAX is comprised have been available for many years, the use of them together is what AJAX represents. The term AJAX is then used to represent an application development paradigm and not a new technology. One of the main reasons that AJAX has become so popular in such a short time is that it makes use of existing technologies already present in many browsers. The ability of AJAX applications to work out of the box with no browser plug-ins, as is not the case with technologies such as Macromedia Flash, make it an attractive option for developers since the benefits to end-users are so great.

When the first Web applications and websites were deployed, the way in which users interacted with these sites was simply by clicking on links and being redirected to another HTML page. This approach worked well for many years, however, with the growing complexity of desktop applications and amount of effort being devoted to usability of such interfaces, Web applications were compelled to move in the same direction. The click and link style of websites has therefore slowly given way to more intuitive and user friendly desktop-application like interfaces with technologies such as DHTML and AJAX. In Figure 2.5 [Crane et al., 2005], a traditional Web application is illustrated. In the figure, a user makes a request for a Web application, the server then completes the request by running back-end business logic, and returns the Web page to the user's browser. This process is repeated as the user requests new information from the server, therefore resulting in a new page being served with each request. In the dynamic AJAX model, this process is somewhat different, (see Figure 2.6). When a user makes a request to a Web application, the server delivers the application to the client's browser. This application is not a static Web page as in the previous case, but a fully functional application that can make asynchronous calls to the server when certain events are triggered. When a user executes a function on the Web page, a request is sent to the server. The server then handles the request and

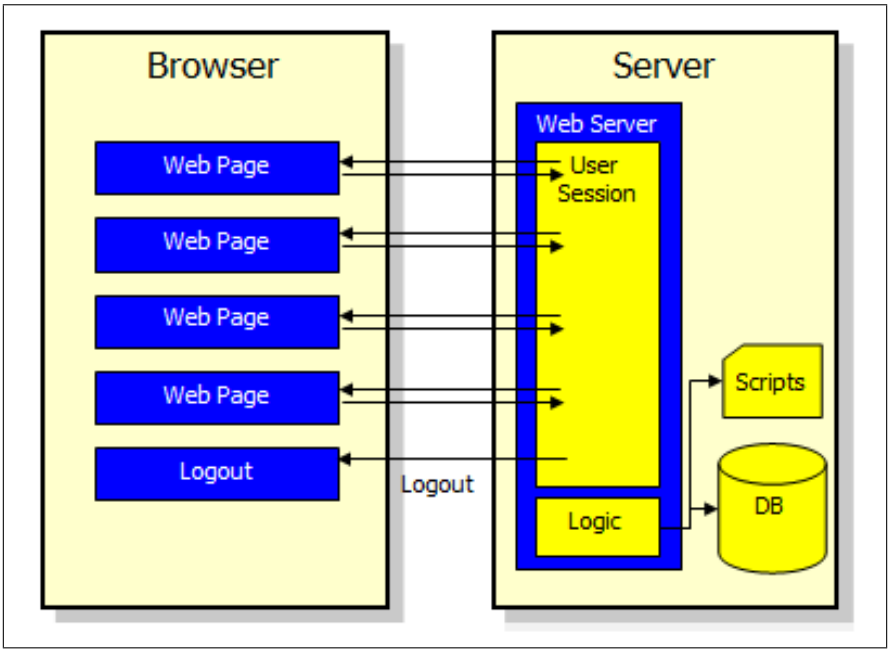

Figure 2.5: Traditional Web Application

sends data back to the client application, usually in some platform neutral format such as XML in the case of AJAX. Once the client application receives the XML document, the application updates the DOM in the browser with the new information and the page is therefore changed dynamically. Therefore, no Web pages are served with the AJAX model once the application has been delivered to the client, only small snippets of data. In Figure 2.7, an example of such AJAX functionality present in Google Suggest[14]is shown. In the figure, as a user starts entering a query into the search bar, Google Suggest provides potential search queries that the user might be interested in. So, for example, if the user starts to type the word "AJAX", Google Suggest starts off by suggesting "Amazon" as a possible search term for the first letter "A". Similarly as the user enters the "J", in Figure 2.8, Google Suggest changes the possible query space and displays it in the combo box below the search. The use of AJAX in Google Suggest is evident from the fact that the content displayed in the search bar is updated dynamically as the user types the search query.

Of significant importance to the underlying dynamic principles of AJAX is the way in which the DOM in the browser is dynamically modified by means of asynchronous calls to the server. This ability is what sets AJAX applications apart from traditional Web applications since the manipulation of the DOM is directly responsible for the dynamic property of AJAX. The W3C[15] defines the DOM as a platform- and language-neutral interface that allows the tree to be dynamically modified. Since the architecture of the DOM, like XML, takes the form of a tree, manipulation of objects within the tree is as simple as specifying the path to a particular tree node. This inherent feature of the DOM makes it easy to update only certain portions of the tree, where each node in the tree could be a component of a Web page, thereby resulting in the page being updated dynamically. By making pages dynamic with the use of AJAX, one achieves a number of efficiencies and characteristics that websites have strived for over the years. Since AJAX is dynamic and updates only a portion of a page on request, the end-user notices a decreased latency, with the exception of the first page load. Since only certain parts of the page get refreshed, the bandwidth requirements of the application are minimized since there is less

[14]http://www.google.com/webhp?complete=1&hl=en
[15]http://www.w3.org/TR/REC-DOM-Level-1/

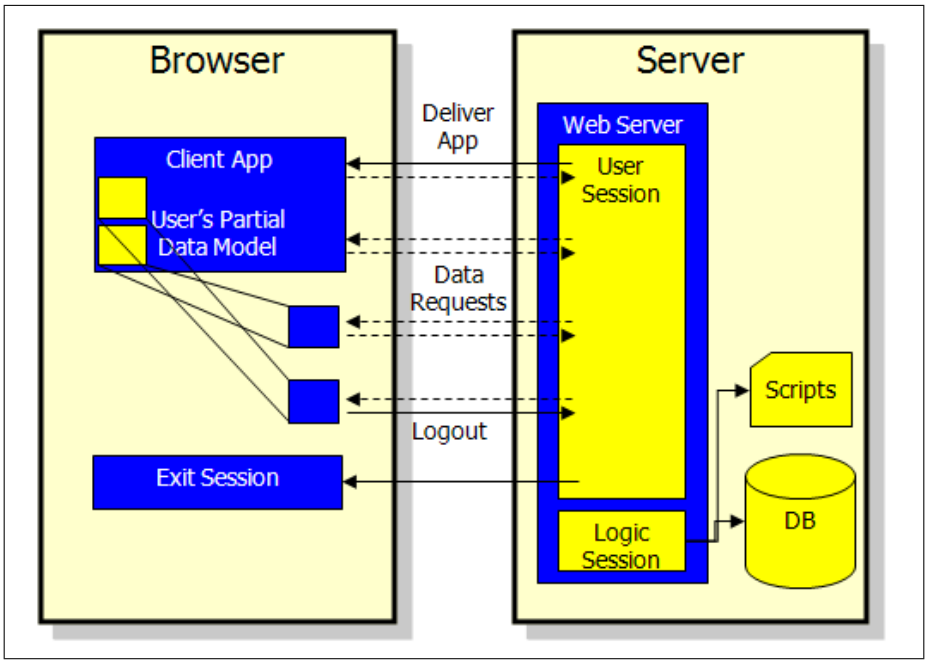

Figure 2.6: AJAX Web Application

Figure 2.7: AJAX Google Example I

Figure 2.8: AJAX Google Example II

data to be transferred between updates. Furthermore, the fact that pages load more quickly and that users do not spend time having to wait for entire page refreshes lead to an increased overall user experience. In terms of development, AJAX provides a solid model for developing Web applications since the componentized approach taken in the specification of an AJAX application separates elements such as basic components from style. This property of AJAX leads to a more structured design of the Web application and also increases the manageability of the code.

2.6 Summary

This chapter has provided an overview of distributed computing technologies including cluster, volunteer and Grid computing. The subtle differences between these various forms of computing were discussed in terms of both the problems these systems are geared to solve and the environment in which they operate. Furthermore, an overview of the relationship between Grid computing and scheduling was presented, as well as an overview of a number of popular schedulers. One of the main issues addressed in this chapter was the issue of Grid usability and the way in which low-level tools make use of such systems difficult for users with little HPC knowledge. Ways in which usability can be enhanced were discussed in order to provide the grounding for further discussion later on in this thesis. An important outcome of this chapter is the recognition of the fact that Grid software needs to be more usable if users are to adopt it in the long-run. If this obstacle is not overcome, Grid computing may not be adopted by enough users to make it economically feasible.

Chapter 3

Condor

3.1 Introduction

Since the Condor system is one of the main components upon which this research is built, this chapter will provide an in-depth overview of Condor.

As mentioned in Section 2.2.5, Condor is a batch scheduling system designed at the University of Wisconsin at Madison in the United States of America [Litzkow et al., 1988]. Condor schedules jobs to any available machine that meets a set of job-specific requirements on the Condor network. Since Grid middleware such as Globus, discussed in Section 2.2.4, cannot schedule jobs at the machine level, such middleware relies on the presence of local resource managers such as Condor to allocate jobs to machines. Condor has a variety of mechanisms in place that allow it to make use of multi-purpose machines such as desktop workstations as well as dedicated machines and clusters. This chapter will discuss how Condor achieves its goal of scheduling resources, the different ways in which it does so and the way in which jobs are submitted by users.

3.2 System Design

3.2.1 Condor Architecture

The Condor scheduling software is able to utilise a variety of resource types in order to cater for a variety of applications. The first such resource type is the desktop workstation. Condor's role in CPU or cycle scavenging from desktop workstations is best described in the 1988 paper [Litzkow et al., 1988] entitled *"Condor — A Hunter of Idle Workstations"*. Since the Condor research group is based at a university, they saw the potential of idle laboratory workstations on their campus and implemented a mechanism whereby this otherwise wasted compute power could be utilised. Condor provides a variety of policies that can be configured on such desktop workstations which give the user of the workstation control over when and for how long his/her machine will be utilised on the Condor network. The next resource type catered for by Condor is the cluster. Condor has support for both the dedicated cluster, a cluster dedicated to Condor jobs only, and the non-dedicated cluster which is used on a scavenging basis when not being utilised by its owner.

In order to schedule jobs to the resources just described, the Condor system communicates with daemons on all Condor-enabled resources on a regular basis to gather information on the status of the machines on the Condor network. To illustrate how Condor communicates with resources on the Condor network, Figure 3.1 is a schematic representation of the Condor communication layer. In the figure, dashed lines represent process spawning by the master process

and solid lines represent system status updates in the form of the Condor Classad language. Classad is short for classified advertisement and such a Classad is periodically sent to the master node. This advertisement contains information pertaining to the status of the node from which it came. In other words, one can think of a Classad as an XML document where each tag represents a particular attribute of a machine such as the machine name, amount of memory, disk space, average load, et cetera. The information present in the Classad is used to build a snapshot of the Grid at a particular point in time. This information can then be used to schedule jobs. An XML snippet of a Classad is shown in Figure 3.2.

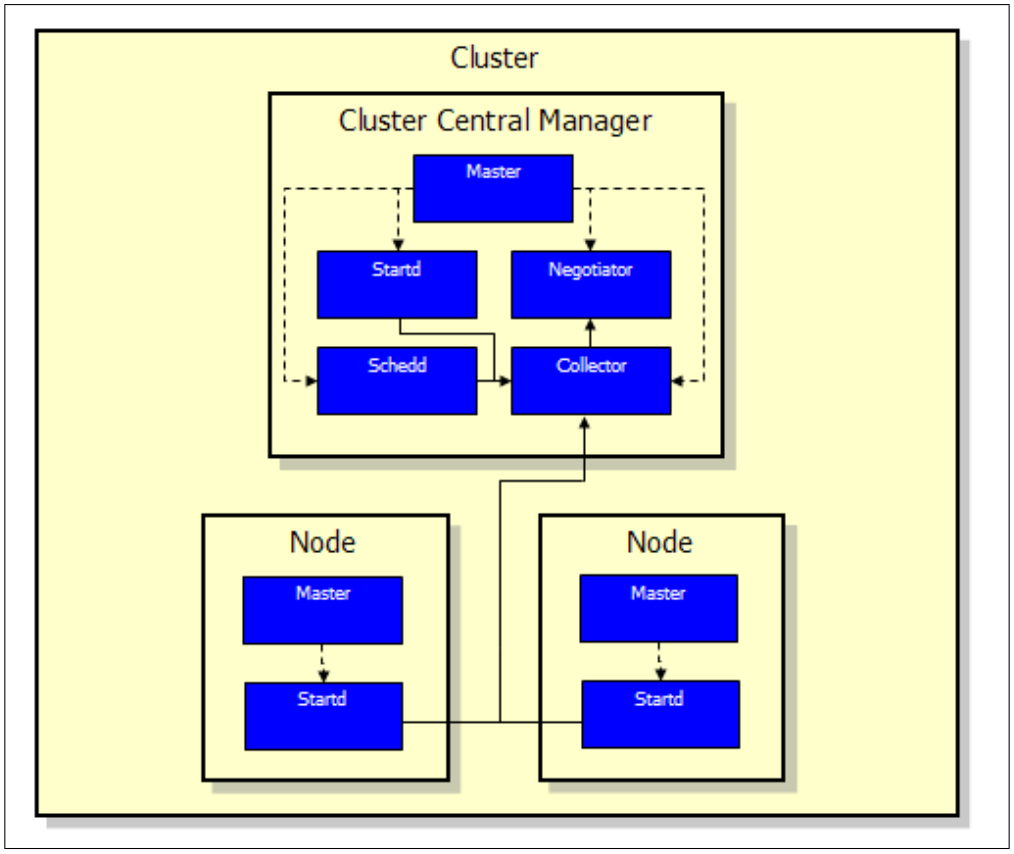

Figure 3.1: Condor Communication Layer

The Condor system is made up of many different daemons that exchange information using the Classad mechanism described above. Figure 3.1 gives a graphical view of these daemons. In the figure, the primary daemon is called the master daemon and spawns all other daemons that Condor relies on. The master daemon also is responsible for keeping all the other daemons up and running. The startd daemon represents a resource or machine in a Condor pool and runs on all Condor-enabled resources. The schedd daemon allows jobs to be submitted from a Condor-enabled machine. It is important to note in Figure 3.1 that cluster nodes do not have the schedd daemon running as they will generally not be used to submit jobs. A desktop node for example might have a schedd daemon running as the user owning the machine might want to submit jobs from it. Therefore, only a central manager or a dedicated submit node will have the schedd process running. The next daemon is called the collector daemon and is responsible for collecting status information from all Condor pools. The negotiator daemon matches jobs to machines given a set of requirements for each job. The negotiator uses information collected by the collector daemon. The collector and negotiator daemons are only present on the central manager as execute nodes are not concerned with collecting information from other nodes or matchmaking respectively. The daemons just described collectively allow the Condor software to form a global picture of the status of the resources on the Condor network.

```
<?xml version="1.0"?>
<!DOCTYPE classads SYSTEM "classads.dtd">
<classads>
<c>
    <a n="MyType"><s>Machine</s></a>
    <a n="TargetType"><s>Job</s></a>
    <a n="Name"><s>vm1@ed.cs.uct.ac.za</s></a>
    <a n="Machine"><s>ed.cs.uct.ac.za</s></a>
    <a n="Rank"><r>0.000000000000000E+00</r></a>
    <a n="CpuBusy"><e>((LoadAvg - CondorLoadAvg) &gt;= 0.500000)</e></a>
    <a n="COLLECTOR_HOST_STRING"><s>ed.cs.uct.ac.za</s></a>
    <a n="CondorVersion"><s>$CondorVersion: 6.8.4 Feb  1 2007 $</s></a>
    <a n="CondorPlatform"><s>$CondorPlatform: I386-LINUX_RHEL3 $</s></a>
    <a n="VirtualMachineID"><i>1</i></a>
    <a n="VirtualMemory"><i>1184788</i></a>
    <a n="Disk"><i>7002702</i></a>
    <a n="CondorLoadAvg"><r>0.000000000000000E+00</r></a>
    <a n="LoadAvg"><r>0.000000000000000E+00</r></a>
    <a n="KeyboardIdle"><i>93447</i></a>
    <a n="ConsoleIdle"><i>358958</i></a>
    <a n="Memory"><i>1000</i></a>
    <a n="Cpus"><i>1</i></a>
```

Figure 3.2: Condor ClassAd snippet

3.2.2 Heterogeneity

As mentioned already, Condor supports a variety of resources including desktop workstations, servers and clusters (both dedicated and non-dedicated). What has not been mentioned, however, is the support Condor has built-in for heterogeneous computing. Since it is inevitable that a variety of different hardware and software configurations exist in many organisations and production environments, Condor has built-in support for differing hardware platforms and operating systems such as Windows, Linux, UNIX/BSD and Macintosh.

3.2.3 Application Support

As discussed in Section 2.3, two main types of parallel applications exist, namely fine- and coarse-grained applications. Condor has support for both these application types and can therefore accept both MPI and parameter sweep type applications, for example. Condor has support for many job types defined as "universes". The "vanilla" universe refers to applications that have not been linked against the Condor software and run as is over the Condor network. Applications falling into this category are typically coded in advance, are closed-source or downloaded in binary form from the internet. The next universe type is the "standard" universe. This universe covers applications where the source-code is available and where such applications have been linked against the Condor software. Such applications have the benefit of the Condor checkpointing mechanism whereby jobs can be checkpointed in the event that a remote execute node becomes unavailable. The job can then be moved to another available machine that can resume computation. Finally, Condor has support for both MPI and Java jobs in the "MPI" and "Java" universes respectively. It should be noted, however, that MPI jobs require dedicated resources as such jobs tend to be fine-grained and require a fixed resource set.

```
executable = /usr/local/indexer
universe = vanilla
requirements = Memory >= 32 && OpSys == "LINUX" && Arch =="INTEL"

should_transfer_files = YES
when_to_transfer_output = ON_EXIT
output  = output.$$(Process).index
error   = error.$$(Process).index
log     = log.$$(Process).index

input   = some_doc1.txt
arguments = some_page1.html 1
queue

input   = some_doc2.txt
arguments = some_page2.html 2
queue
```

Figure 3.3: Condor sample submit file

3.3 Job scheduling

The purpose of all the Condor daemons is to gather data on the status of the resources present in the Condor environment so that jobs can be executed upon such resources, as well as submit and monitor jobs. However, in order to match a job to a specific resource, some work is involved on the part of the user.

3.3.1 Submit File

Before a job can be launched on the Condor network, the user must first create a job submission file called a submit file. This file contains all the necessary elements that Condor needs to match a job to the resources upon which the job must run. This file must therefore include a universe identifier, the path to the executable to be run, a logical requirements statement specifying which resources to utilise, the specification of input and output files as well as any input arguments needed by the application. There are many more options, however, the ones just mentioned are the most common. An example of such a submit file is shown in Figure 3.3.

The submit file starts off with the specification of the executable. In the case of Figure 3.3, the executable is a custom written text indexer that can be run in either serial or parallel mode by specifying a set of input arguments. Next, the universe is specified. Being the vanilla universe, the user does not wish to make use of features such as checkpointing and effectively stipulates that the binary should be run as is on remote nodes. The resource allocation directive, in the sample submit file shown in the figure, is the "requirements" keyword. This keyword specifies which machines on the Condor network should receive and attempt to execute a job. The user specifies the memory, disk, CPU, operating system and platform requirements of the job by using this directive. There are many ways to specify where a job is to run and only a subset are shown in the figure. It is also possible to have Condor choose between different platforms and operating systems by providing executables for each of these differing scenarios. In other words, multiple executable directives can be used to specify a range of different platforms and operating systems. For the sample job present in the figure, the requirements directive will

force Condor to choose machines with greater than 32MB of main memory, running the Linux operating system and having an Intel 32-bit architecture.

Now that the executable and resources have been specified, a user would typically specify how the output and file transfer mechanisms should operate. In the figure, the `should_transfer_files` and `when_to_transfer_output` directives are used to prompt the Condor system to copy any files needed by the job to the remote execute nodes and on completion copy any data files back to the submission host. These directives are needed in an environment that does not have a network file system. The file transfer settings are followed by the output file settings. A user can specify that a job is to output not only the result, but also files that can be used during troubleshooting of failed jobs. These typically include both error logs and console output logs. It also is possible to use macro directives such as those shown in Figure 3.3 to automatically assign unique names to output files.

Parameter Sweep Applications

The final step in a Condor job submission is to specify the way in which the executable operates, and since Condor is a batch scheduling system, the number of times the job is to be run. Figure 3.3 shows two runs of the `indexer` application, evident from the two `queue` commands present in the file. The `queue` command prompts Condor to create a new sub-job under the umbrella of one job cluster. In other words, if the user wishes to run the `indexer` application twice with differing input files and input arguments for each individual run, such directives are specified before a `queue` statement. Applications executed in this way are known as parameter sweep applications— multiple executions of the same application using different sets of input data, run one or more times. This dissertation focuses mainly on parameter sweep applications as a means of job submission. Due to data gathered during interviews with scientists making use of Grid technology as well as noticing shortcomings in existing Grid portals to support parameter sweep applications, developing enabling tools to support this application type was found to be useful. Chapter 4 will provide more insight into why the parameter sweep application type was selected for use in this research.

Condor will utilise the global job specification directives described in the previous paragraphs on each sub-job, but will augment each run with the corresponding `input` and `arguments` directives. If, for example, 1000 such jobs are to be run, Condor will match each sub-job to an available resource on the Condor network and monitor each sub-job separately.

3.3.2 Submission Process

Once the submit file is complete, the job can be submitted to the Condor system by utilising the `condor_submit` command. During submission, Condor creates a spool directory for each sub-job on the submit machine. This folder houses all the input files and executables that make up the job. Depending on the size of the data files making up the job, submission can take a long time. Once the job is submitted, however, the Condor system submits and monitors all aspects of the job automatically. If a sub-job is terminated due to a machine going down, for example, Condor will automatically restart the job. Such error recovery mechanisms ensure that the job runs to completion, thereby removing the burden of administration from the user. The user can view the status of his/her job(s) by using the low-level command-line `condor_q` command and also view the status of the Condor network at any time by making use of the `condor_status` [Condor, 2007b] command. These commands do not work for flocked pools as they only show information pertaining to the central manager for which the command was executed. A user wishing to obtain information on other pools on the Condor network must specify the address of the remote central manager when using these commands to view the status

of such a pool.

Example

An example of a Grid creation, submission and monitoring procedure is outlined below for a typical Grid job, assuming a local scheduling system is used. This example is by no means exhaustive, but serves to illustrate the complex process required to submit a job manually to a Condor-enabled Computational Grid. The example below elaborates on the example presented in Chapter 1.

1. **Using a shell, obtain information on the status of the Grid.**
2. **Obtain information on the available resources on the Grid in terms of operating systems and architectures.**
3. Write or use an existing Grid application and compile, taking the above information into account.
4. **Use a reference manual to describe the job in terms of Grid resources by creating a submission file.**
   (a) Define the type of application.
   (b) Define the resources required by the job in terms of the information supplied by the Grid status commands.
   (c) Define the input files and arguments for the job.
   (d) Define the output settings of the job.
   (e) Define file transfer settings (NFS / FTP).
5. **Transfer input files if needed.**
6. Submit the job using the Grid job submission utility.
7. **Obtain status information on the Grid job just submitted.**
   (a) Check the status information for errors using the job queue utility.
      i. If the job is running on a remote pool or cluster, supply the address of the remote cluster.
   (b) If errors are found, manually remove the job using the job removal utility.

As already stated, the above example deals with a manual Grid job submission. The interface designed as part of this research, however, automates many of the steps outlined above. The steps printed in bold type (as well as their non-bolded sub-steps) are all dynamically handled by the interface, thus reducing the overall effort required on the part of the user in submitting a Grid job.

3.4 Flocking & Grid Computing

There are two mechanisms that Condor has built-in to support being Grid-enabled. Flocking is Condor's attempt at Grid computing whereby pools of machines are connected to a "remote" central manager [Butt et al., 2003]. The term "remote" in this context refers to a central manager not belonging to the pool that has flocking enabled and might belong to another pool, or be

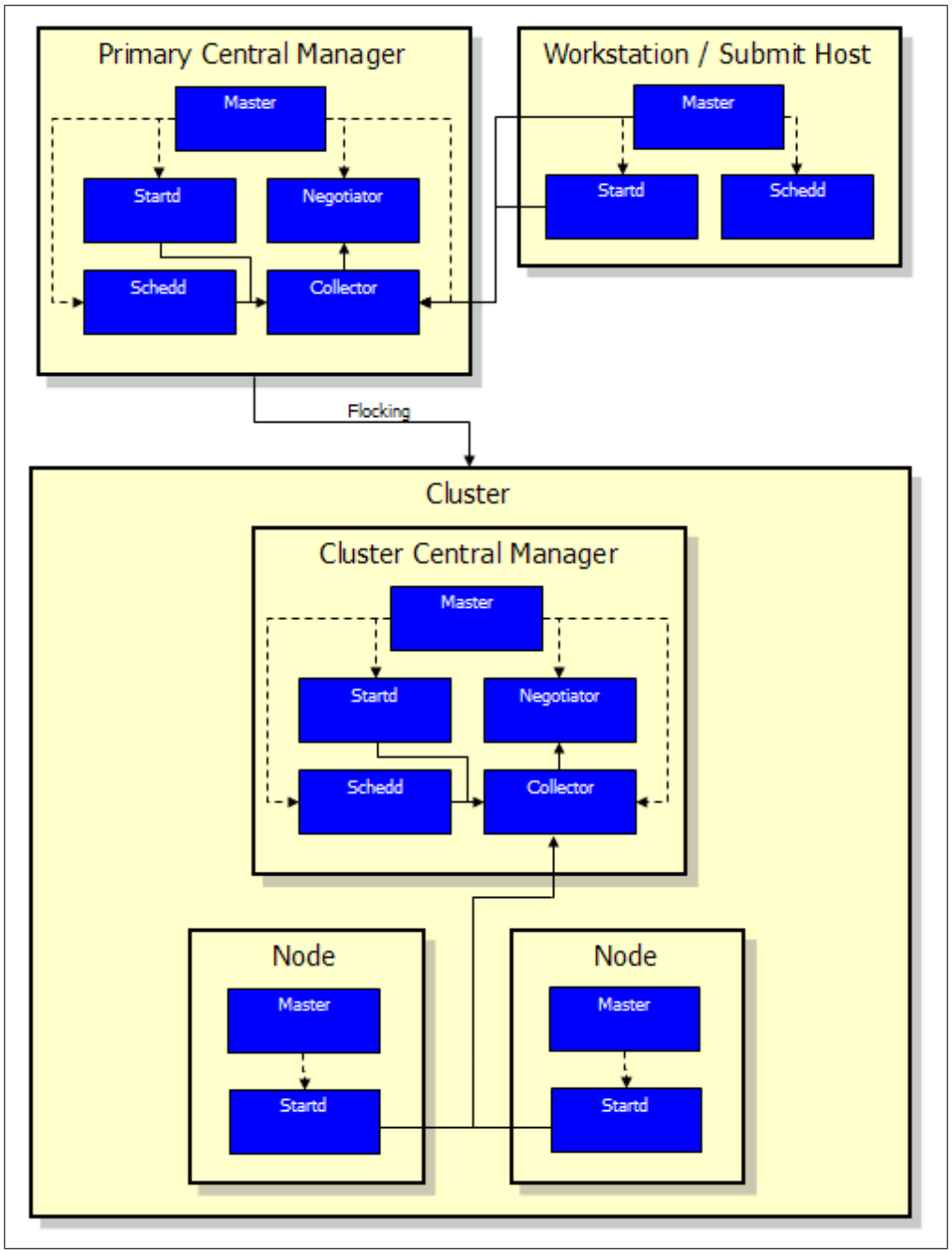

Figure 3.4: Condor Flocking Architecture

entirely on its own as a primary central manager. Flocking allows jobs to migrate from one pool to another pool if such jobs are unable to run at the original pool to which they were submitted. Flocking requires that Condor be installed on at least two separate pools— these could be clusters or groups of standalone workstations— and subsequently configured to flock (receive) jobs from a remote central manager. Flocking-enabled clusters or pools can be configured to flock from multiple central managers, which in turn can flock from other central managers. Figure 3.4 shows how two central managers can benefit from the flocking mechanism. Since the standalone central manager in the figure has only one machine connected to it, it would not be of much use as a High Throughput Computing tool. However, with the addition of the cluster by means of flocking, jobs submitted to this central manager can now be run remotely without the need to have access directly to the remote nodes. This decreases cost of administration and also increases security.

The process of flocking, however, leads to problems in practice. Condor requires the submit host to have direct access to the nodes upon which jobs eventually might run. Since clusters are often firewalled away from public access, the submission host has no way of directly accessing such nodes. For this reason, the Condor team developed Generic Connection Brokering (GCB) [Condor, 2007a], where the central manager of a cluster hosts a relay server through which the submission host can access the internal Condor nodes.

The second Grid-enabling mechanism that the Condor system has built into it is that it is able to interoperate with the Globus Toolkit. This interoperability with Globus known as Condor-G [Frey et al., 2002], as well as the scheduling mechanism and job monitoring capabilities of Condor itself, can transform the basic Condor network into a capable Grid computing environment.

Above and beyond all the functionality mentioned so far, the main Condor Project includes numerous other smaller research projects. These projects make software available that is critical for the successful operation of a Grid. For example, systems such as Condor BirdBath[1] aim to move the Condor scheduler into the world of Web services [W3C, 2004]. These services currently allow jobs to be submitted and monitored using a simple API. This is ideal for the development of Web-applications, and was used during the development of parts of the Web interface.

3.5 Summary

The Condor scheduling system was chosen for use as the primary scheduler in this research. Its Grid-like features and popularity made it a suitable candidate for the experiments and software which was developed as part of this research project. The features discussed in this chapter with the exception of checkpointing and Condor-G have all been used in some way or another during the development of the Web interface. For this reason, it was deemed necessary to elaborate on the system itself as many design decisions were based somewhat on the Condor approach. Although other schedulers do exist, Condor's features, widespread use and Grid integration components were instrumental in selecting it for use in this research. Due to time constraints, only Condor was thoroughly researched in order to ensure that its features could be used to prove the research objectives.

The rest of this thesis will discuss the approach taken in developing the Web interface built on top of Condor and one other scheduling system, namely IBM LoadLeveler which is similar to Condor in many ways.

[1] http://www.cs.wisc.edu/condor/birdbath/

Chapter 4

Prototype Development and Evaluation

4.1 Introduction

Graphical user interfaces to complex software systems have been one of the most influential promoters of software usability since the early beginnings of personal computers (PCs) [Wikipedia, 8 09]. With the advent of PCs, it was necessary to find a way in which ordinary people could interact with their computer in an intuitive way in order to accomplish a simple task such as typing a document. Previously, the use of computers was limited to people in the computing profession as low-level and difficult-to-use interfaces were commonplace. However, the design and implementation of an effective user interface has its challenges. In a 1988 paper by Mackinlay [Mackinlay, 1988], the research conducted into user interface design showed that it is not only creativity that builds good interfaces but an understanding of the underlying theory as well. The creativity referred to inspires a good design, whereas the theory refines, tests and extends this design. Twenty years later, these principles are still used in modern interaction design processes [Preece et al., 2002]

When attempting to build any system that is to be used by a large number of users, it is important that the opinions of these users are considered early on in the design process. With particular reference to the Mackinlay paper [Mackinlay, 1988], the user evaluations would represent the theoretic part of the design process. One of the main reasons to consider users is that they are ultimately the people who will use the system once it is built. It is therefore better to design a system that meets the ideals of users rather than the ideals of programmers. Failure to consult users can result in a system that either does not provide features expected by users or is not usable in the way in which users would expect and therefore not utilised in the manner envisioned. Furthermore, consulting users during the design process can lead to early user adoption of a system and even result in such users becoming alpha testers of the system.

One of the most important questions that this research investigates is whether or not it is possible to create a high-level Grid interface that both specialists (people familiar with Grid and HPC technology) and non-specialists (ordinary scientists) can utilise in order to submit an application to a Grid. In order to help answer this question, it was deemed necessary to first build a paper prototype of such a high-level Grid tool— in this case a Web application, capable of at the very least submitting, monitoring and viewing the status of a computational job. The next phase necessary to help answer this particular research question was to conduct user evaluations of this prototype. Only once these evaluations were conducted was an initial software prototype of the system built. The evaluation techniques used, the methodology behind the evaluations and the results obtained from these evaluations will be discussed in this chapter.

4.2 Methodology

Since a large portion of this research involved the implementation of a high-level Grid tool, the success of the research is thus largely determined by the usability of the tool itself. The need then to build prototypes and have these prototypes evaluated by potential end-users is therefore important. Firstly it was necessary to find out how various departments on a University campus utilise HPC resources before any prototype could be developed. The design phase was broken up into two parts. An overview of these phases will now be presented.

4.2.1 Phase 1 : Requirements Gathering

The first phase towards the design of a prototype was aimed at gaining insight into the types of applications being run on HPC equipment across the university. Several departments known to make use of HPC equipment were contacted during this phase of the design process and meetings were set up with researchers from these various departments. Five senior researchers from Physics, Chemistry, Computer Science and Health Sciences were interviewed and data was gathered in order to build a complete profile of the types of applications they run as well as the resource requirements of these jobs. In order to gather the resource requirements, the size of input datasets, output file sizes, types of machines being used, hardware requirements, job length and many other such detailed specifications were researched during this phase. Other than job related information, it was also important to determine what cluster software and operating systems these departments made use of in order to determine if scheduling software, important in Grid computing infrastructures and discussed in Section 2.2.5, was utilised. If these departments all made use of the same scheduling software, for example, it would make sense to incorporate this scheduler into the high-level interface.

Another important aspect of HPC jobs that was considered during this first phase was the use of user interfaces for submitting jobs— a strong focal point of this research. Researchers were asked what type of interfaces they preferred using, which type of interfaces they used most often and what they liked about their preferred interface type. The results from all the studies discussed in this section are reported at the end of this section. The full questionnaire used during the interview sessions can be found in Appendix B. The questions present in the questionnaire were not answered in written form directly by the researchers but took the form of an interview or discussion. The questions were therefore used as a guide for the interviewer and the inclusion of this questionnaire serves only to inform the reader of the type of information gathered during Phase 1. In order to gain a better understanding of the interface design process mentioned thus far, Figure 4.1 provides a graphical representation of the design phases that made up the usability study. There was only one refinement cycle due to time constraints, however, for a production system these cycles could be repeated many times until the interface meets certain predetermined requirements where usability is concerned.

Once the requirements discussed so far were gathered, the next step was to build a paper-based prototype Web interface based on this information. The design of the paper prototype was based on basic Web interface layouts. Some functionality, such as the submit interface, was constructed by making use of fields described in the JSDL (see Section 2.4.2). Although the paper prototype interface was not constructed using any user feedback other than that of the pilot interview sessions, the aim was to try to create an interface that a large number of users would deem usable and also make use of features that such users are familar with from exposure to existing Web applications. A description of the actual interface design is presented in Chapter 5.

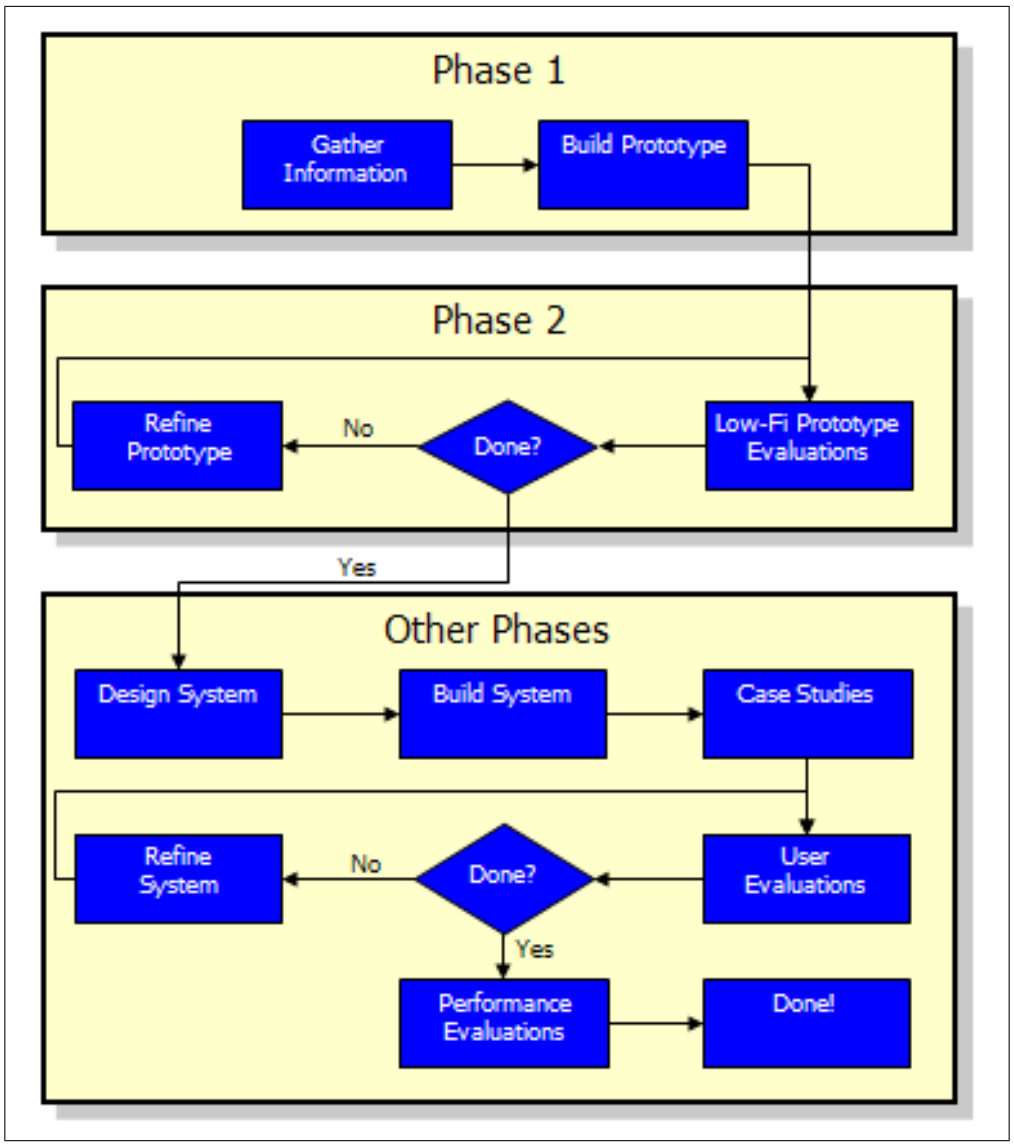

Figure 4.1: Interface Design Process

Results

The requirements gathered during initial interviews and meetings held with computational scientists from various faculties were instrumental in developing the tools built as part of this research. Without a clear understanding of the needs of such researchers and, more generally, users of Grid systems, it would be futile to develop applications with the hope that all their needs would be met. It was for this reason that informal discussions were held with these researchers, based on a series of focused questions. After consolidation of all the data gathered from the researchers it was found that they simply wanted a system that would make Grid computing platforms, as well as other high performance computing platforms, more accessible. In order to achieve this goal, a system that was capable of submission and monitoring of a Grid job was emphasized by researchers to be the most important. Other features that such tools could possibly support were well received, but the researchers made it clear that the main emphasis should be placed on the submission and monitoring tools. Furthermore, after determining the types of applications that such scientists generally made use of, and factoring in the constraints imposed by a Grid-based job execution approach, the parameter sweep or coarse-grained approach was chosen as the fundamental Grid job type to be supported by the tools that were to be developed.

The results from these interviews, although covering a broad spectrum of possible system designs, provided a good starting point from which the paper prototypes took form.

4.2.2 Phase 2 : Evaluation

As mentioned above, the first system prototype was built using the paper prototyping method of prototype design. Paper prototyping is a well known vehicle for prototype development, has been in use since the mid-80s and is still widely used today. Paper prototyping is done by designing a prototype of a system by making use of paper-based products such as folio and post-it notes, presenting the prototype to a user and asking him or her to use it as if it was a real system [Sefelin et al., 2003]. This approach to usability testing has several benefits, the most prominent being that it is quick and cheap. Since time is often the most limited resource in a project, the use of techniques such as paper prototypes as a vehicle for rapid evaluation are common. Appendix A contains images of the original paper prototypes used during this part of the design process. This section will present the way in which the user evaluation sessions were conducted.

Prototype evaluations

For the results of a prototype evaluation to be credible, it is necessary to consider a wide variety of potential end-users of the envisioned system to perform the prototype evaluation. In other words, interviewing a relatively small group of users with differing skill sets who are able to give different views on the prototype with respect to functionality, complexity and layout is more useful than considering a large group of users from the same discipline. It was therefore decided to conduct prototype evaluations by considering users from three distinct backgrounds. The first group of evaluators selected had skills in Computer Science. The aim of choosing a group consisting of purely Computer Science students was to gather feedback on the layout of the Web interface as all these students had done a course in Human Computer Interaction. The second group selected also had Computer Science background, but were knowledgeable in the field of HPC. The intended goal with the choice of this group was to gather information pertaining to the functionality of the high-level tool with respect to HPC resources as well as comments on the layout of the interface. The third and final group was a group of evaluators who do not have any Computer Science experience, but who use HPC resources as part of their research. Since this research is concerned with building a user interface that is usable by non-Computer Science researchers, the opinions of this group are of utmost importance. Four evaluators from each of

these groups were selected in order to perform the prototype evaluation.

The prototype evaluation session itself consisted of more than just the paper-based prototype evaluation of the proposed high-level Grid tool. Each session started off with an overview of Grid computing if evaluators were not familiar with the concept, as well as an overview of batch schedulers and scheduling concepts. It was deemed important to provide evaluators with a basic knowledge of the underlying infrastructure which the proposed tool was meant to abstract in order to gather more useful feedback. Once it was clear that the evaluators had a basic understanding of the principles of Grid computing, sample interfaces to both an existing Grid portal and a network management system were shown and briefly discussed. Evaluators were then asked to comment on the layout as well as the effectiveness of the flow of these interfaces. After comments were made, evaluators were asked which of the interfaces they preferred and were also asked to give reasons for their choice. The two interfaces were specifically chosen for their distinctive use of graphical components and their differing layouts. The first of the two interfaces, namely the Gridport interface [Dahan et al., 2004] shown in Figure 4.2, is indicative of the traditional style of Web application development. Functionality is separated into tabs requiring users to change from one mode to another in order to accomplish a task. The lack of graphical elements in this interface is also typical of this style of Web application development. When comparing this interface to the more graphical and clearly more modern network management interface (see Figure 4.3), the differences are clearly discernible. However, modern interfaces with fancy graphics might not necessarily be welcomed by the research community and therefore these evaluations were constructed in order to determine which type of interface is preferred for a research tool. Once evaluators gave feedback on these two interfaces, they were shown a paper prototype of the interface built from feedback gathered from the phase 1 interview sessions. An overview of the functionality of the prototype was provided to the evaluators who were then asked to comment on possible layout problems, feature suggestions, interface flow irregularities and so on. Evaluators were also instructed to ask questions if a part of the interface did not make sense, which would help in determining where more design emphasis could be placed.

Results

As discussed above, evaluators were shown two separate interfaces, one relating to Grids and one of a network management system. The evaluators were subsequently asked to comment on these interfaces in terms of layout, intuitiveness and functionality. The two interfaces are shown in Figure 4.2 and Figure 4.3 respectively.

Viewed from a high-level, evaluators seemed to prefer the network management interface simply because of its modern appearance, although it was found to have many downfalls. The Gridport interface, on the other hand, was not the preferred choice by the majority of evaluators, however many positive comments were made in its favour. The same cannot be said for the management interface. The fact that the evaluators, on the whole, preferred an interface for which they did not have many positive comments was surprising. The appearance of a system therefore appears to play a substantial role in its perceived usability. The rest of this section will present more in-depth results gathered from evaluators pertaining to each interface.

Gridport Interface

The Gridport interface is indicative of the traditional style of Web application development. The fact that evaluators deemed this interface to have a good layout, seems easy to use and has a "good step-by-step flow" indicate that this classification falls in line with this development style. The reason for this is that traditional Web applications tend to take a methodical, task-

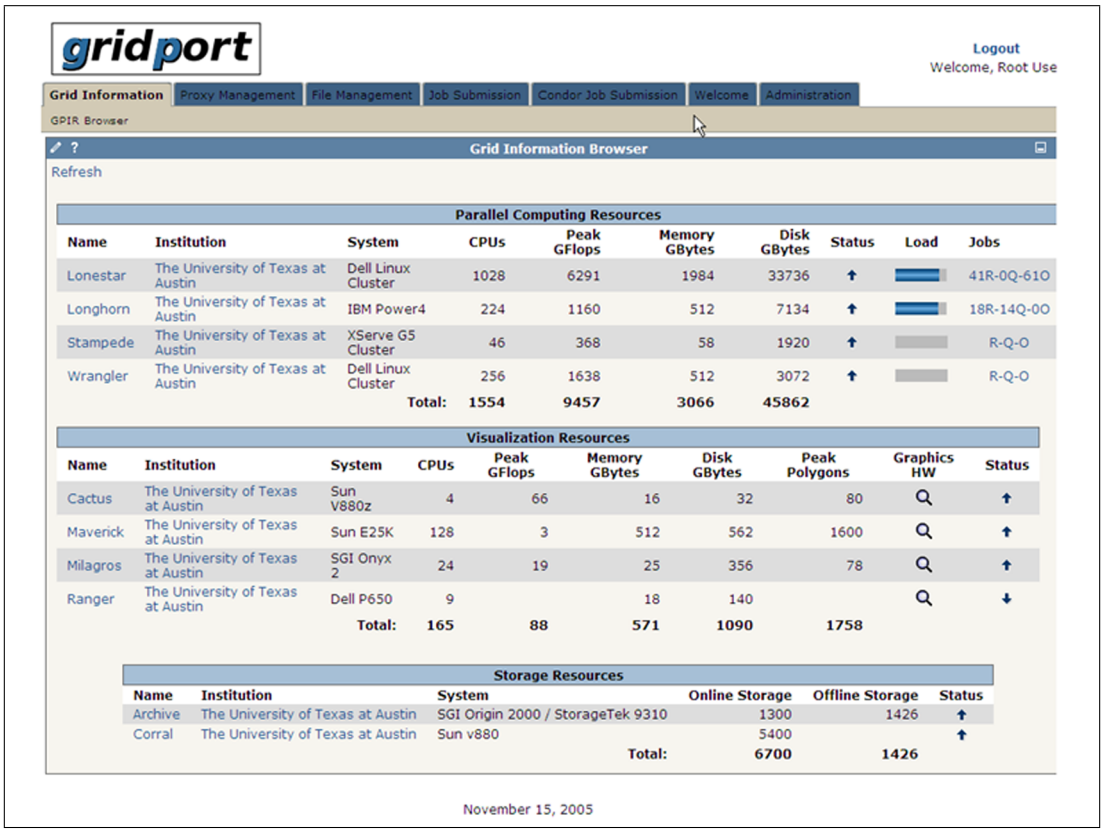

Figure 4.2: Gridport Interface - Sample 1

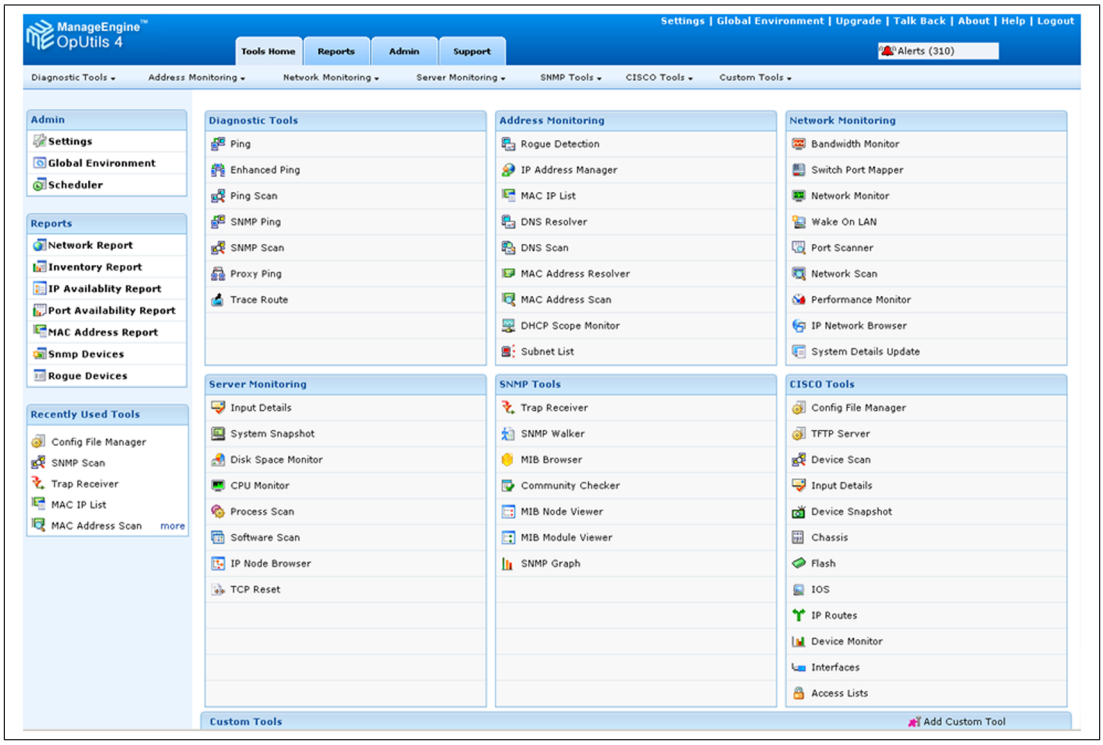

Figure 4.3: ManageEngine(TM) OpUtils 4 Interface - Sample 2

orientated approach to their design and thereby allow a user to step through a process to achieve some desired outcome. It was clear from many of the interview sessions that this approach is still popular and many of the evaluators preferred this approach. However, many comments relating to the usability of the actual functionality present in the interface were negative. Many of the comments received had to do with the lack of dynamic information from the interface. The fact that one has to, for example, click a refresh button to see task progress was a major drawback for many evaluators. Similarly, the lack of graphical elements makes the interface feel as though "textboxes have been slapped onto a script" to quote directly from one of the evaluators.

Another major concern about the interface was the amount of detailed information displayed at a time. Evaluators expressed doubts about displaying too much technical information as doing so could confuse new users of the system. The solution provided was to hide advanced features and information and allow for basic and advanced modes. These features allow new users to get to grips with the basic interface functionality and allows them to change the appearance of the interface as they gain more experience with the system. On the other hand, a small number of evaluators did not mind large amounts of technical information being displayed from the start as long as there was adequate help functionality. These results clearly indicate that there is a need for information hiding, however, the level at which this is done must be carefully considered in order to satisfy both experienced and novice users.

In terms of help functionality, a recurring theme with all sample interfaces shown was of there being good help available at each step in an online transaction. Both the Gridport and network management interfaces lacked adequate help functionality according to the evaluators. Although both these interfaces had a help option, evaluators noted that they preferred help along the way in addition to a complete help guide. Help text associated with textboxes, perhaps in the form of tooltips would make the interface more usable.

Network Management Interface

The network management interface turned out to be the more popular of the two interfaces shown to evaluators. However, as mentioned above, there were very few positive comments about its layout. Many evaluators found the interface to be too cluttered. Words such as "busy" and "feature rich" were used to describe the layout of this interface, the result of which was deemed to be overwhelming to new users of the system. Although many evaluators agreed that the interface was intuitive and that finding functionality would be simple considering the amount of information being displayed when logging in, the number of icons being displayed was considered to be far too many. Evaluators noted that trying to find a particular function among all the functions presented when one logs in would require learning where certain functionality is physically located, certainly not a desirable feature for any interface. Another concern that was noted had to do with the mapping of menus at the top of the interface to the palettes in the main interface. At least one evaluator mentioned that an interface should have only one way of achieving some desired functionality. By having multiple ways of performing one operation, the interface can become cluttered and could potentially confuse users who assume that different menu options perform different operations.

The main strengths of the interface, according to the evaluators, were the static menu on the left side of the screen, the tabs which allow one to change modes and the alert bar at the top right of the screen. Evaluators liked the static menu as certain functionality is always available and therefore does not require searching through menus for the required options.

Discussion

The Web interface presented in this dissertation aims to improve on many of the usability and aesthetic problems from which the two interfaces discussed in this section suffer. A lot of the problems identified with the Gridport interface have to do with the lack of dynamic properties. The AJAX-based design approach which will be discussed in Chapter 5 aims to provide a solution to this problem by presenting the user with information in real-time. Another shortcoming of both interfaces is that they bombard the user with information. Once again, by making use of a dynamic AJAX-based approach, this problem will be alleviated by making use of information hiding techniques to display only relevant information, with the option of exposing more as needed.

4.2.3 Prototype

After the sample interfaces were shown to evaluators and comments on these interfaces were received, the first paper prototype of the final Web interface was presented to the evaluators. Comments on the layout and general functionality of the interface were generally positive and only minor problems were reported. The most significant of these problems was the way in which the resource-centric view was to be displayed. The aim of such a view is to allow the status of the Grid to be visible at all times, except when a user changes the "mode" of the interface from Grid view to Job view. A prototype resource-centric view is shown in Figure A.1 (see Appendix A). Many of the evaluators noted that a logical grouping of Grid resources would be more beneficial than a large scrollable list of machines with no logical structure. The concept of the Grid view, however, was positively accepted. Another significant problem noted was with the submission interface. This interface has many fields, which evaluators thought should have values filled in by default. In other words, having the system automatically detect the optimal set of values from the current state of the system would, according to the evaluators, enhance usability and save time on the part of the end-users. Evaluators also felt that the visual command line argument editor, see Figure A.6, should have drag functionality built in and not rely on clicking in order to move arguments up and down in the list. This editor allows users to specify an argument as a field, similar to a field in a database, by choosing a type and then stipulating how to dynamically define values for that particular type. This editor is discussed in more detail in Chapter 5.

Minor problems that were brought to light were related to ordering of items on the "Welcome" screen. According to the evaluators, job errors should be prominently displayed on the screen above all other information with the exception of news. It was also noted that having too many overlapping windows could result in a cluttered interface and lack of control or visibility of underlying windows. In order to alleviate this problem, the number of windows per operation can be reduced, thereby minimizing window overlap.

Apart from the problems mentioned above, a number of usability enhancing suggestions were provided. Some evaluators noted that the use of mock diagrams for certain key tasks would help provide a clearer understanding of what users are trying to achieve. This is especially the case with new users that have not used HPC resources before. Similarly, the use of graphics in the form of graphs in order to display certain key system statistics was suggested in order to help users make informed decisions about when to run jobs. Users could look at trends in such graphs in order to find times at which the system is relatively idle and then have their job launched at that time. Evaluators also noted that the use of graphical elements that could approximate the time to completion of a job would also be beneficial to users. In terms of submitted jobs, evaluators felt that it was important for users to be able to view jobs that were being run by all users on condition that the specifics of such jobs were not displayed. In order to achieve this, it was

suggested that the notion of groups be formed whereby users belonging to the same group were able to see one another's job details. In terms of the high-level interface itself, evaluators saw the need to have the interface change its appearance depending on which scheduler a user chose to submit jobs to. In order to achieve this, technologies such as CSS (Cascading Style Sheets) can be employed in order to change the look and feel of a page depending on the scheduler chosen.

A recurring theme throughout the interviews conducted was that of information hiding. Most of the evaluators were adamant that only commonly used options and functionality be displayed at any given time. Suggestions for the implementation of such a feature came in the form of basic and advanced modes, already discussed in Section 4.2.2, as well as the use of tree structures where only certain nodes are visible at a time.

4.3 Summary

This chapter has presented the prototype evaluation sessions, the feedback obtained from these sessions as well as user evaluation techniques employed during these sessions. Feedback concerning preferred interface layouts, suggestions concerning possible functionality as well as problems with the prototype that was built as part of phase 1 were presented. None of the problems mentioned were deemed to be serious, however. The results from the interview sessions indicate that the prototype built as part of phase 1 has a good, usable design. One of the more significant results obtained from this part of the design process was the importance of dynamic properties in Web applications. Many evaluators, when shown sample interfaces, noted that dynamic properties such as automatic updating of certain screen elements would go a long way to promote usability of such interfaces. Since this research has a strong focus on dynamic Web applications, these results were well received.

Chapter 5

Infrastructure & Design

5.1 Introduction

Up until this point, this thesis has covered the background concepts relating to this research and has provided an overview of the various techniques used to evaluate the design of the Grid tools proposed. Furthermore, the results from these prototype evaluations have been reported. The specifics of the Grid tools (referred to as the Web interface from this point forward) were not decided upon until after these prototype evaluations were conducted since it was uncertain if new features would be proposed by users.

This chapter will focus on the design of the Web interface, as well as a detailed overview of the supporting infrastructure. This chapter concludes with a discussion of techniques that have helped to improve the performance of the interface itself.

5.2 Infrastructure

A discussion of the Web interface has been split into two parts. This first part will outline the underlying server-side infrastructure upon which the interface components rely. These software tools and systems which make up the back-end of the Web interface have been split into a few categories, namely: test Grid, container, schedulers, database and directory structure. Each of these will now be discussed in turn.

5.2.1 Test Grid

Since this research assumes the presence of a Grid infrastructure, a test Grid was built at the beginning of this research project (see Figure 5.1). As can be seen from the figure, the test Grid consists of three clusters, one Grid server and a handful of Windows workstations. Furthermore, one of the clusters is located on a separate subnet and is owned by a different department.

The test Grid consists of a wide range of operating systems and architectures. The first cluster, "simba", is an Intel X86-based cluster with both the FreeBSD and Linux operating systems. The second cluster, "mkuttel", is similar to "simba", but consists of only Linux machines. Finally, the "lcc" cluster consists of both X86 32-bit and 64-bit machines, and runs the Linux operating system. As mentioned, a handful of Windows workstations were connected into the test Grid. These machines were added on a volunteer basis— in other words, they are only utilised if they have been idle for a sufficient amount of time. All these machines were connected using the Condor flocking mechanism thereby making the test Grid a Condor-based Grid. The LoadLeveler scheduler was, however, installed on the "mkuttel" cluster as well.

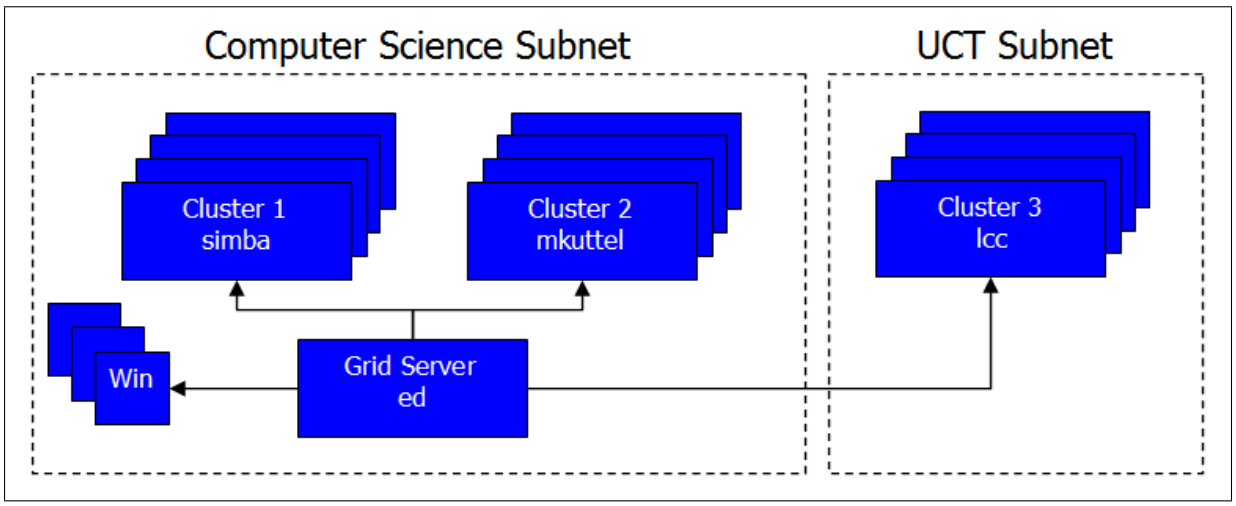

Figure 5.1: High-level Grid architecture with the Condor flocking system

Component	Value
CPU	Intel Core 2 E6300 @ 1.86 GHz
Memory	2.0 GB
Disk	320 GB
Network	Gigabit LAN
Operating System	Ubuntu Server 7.10

Table 5.1: Grid server machine specifications

The test Grid also includes a Grid server, as can be seen in the figure. The main role of the Grid server is to host the Web interface and provide a central point from where jobs are submitted. Results of Grid jobs are also therefore returned to this server on completion. Since all the software built during this project is installed on this server, the server specification can be found in Table 5.1. To summarise, the test Grid consisted of approximately 25 machines (or 60 cores) spanning two administrative domains as well as two independently administered networks (one managed by Computer Science and the other by the University networking department).

5.2.2 Toolkit & Container

Web browsers were initially designed to display static content and, by means of hyperlinks, link to pages containing similar content. As desktop applications have matured and become more usable over time, the Web too has moved to a more dynamic and interactive Web [Weinreich et al., 2008]. That said, browsers today are performing tasks that they were never engineered to perform. This makes developing for the Web particularly challenging, especially when one has to deal with problems such as cross-browser incompatabilites and the sheer complexity of Web APIs.

For the Web interface designed as part of this research, the ZeroKode (ZK) AJAX toolkit was used [Zkoss.org, 2008]. One utilises the Java programming language as the primary development language in order to write an application with the ZeroKode toolkit. A strength of this toolkit is that it automatically renders the browser-based JavaScript, so the developer only needs to know how to program in Java. This significantly decreases development time, however, the penalty is a slightly bloated system due to standard JavaScript libraries, with superfluous elements, being transmitted to the browser as part of the Web application. The benefits of using a toolkit such as ZK may outweigh these penalties, however. The number of predefined widgets, access

to a public help forum and adequate documentation flatten the learning curve associated with learning how to write an AJAX application somewhat, thereby abstracting the complexities of developing for the Web, and enable a prototype system to be developed reasonably quickly. A production system would usually make use of hand-coded AJAX calls so as to make the application as efficient as possible, however ZK was used for the prototypes produced in this research.

The ZK toolkit requires the use of the Apache Tomcat[1] servlet container. This container was chosen since it is a reference implementation of the servlet standard. The Tomcat container allows for the creation of WebDAV[2] folders on the server. WebDAV utilises HTTP to transfer data between a user's desktop and the Web server, usually without the need for third-party software since most modern operating systems have WebDAV capabilities built into the standard network management software. This is used as the primary mechanism for transferring large numbers of input files to the server. This will be discussed further in Section 5.3.3.

5.2.3 Schedulers

The primary function of the Web interface is to serve as a Grid front-end. Although good usability and a lightweight design are important research outcomes, the scheduling capability forms the foundation of the system. That said, the system makes use of two scheduling systems, namely Condor and LoadLeveler (see Section 2.2.5). Instances of each of these schedulers are installed on the Grid server from where jobs are propagated to the Grid by means of the Condor flocking and LoadLeveler multi-cluster systems. These scheduler instances can be run on a separate machine— however, for this research only one dedicated Grid server was available. Having the schedulers installed on the same machine as the interface has the benefit of access to the local database, thereby eliminating the need for remote database connections. Due to the database-centric nature of this system, this is a significant advantage.

As mentioned in the previous paragraph, the test Grid had two scheduler implementations installed, namely— Condor and LoadLeveler. Referring back to Section 1.2, one of the objectives of the interface is to ensure that it is extensible. In order to achieve this objective, two converters were written in order to translate the interface PSDL into a format understood by each scheduling system. Furthermore, the convertors each implement a set of methods which the various interface components call in order to populate the relevant on-screen panels. Figure 5.4 presents a graphical view of how each of the converters tie in with the system as a whole. The figure shows how the various components (submit, query and status) interact with the different converters based on the scheduler chosen by the user. The convertors not only enable different scheduling systems to be added over time, but will also allow for more complex interface features to be added in time. Load-balancing across different scheduling systems is one such example and is discussed further in Section 8.3.

From the results of the initial interview sessions held with scientists, discussed in Section 4.2.1, the need for a way of displaying the status of the Grid was deemed important. In order to accomplish this, it was necessary to build scripts that would gather status information and store it in a usable way. These scripts, custom-written for each scheduler, ensure that the multi-scheduler design approach is realised by separating scheduler logic from interface logic. Use of these scripts from the perspective of the Web interface will be discussed in Section 5.3.

[1] http://tomcat.apache.org/
[2] http://www.ietf.org/rfc/rfc4918.txt

5.2.4 Database

All Grid status information as well as information on jobs, machines and users are stored in the database. Data is retrieved from the database on each rendering of the interface as well as each time a user calls upon a data-bearing element to be refreshed. Furthermore, the WebDAV file transfer component relies on the Tomcat database for user authentication to the WebDAV shares.

Due to the amount of data being generated in terms of Grid status information as well as job information, the performance of the interface at the beginning of the development process degraded as the database grew in size. For this reason, a number of table indices were created in order to speed up these common operations. Other techniques such as periodic data exports of data no longer needed by the interface were also considered, but not implemented.

5.2.5 Directory Structure

As already mentioned, WebDAV shares were created to allow users to bulk-upload files to the Grid server. In order to accomplish this, a directory on the server is created for each user. Each user directory contains three folders in which data from various steps in the Grid job submission process are stored. The first such directory is the `PROJECTS` directory which contains data pertaining to each "project" or job the user wishes to submit to the Grid. All input files, binaries and any file needed by the job should be located in an aptly named project directory within the `PROJECTS` folder.

The second folder is the `JOB_TEMPLATES` folder. When a job is created using the Web interface, an XML template file coded in Parameter Sweep Distribution Language (PSDL) is stored in this directory. PSDL is a customized version of JSDL, discussed in Section 2.4.2, that overcomes the shortcomings of JSDL when specifying parameter sweep applications. Since JSDL does not have a mechanism for supporting multiple consecutive sets of arguments pertaining to each run of a parameter sweep Grid application, the current version of JSDL was modified to support such a mechanism— this is known as the PSDL. The PSDL-based template is created so that a user can run a job multiple times without having to perform the job creation process each time it is to be run on the Grid.

The third folder present in each user directory is the `COMPLETED_JOBS` folder. As the name suggests, all output generated by a Grid job is written to a subfolder within this directory on completion of the job.

5.3 Design

After the initial interviews and paper prototype evaluations (see Chapter 4) were conducted, the final Web interface design was finalised. This design incorporates many of the enhancements and suggestions brought to light by the potential users of such a system as well as some of the initial design features. This section will present an overview of the design of the Web interface as well as a detailed discussion on each of the core components of which the system is comprised. Furthermore, the way in which these components interact with the infrastructure outlined in the previous section will be discussed where relevant.

5.3.1 Design overview

Before a discussion on the detailed design of the interface is entered into, an overview of the layout of the Web interface will be presented (see Figure 5.2). As can be seen from the figure, the interface consists of four main panes, namely the information header at the top of the screen, the

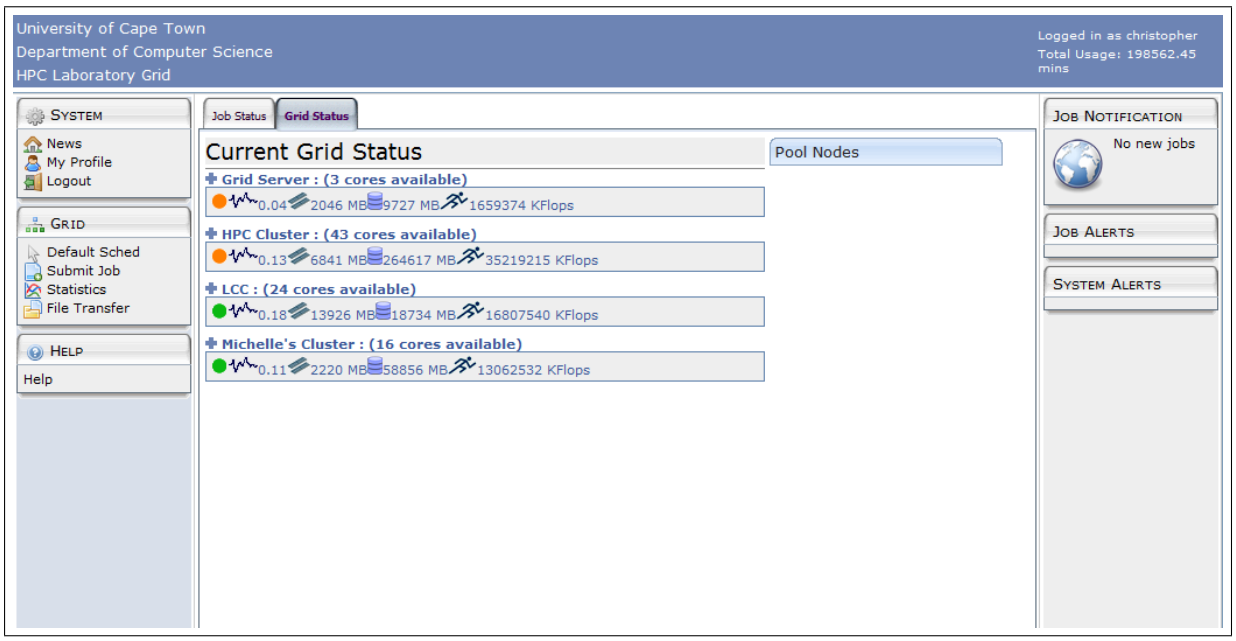

Figure 5.2: Grid interface layout

menu pane on the left-hand side, the status pane in the center and the notification pane on the right-hand side of the screen. Other than the menu pane, which will be discussed in the section to follow, the status pane is one of the most important parts of the interface. This tabbed pane provides the user with access to Grid status information as well as job status information, and depending on the operation the user is performing, one of the these two panes is always visible.

All other interface components, excluding the job and Grid status components visible in the status pane, are activated using the menu pane on the left.

5.3.2 Design Considerations

Interface Visualisation

The most distinct design feature of the Grid interface is the resource-centric view of the system. Traditional websites have assumed what can be best described as a task-centric approach to interface design. Since AJAX is a relatively new design methodology, concepts such as a task-oriented approach to design do not really exist since this development paradigm has always been the norm. Popular websites such as Amazon.com and the South African variant, Kalahari.net, make use of such an approach. This approach is characterised by the different "modes" in which the site can find itself. In browsing "mode", the user is able to browse though a set of products and is able to add these products to his/her basket. In payment "mode", the user is taken to a different part of the system where product information is no longer displayed. In these cases, however, the metaphor makes sense, since real world shopping consists of two "tasks": browsing and paying. This approach focuses the attention of the user on the task on which he/she is currently busy.

In contrast to the task-centric view, what can best be described as a resource-centric view keeps certain information visible at all times, even if a user is in another "mode". The concept of the Desktop on a PC, for example, is an example of such an approach. Even though a user is busy writing an email message, for example, it is still possible to view the status of the processor or see an incoming instant message. For the Grid interface, it was decided to make use of this resource-centric design approach since a Grid environment is a dynamic system. The status of

the Grid is therefore visible at all times, except when a user changes from Grid status mode to job status mode (see Figure 5.2). As can be seen from the figure, a tab box houses the Grid and job status components. Since screen real-estate is limited, it was decided to make use of the full display area for the job status window as it displays a large amount of data. Furthermore, since a user is unlikely to be using the information within the status window in querying a job, the Grid status information needs not be shown.

Menu Pane

Another prominent feature of the Web interface is the static menu pane situated on the left of the interface (see Figure 5.2). From the sample interfaces shown to test subjects during the inital prototype evaluations, many pointed out that the use of such a menu provided a consistent way to present the options available in the interface. The idea for this menu was drawn from the "Control Panel" present in operating systems, specifically Microsoft Windows. Such a menu makes it clear what functionality is available to the system with a simple and intuitive layout, and does so in a consistent manner. During the prototype evaluations, users felt that it was better to have only one mechanism to perform a certain operation, a function which such a menu fulfils. Furthermore, the use of overlapping windows and information display techniques inside the browser (which will be discussed in the sections to follow) that mimic that of the Desktop, were used to further promote a resource-centric view of the Grid.

The techniques provided here formed the basis of the design of the Web interface. However, many other important design considerations that are best kept in context of their respective components will be discussed in the sections to follow.

5.3.3 Interface Components

The functionality present in the Web interface has been split up into a few main components. In order to better illustrate how each component ties into a Grid job submission and monitoring workflow, each component has been mapped in the flowchart shown in Figure 5.3. As can be seen from the figure, each component operates independently, except for the job submission component. This component, which consists of a wizard, utilises functionality present in the file browser for its operation.

An overview of each of the components in terms of design and functionality as well as the way in which they communicate with the server will be presented in this section. Since the Grid interface has been built to support parameter sweep applications, the ways in which the interface has been built to suit such an application style will be highlighted. Furthermore, Figure 5.4 shows the high-level system design and can be used as a reference as each component is discussed throughout the course of this chapter.

Grid Status

As already mentioned, the Grid status component displays the current status of the Grid. When the interface is first loaded, this status page is the first to be displayed. To generate this data, Python scripts were written which wrap around low-level command line utilities bundled with each scheduler. The Condor-specific script retrieves an XML document containing the status information of each pool on which the script is run. The data in the XML document is then parsed and used to populate a database with the relevant fields (see Figure 5.4). The LoadLeveler script operates in a similar way but an unstructured flat file is generated instead of XML as in the case of Condor. These scripts, known as the Status Daemons (SD) (see Figure 5.4), are executed every five minutes, thereby providing snapshots of the status of the Grid at five minute intervals. Furthermore, since the way in which scheduling software is written differs

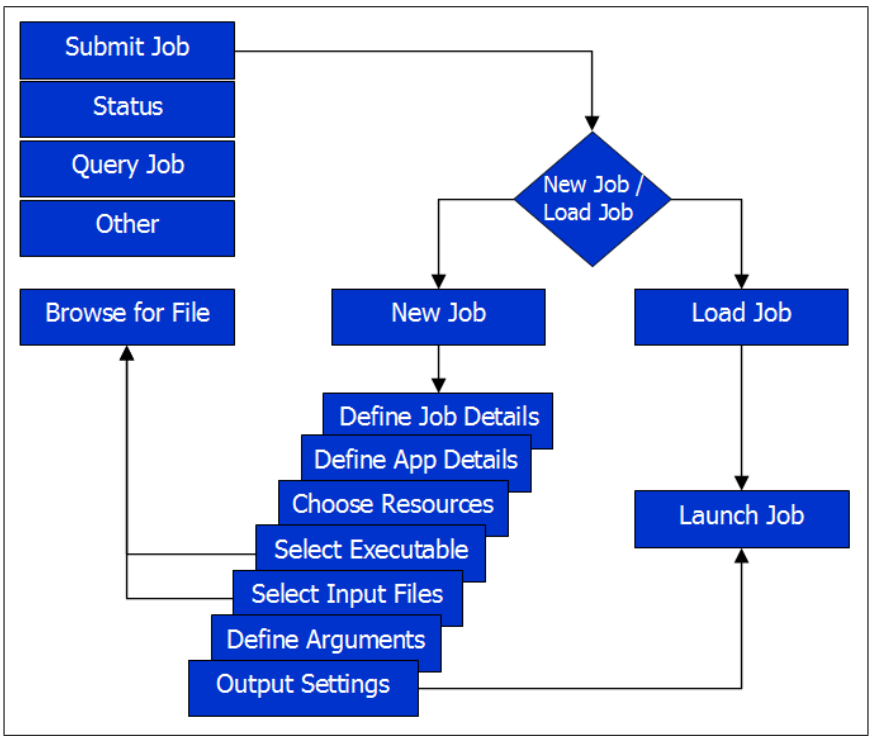

Figure 5.3: Interface component flowchart

greatly, scheduler-specific scripts for job management were created. These scripts handle the way in which jobs are created and deleted and are loaded dynamically by the Web interface when scheduler-specific methods are called.

Figure 5.5 shows a typical Grid status snapshot in both collapsed and expanded view. By default, the detailed information will not be displayed — however, for purposes of illustration, the status of some Grid pools are shown in their expanded view. For brevity, the LoadLeveler pools have been left out of this discussion as their status information is displayed in a similar fashion. From the figure, four pools are visible. The first, "Grid Server", displays information on the status of the Grid server itself. Since both the LoadLeveler and Condor scheduling systems were installed on the Grid server, the server is considered to be a separate pool. There also are three further pools visible in the figure, representing the three Condor-enabled clusters mentioned in Section 5.2.3. The figure also shows status information pertaining to each pool. As mentioned already, this data is extracted by the Status Daemons and inserted into the database from where it is read by the status component. In order to explain what all the values mean, the information displayed for the second pool - the "HPC CS Cluster" - will be elaborated upon.

The first important value to note in Figure 5.5 is the number of cores available. This value represents the number of virtual CPUs that are currently able to accept or process jobs. The concept of a virtual CPU is used to represent a single core in a multi-core processor. In other words, if a machine has a quad-core processor, the scheduling software will partition the CPU into four virtual CPUs and schedule work to each core independently. This strategy works well since non-threaded applications typically do not maximise use of a multi-core CPU. The next set of values, along with their respective icons, give an averaged overview of the entire pool. The values shown here (from left to right) represent average pool load, total pool memory, total pool disk space and total pool processing capacity (measured in KFlops). Each pool also has a coloured icon to the left of the set of average metrics. This icon is used to quickly identify the state of machines within a pool. If a pool and all the execute nodes of which it is comprised are

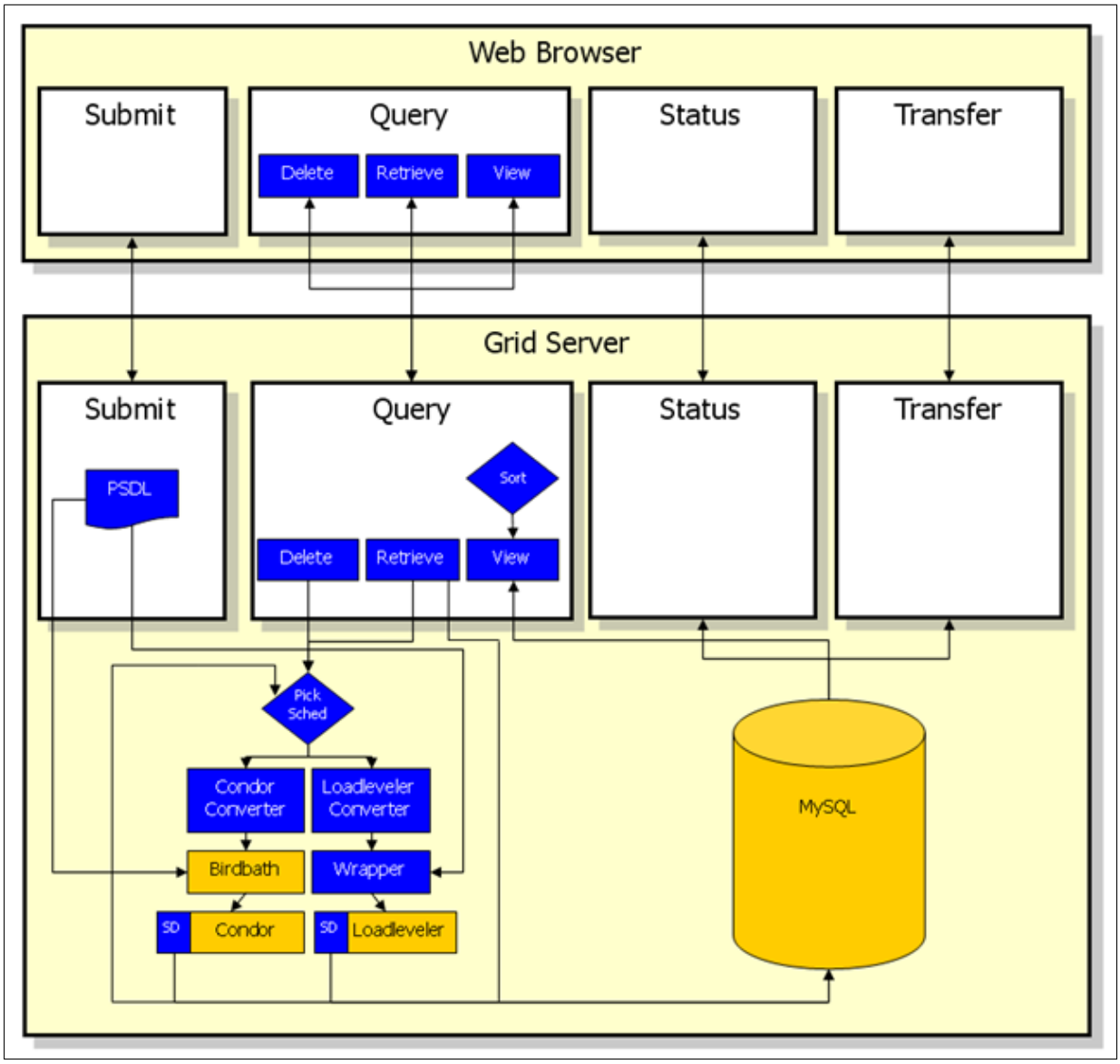

Figure 5.4: High-level system architecture. Components coloured in yellow represent existing systems, those coloured in purple represent custom-built components.

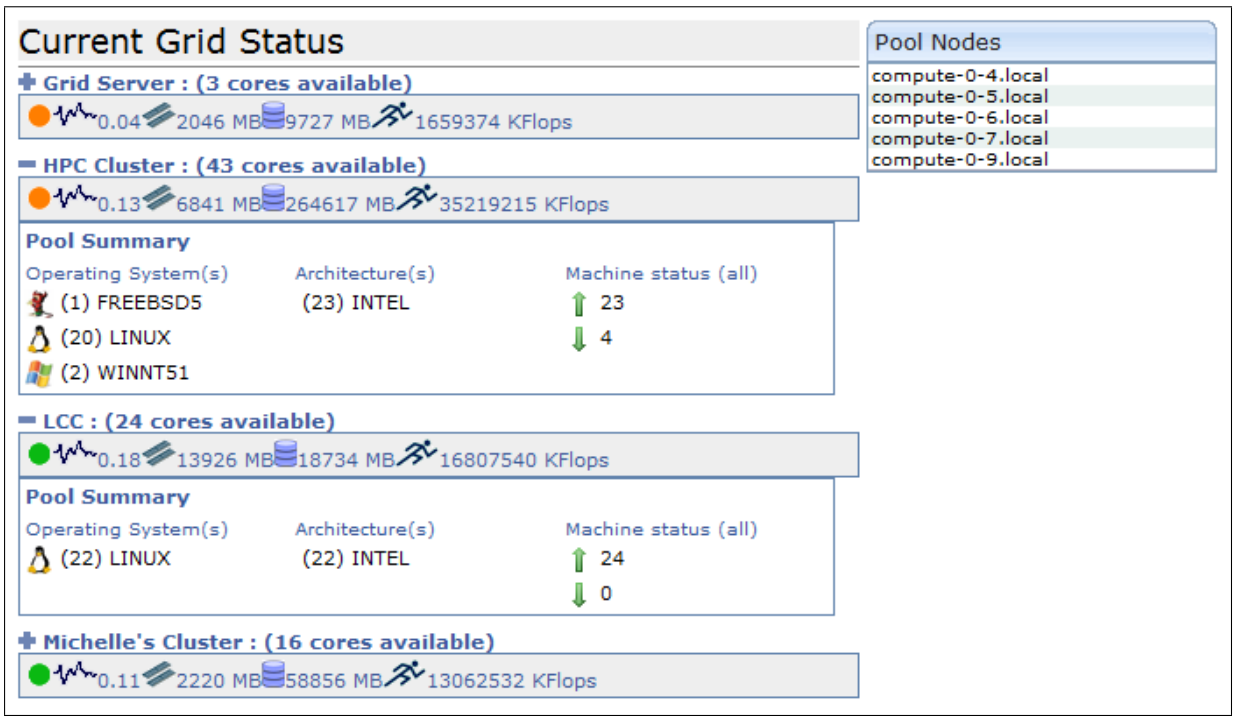

Figure 5.5: Grid status component

fully operational, this icon is coloured green. If some of the nodes are down or not running the scheduling software, the icon is coloured orange. If all the nodes, including the head node of the pool are down, this icon is coloured red. The information overview bar as well as the number of cores available therefore make up the default view.

As already mentioned, the status is first displayed in the default view. This is a collapsed view displaying minimal information on the status of a pool and includes the icons mentioned. There is, however, an advanced view or expanded view which shows more detailed information on the status of each pool's nodes. This summary provides a breakdown of the architectures and operating systems present within the pool as well an indication as to the number of machines that are up and running as well as the number that are down. The operating system and architecture information is only applicable to the machines that are willing to accept jobs on the Grid. Therefore nodes that are down or unwilling to accept jobs will not be included in this summary. The final element to the status display is the "Pool Nodes" box. Each time a particular pool is expanded, this box is updated with the names of the machines in the particular pool.

The Grid status component is one of the most important parts of the Grid interface. Without this information, users would simply be submitting jobs to the Grid without prior knowledge of the resources available. Since executables are platform- and operating system-dependent, this information is useful to users developing applications that are to be run on the Grid. Furthermore, since the interface can be configured to use many different schedulers, this information becomes useful in determining which scheduler will be best suited for a particular application, pgiven the current state of the Grid.

Job Specification

The second major interface component is the Job Specification component. The job submission process is split into two steps, namely: job specification and job launching. This section will cover the job specification process. This process takes the form of a wizard. The wizard (see

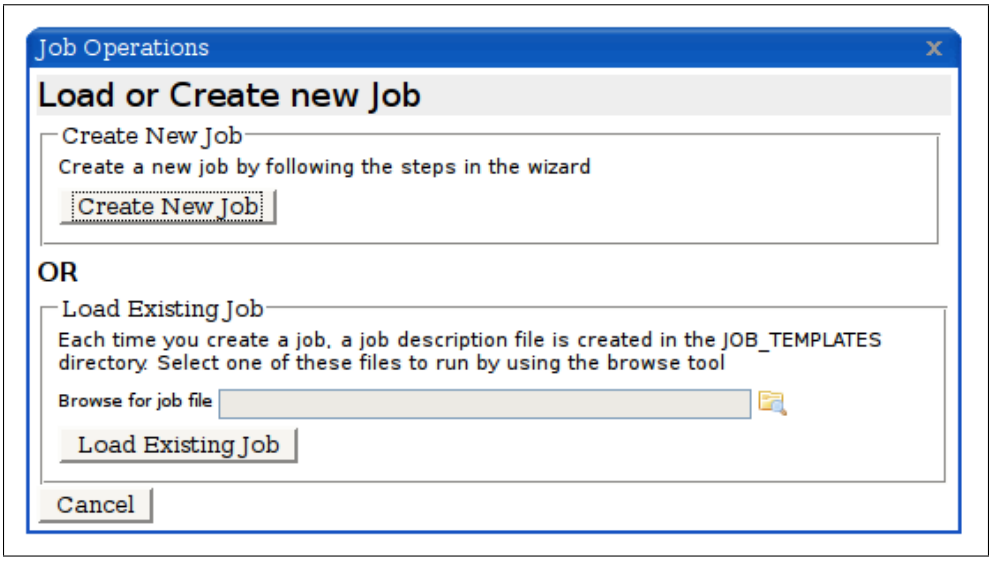

Figure 5.6: Load or create new job window

Figure 5.6), allows a user to choose whether to create a new job or load a PSDL template from file, the origins of which will be discussed later. For now it will be assumed that the user chooses to create a new job specification. The user is then shown a wizard which will guide him/her through the rest of the process. The wizard consists of seven screens (see Figure 5.3) which include the following steps:

1. Job specific information (job name, description, etc.)
2. Application specific information
3. Resource filtering (architectures, OSes, etc.)
4. Executable selection
5. Input file selection
6. Input argument enumeration
7. Output-specific settings

Steps 1 & 2 : Job and Application Specific Details

The first two screens, namely job- and application-specific information, prompt the user for a name and description of a job as well as an application version number. These details distinguish one job from another and allow specific directories, which will house the relevant files, to be created on the server.

Step 3 : Resource Filtering

The third screen, the resource filtering screen (see Figure 5.7), allows a user to choose the systems his/her job is to be executed upon. Although scheduling systems have a record of the resources available on the Grid, they have no way to match such resources to applications. To use such scheduling systems, it has therefore traditionally been up to the user to manually specify which resources he/she would like to utilise. This process requires consulting a scheduler reference manual in order to determine which resources need to be specified in a submit file that is passed along with the application to the scheduler. Furthermore, low-level scheduler

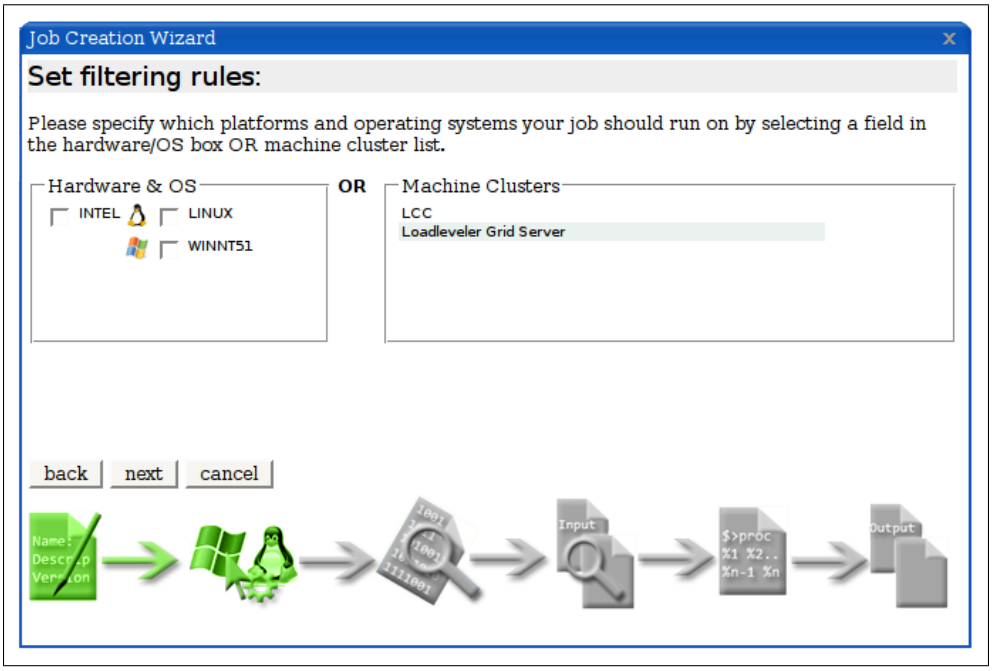

Figure 5.7: Resource Filtering Wizard

utilities need to be consulted in order to determine which resources are available to the Grid. This information is then manually specified in a submit file which outlines exactly how the application will be submitted to the scheduler, what resources to use, what input files the application takes, et cetera. The aim of this step in the wizard is therefore to automate this process.

The resource filtering screen provides the user with two ways in which to choose the resources his/her job requires. A user can either directly choose the architectures and operating systems from the "Hardware & OS" list, or choose the pool(s) that the job is to be run on from the "Machine Clusters" list. Each of these selections is independent as the user may only select from either of these two lists. For example, if a user chooses a pool on which to run Grid jobs, the manual operating system and architecture selection option will be disabled. Another feature of the wizard is the way in which erroneous filtering permutations are disallowed. As a user selects certain options from either of these lists, invalid combinations are automatically removed. For example, if a user chooses "WINNT51" from the "Hardware & OS" list, the "LCC" option on the "Machine Clusters" list will be removed since this pool does not contain any Windows-enabled machines. This mechanism prevents jobs that will never run from being submitted to the Grid.

Step 4 : Executable Selection

Once the user has selected the resources on which his/her Grid jobs will run, the next step in the wizard prompts the user for the binaries or executables which form the basis of the job. These binaries or executables can be C++ programs or simple bash scripts, for example. As can be seen from Figure 5.8, based on the selection made on the filtering screen, the interface will prompt the user to browse for executables located on the server. In order to do so, a custom-built AJAX-based file browser is used (see Figure 5.9). Since there is no standard way of viewing files residing on the server from within the browser, this component had to be written from scratch. In order to navigate to the directory containing the executable, a user simply clicks on a folder icon next to the respective folder that the executable resides in and the contents of the directory will be dynamically listed. Once the executable has been found, a user can click on the name of the file in order to select it.

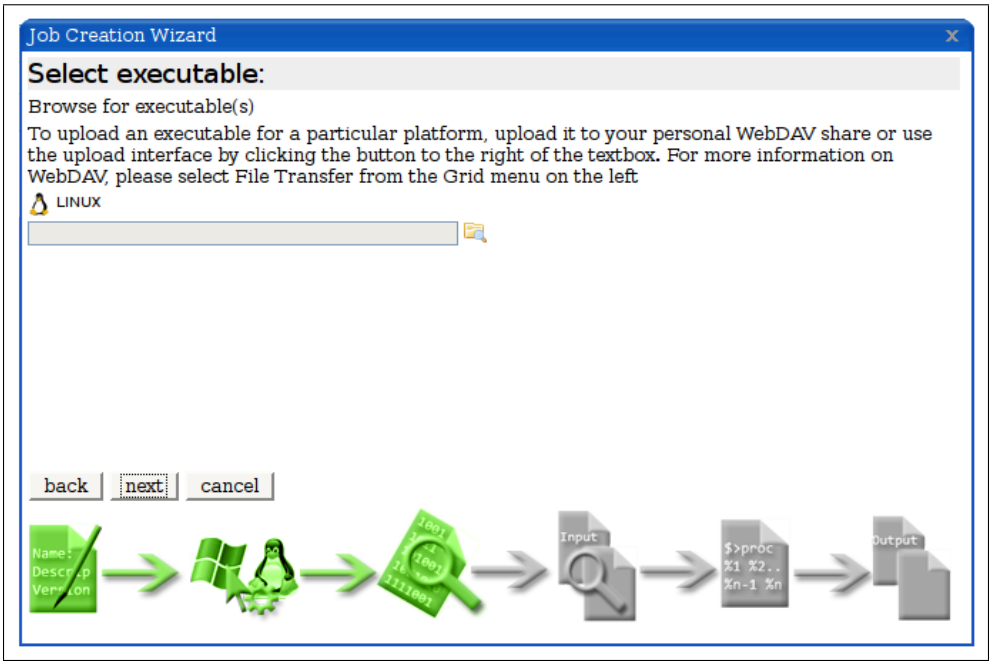

Figure 5.8: Executable Selection Wizard

Step 5 : Input File Selection

Once the executables have been selected, the next step in the wizard prompts the user for a set of input files. Since the majority of Grid jobs are data-based computationally intensive jobs, the reliance on input data is a very important aspect of such jobs. In the context of a parameter sweep application, it is most often the case that the same application will be executed on all Grid nodes; however, the applications will operate on different input files, thus generating different output. This is typical of the SIMD approach of many Grid applications.

The process of selecting sets of input files is similar to that of the executable selection, however, the main difference is that instead of selecting a file, the user is expected to select a directory of files (see Figure 5.10). Unlike the executable selection wizard, however, a user is able to specify multiple independent sets of input files at this step. The reason for this has to do with the adaptation of the interface to cater specifically for parameter sweep applications. For example, if a job taking a single input file as an argument is to be run many times over, each time with a different input file, one would specify a directory of such input files at this step. The interface then maps an input file from this directory to an individual run for each file in the directory. Furthermore, if this job were to take two different input files as arguments (as opposed to one in the previous example), a user can simply click on the "plus" icon to add another set of files. An input file from each directory will then be mapped to a single run and so on.

Step 6 : Argument Enumeration

Now that the user has given his/her job a name, defined the resources on which the job is to be executed and selected the executables and input files, the definition of the application parameters can proceed (see Figure 5.11).

In order to illustrate the process of adding arguments to an application, an example will be used. For some application, assume that the application binary takes as input an integer value

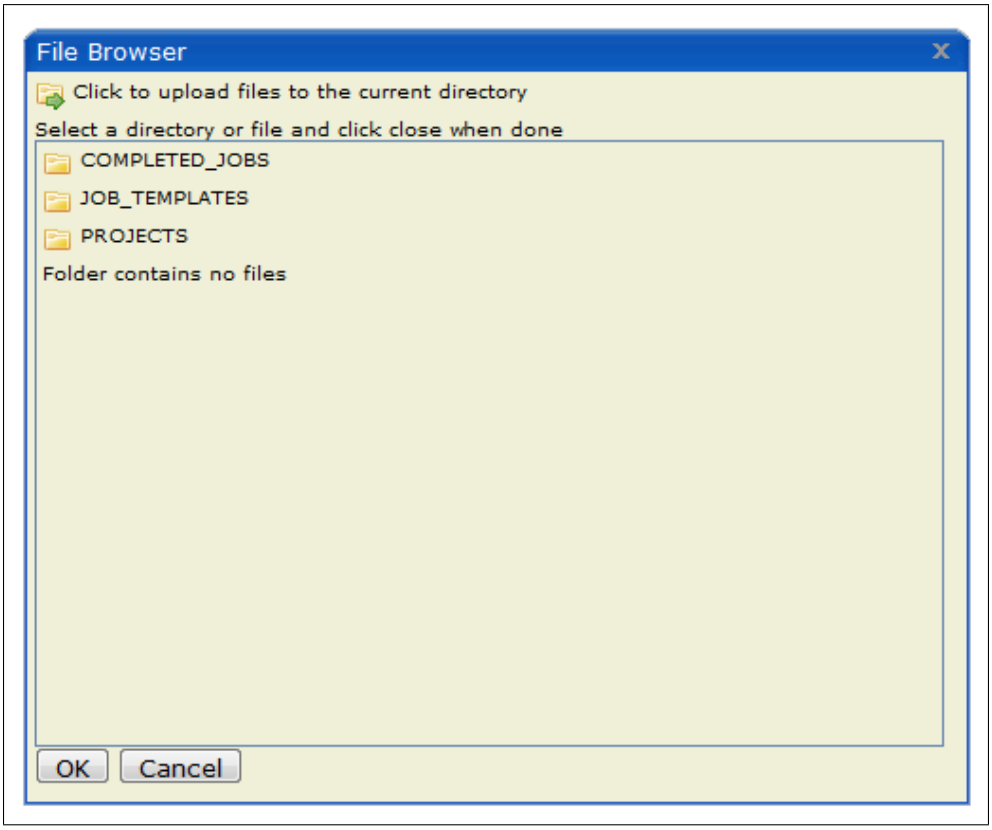

Figure 5.9: AJAX File Browser

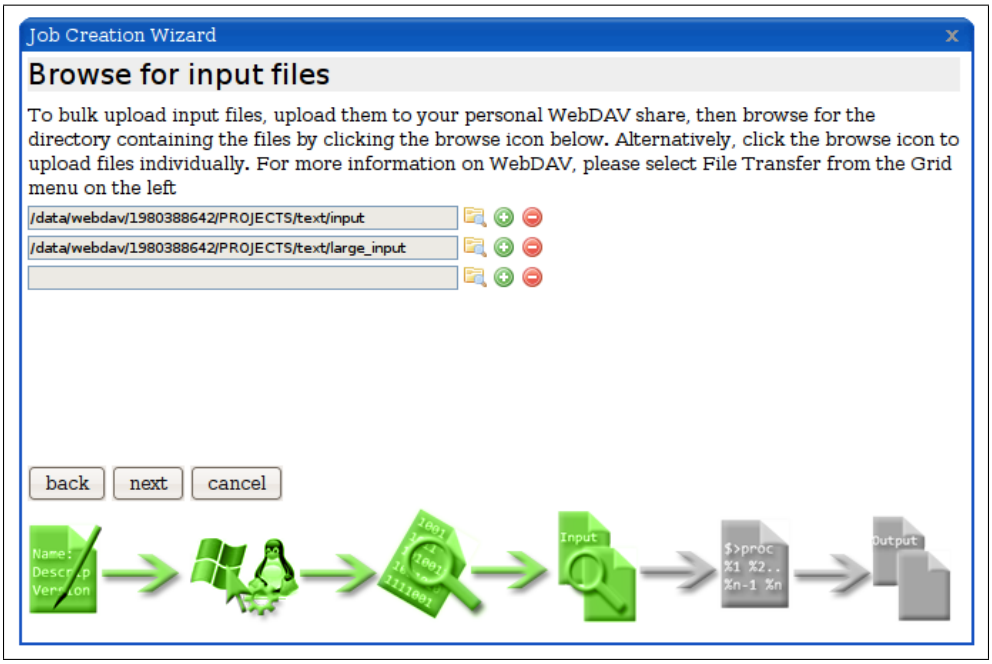

Figure 5.10: Input File Selection Wizard

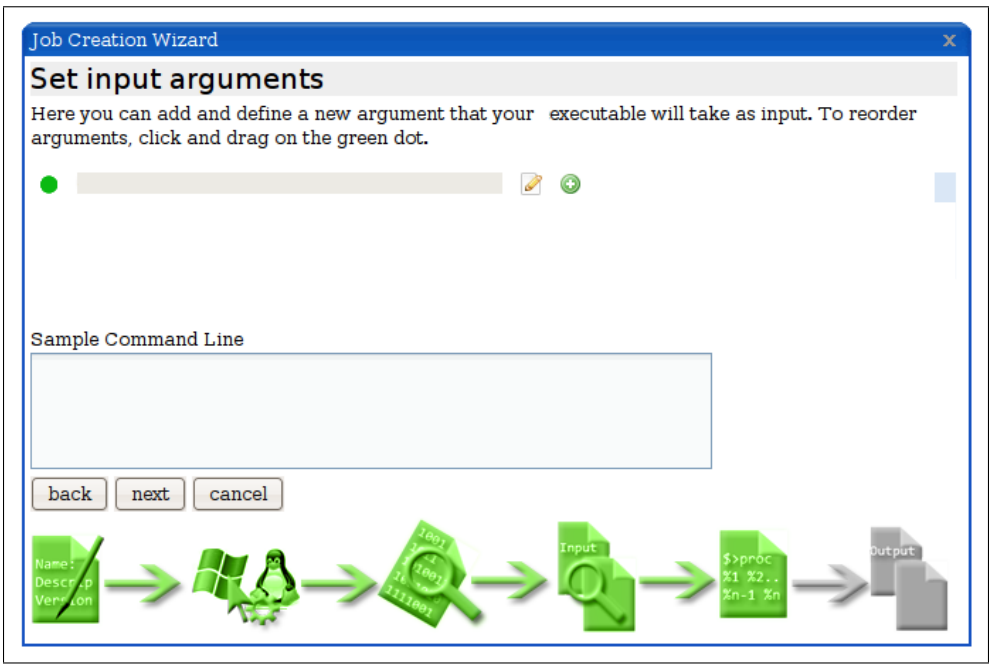

Figure 5.11: Argument Enumeration Wizard

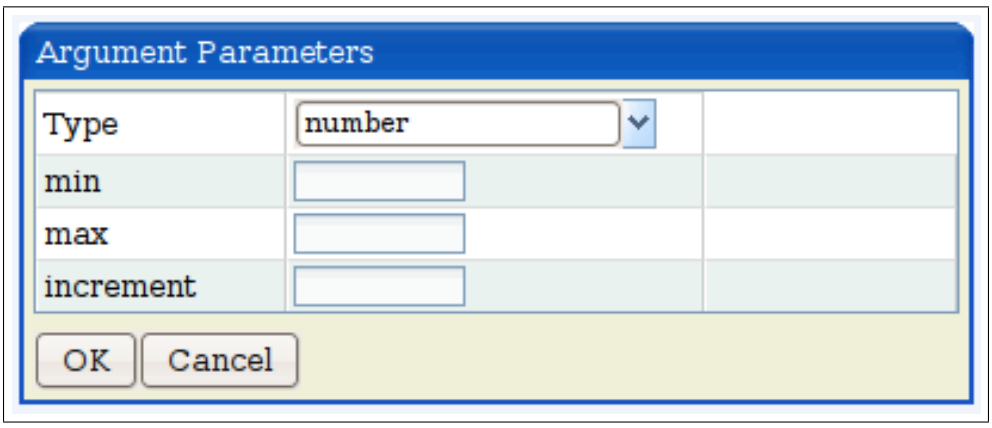

Figure 5.12: Number Type Specification Wizard

preceded by a flag (-i), an input file from a set of files as well as a static file (a simple text file) containing some configuration data. A few runs of such an application would therefore look as follows:

- executable -i 1 inputFile1 data.conf
- executable -i 2 inputFile2 data.conf
- executable -i 3 inputFile3 data.conf

In order to specify such a dynamic set of changing values, the wizard provides four argument types. The first type is the number type— a value that is incremented for each run of the application. The example above illustrates the use of the number type with an increment of one for each run, however it is possible to change the starting value, the end value and the incremental step value of the integer when setting up the argument (see Figure 5.12). The next type, the flag type, is a static type. A flag is merely repeated verbatim for each run of the application, as can be seen from the "-i" in the example above. There is one other static type, the single-file type, which similarly specifies the same file for each run of the application. In the example above,

"data.conf" is of the type single-file. The final argument type is the multiple-file type. This type, like the number type, allows a user to dynamically allocate a file from the directories of input files selected in the previous step. In the above example, "inputFile1" would be the first file in some directory chosen by the user and therefore would be allocated to the first run of the application.

Another important feature of this wizard page is the sample command-line window. This window displays the result of the argument enumeration procedure. The content of this window is updated dynamically as the user adds more arguments in order to display each run of a parameter sweep application similar to the example shown above. To add additional arguments, a user selects the "plus" icon from the area to the right of the first component. Since the ordering of arguments is important, the wizard also provides drag-and-drop functionality that allows arguments to be re-arranged easily. In order to implement this functionality, this wizard relies heavily on AJAX. Dynamic addition of arguments, removal of arguments, updating of the sample command line as well as the drag and drop interface all rely on explicit AJAX calls in order to manipulate the DOM in the browser.

Step 7 : Output Settings

The final step in the job specification process is the selection of output parameters. It is often the case that Grid jobs fail on their first submission to the Grid. In such cases it is necessary to determine exactly what caused the job to fail. Having a job output a log file can therefore be a source of valuable information in such cases. The wizard therefore allows a user to choose whether such log files are produced at run-time. Finally, depending on the type of job, the naming of the output files can be important. The wizard therefore allows the user to select whether the output file names are generated using the process ID of the Grid job or by taking on the name of the input file.

Once all seven steps have been completed by the user, the job specification is complete. At this point, the entire specification is converted to PSDL— as alluded to at the beginning of this section. The PSDL XML document is then written to the user's `JOB_TEMPLATES` directory. The presence of the PSDL file allows the user to load a pre-existing job into the browser using the job submission wizard (see Figure 5.6).

Job Launching

As mentioned at the beginning of the previous section, the job submission process has been split into two steps. The first step, job specification, was discussed in the previous section. Job launching is the second step in the process. Once a user has completed the specification of a new job or loaded in a job from file, the job is ready to be launched. As can be seen from Figure 5.13, a user is notified by a flashing green icon on the right-hand side of the interface. In this case, a job called "indexer" is ready to be launched. When this icon is clicked, the process of submitting the job to the Grid begins.

Once a user has launched a job, it may take a while for the job to be submitted. Since the time taken to submit a job is dependent on the size of the job, the submission process can be time consuming for jobs where large amounts of data in the form of input file and binaries must be submitted. As a job is submitting, an AJAX-based progress indicator is shown at the top of the screen. Since large jobs that take a long time to submit would make the interface seem as though it is "hanging", the progress bar is an important feature of the submission system.

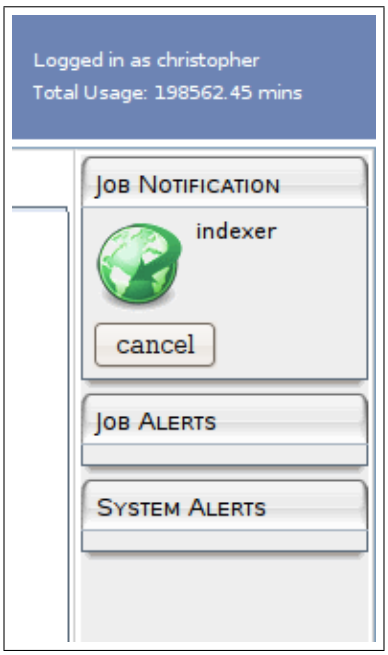

Figure 5.13: Job Notification Pane

Job Querying

The third major interface component is the job querying component (see Figure 5.14). This component allows a user to view the status of jobs submitted to the Grid. When a job is submitted, an entry for each run of the parameter sweep application is written to the database. When the querying window is loaded, it is this data that is imported into the interface. Each sub-job is then displayed along with the date and time the sub-job was submitted, the run time (only filled in when the job completes), the sub-job size (only filled in when the job completes), the status of the sub-job, the executable name and an option to remove the sub-job.

The query window gives the user a few options. Firstly, if a job has been "held" due to some error, the user is able to delete the entire job by making use of the job deletion feature at the top of the window. Furthermore, if only one of the sub-jobs is in error, the user is able to delete such a sub-job by selecting the checkbox of the appropriate sub-job and clicking the "Apply Changes / Refresh" button. The interface also provides a way for users to display only certain jobs by allowing for the selection of either a date range or a list of unfinished jobs (or both) from the sorting criteria area at the top of the window. Finally, when a job completes, the output files can be retrieved and moved to a data directory on the server by clicking on the "Retrieve Data" button.

File Transfer

Parameter sweep applications typically rely on input files as a means of work distribution. Such applications also usually consist of many hundreds or thousands of runs. A typical Grid application can therefore consist of many gigabytes of data files. The main reason for the reliance on WebDAV, as opposed to a Web interface for uploading, is that use of a Web interface is not scalable. If one considers a trivial parameter sweep application consisting of only 50 sub-jobs, then the process of manually uploading 50 files, excluding binaries and libraries, becomes a daunting task. The interface could have been designed to accept zip files for example, however, this would provide less flexibility and would make file management using the Web interface cumbersome. Nevertheless, the core interface also has file upload capabilities built into an AJAX file browser

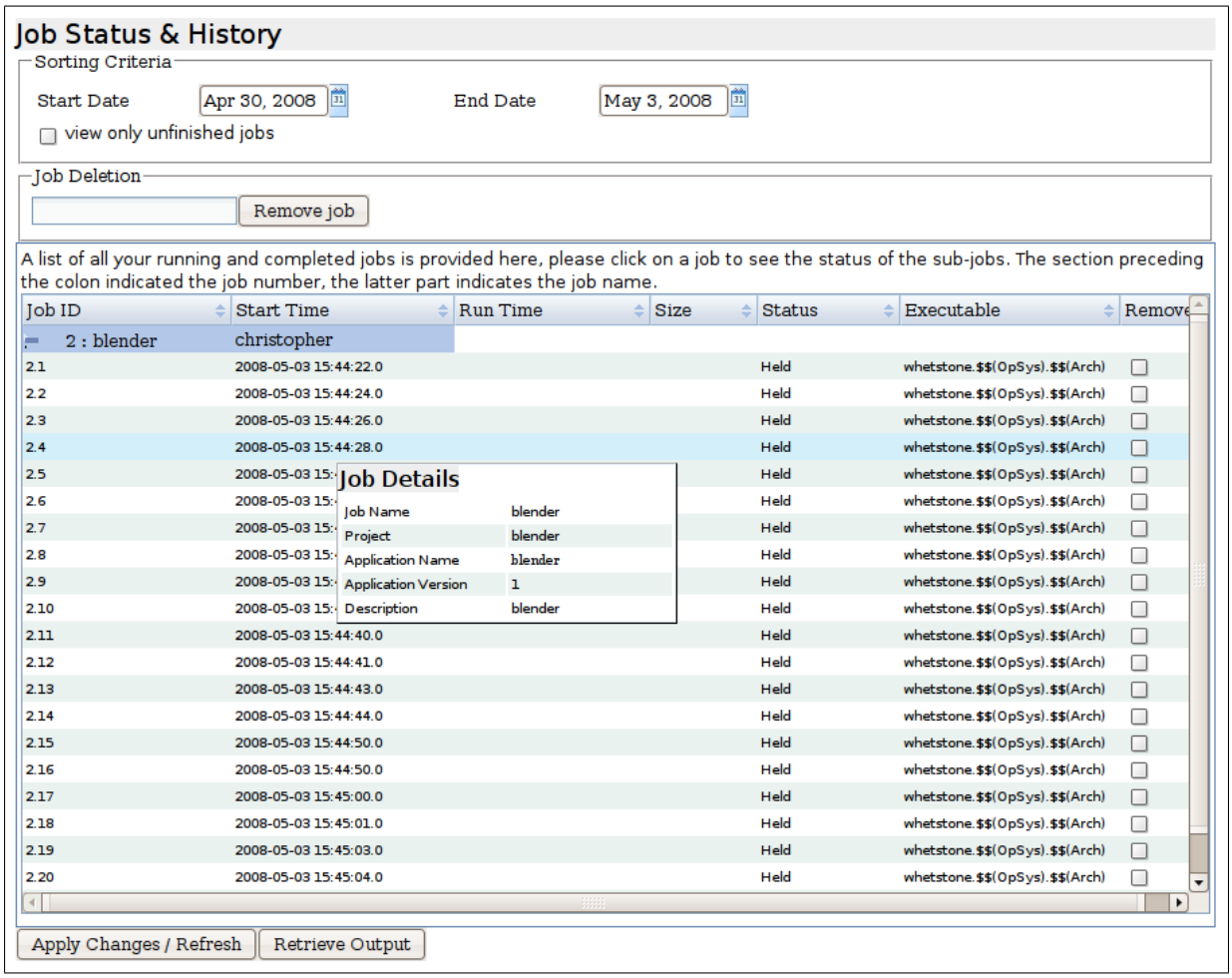

Figure 5.14: Job Status Window

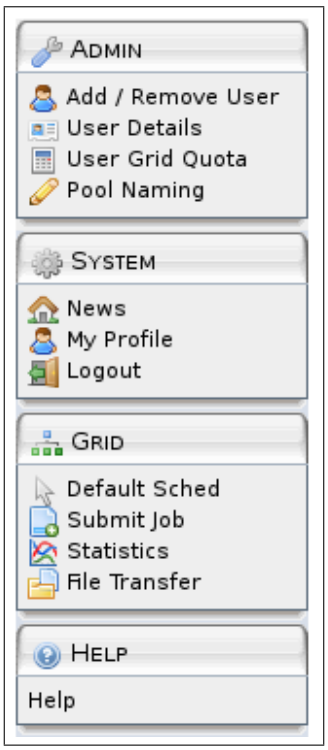

Figure 5.15: Admin Component

for cases when users wish to upload or overwrite single files.

The WebDAV file transfer component is tied into the core of the interface and therefore uses the same username and password as the main interface. Furthermore, the creation of a WebDAV share is an automated process. When users are added to the system using the administration system, discussed in the section to follow, the WebDAV share is automatically generated and is symbolically linked to the main data partition on the server. A unique name for each share is generated based on UNIX date functionality, thereby assuring that no two folders are assigned the same value.

Miscellaneous Components

So far, only the major interface components have been mentioned. There are, however, two other components that are worthy of mentioning— the admin interface and the statistics window. The admin interface, as the name implies, provides an administrator with essential tools for ensuring the correct operation of the interface as well as performing administrative tasks (see Figure 5.15). These tasks include the addition and removal of users, additon of user information as well as user quota limits and removing jobs that have exited in error (via a job query window with administrator permissions). The interface is accessed via a special system login which changes the layout of the interface by adding an admin panel to the left menu pane.

One of the features of the admin interface is the ability to allocate Grid time to each user. This prevents some users from "hogging" the Grid and therefore ensures a fair and equitable distribution of resources. The full quota subsystem was not implemented due to time constraints but the admin component that allows for the addition and resetting of a user's quota was implemented as part of the admin interface. Finally, the admin interface provides the administrator with a means of naming newly found pools on the Grid. Since new pools need to be given a name in order for the interface to function correctly, the admin interface allows the administrator to

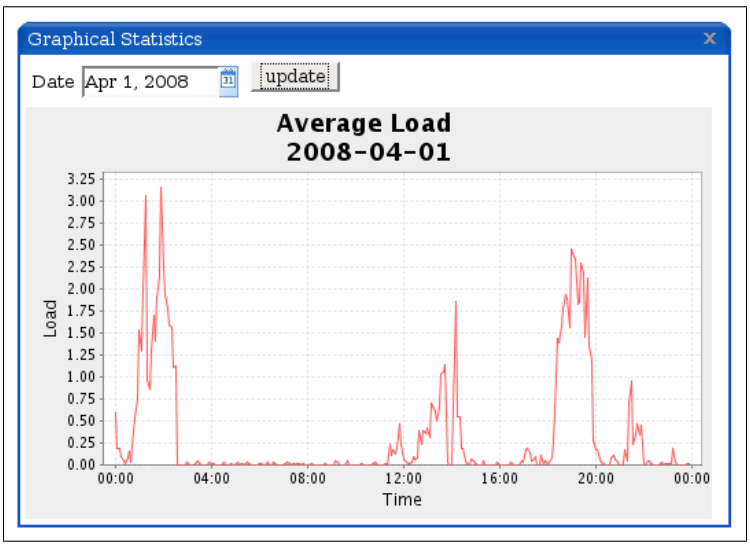

Figure 5.16: Statistics Component

set the name of any new pool found on the Grid.

The second component is the statistics component. It is often the case that users wish to submit jobs to the Grid in periods of low demand, for various reasons. Users could decide to submit jobs in off-peak times if there is no hurry for the results, thereby allowing other users the benefit of the Grid during peak times. Users can also submit in off-peak times to realise the full potential of the Grid, thereby generating results quicker than if a queue exists. In order to determine when off-peak periods are, the statistics component is made up of a graph which displays the average system load of the entire Grid for a user specified date (see Figure 5.16). Although the Grid status component shows similar data for each pool, this data is only available for a five minute window period and is therefore inadequate for the purposes of job submission.

5.3.4 Performance Enhancements

During the development of the Web interface, certain components and features were found to be too inefficient or too slow, necessitating alternative techniques. This section will present an overview of some of the main performance enhancements that helped make the interface more responsive.

When developing a system that consists of so many different components, one often blindly and incrementally continues to add components until all the desired functionality is present. This approach, while leading to the desired result, is often not efficient, as was the case with this system. When loading the interface with all the components in place, it was found that it took far too long to load into the browser. The reason for this was that each component was making its own calls to the server, thus resulting in a flood of communication between the browser and server. In order to solve this problem, certain components were not loaded in full when the application was first loaded into the browser. Only important components such as the main interface, Grid status and file transfer components are initially loaded. Only as a user makes a request for certain functionality, such as a job submission for example, is the component loaded into the browser. This approach has two benefits, the first being a reduction in overall initial data transfer, thus reducing bandwidth, and the second being that unnecessary components, that might not be needed for a particular session, are not loaded. By introducing this technique, interface load times were reduced from approximately 28 seconds to approximately 3 seconds on a particular machine.

One final enhancement, briefly mentioned in Section 5.2.4, has to do with the importing of data from the database. Since many of the interface components, particularly the job query and Grid status components, have to retrieve data from the database each time they are loaded, this common operation had to be improved. Since the job table, for example, can easily contain millions of rows due to the status daemons adding data at five minute intervals, data extraction from this table took a long time. In order to alleviate this problem, a number of database indices were put in place on affected tables, resulting in improved performance.

5.4 Summary

This chapter has presented the infrastructure that was put in place in order to build a Grid Web interface as well as the design details pertaining to the interface itself. Furthermore, a high-level view of the complete system was provided. The techniques that were used to build the interface as well as the methodologies that were used were discussed, paying particular attention to the resource-centric view that the interface has assumed. The use of each of the interface components was also discussed in detail with accompanying discussions on how relevant components serve to abstract and simplify the Grid job submission process. Furthermore, an illustrated view of each of these major components was provided. Finally, an overview of two important performance enhancing techniques was discussed.

Chapter 6

Case Studies

6.1 Introduction

The aim of this research was to create a Web interface that abstracts and simplifies the use of Grid computing tools. To meet this objective, a series of prototype evaluations were held which, after analysis, led to the final design of the system. With the design finalised, the next step after implementation was to test the system for completeness and real world applicability. The definition of completeness in this case is a system that includes all the necessary parts or elements needed to deploy a parameter-sweep Grid application. However, it is not possible to directly prove that such a system is complete. For this reason, a number of case studies representing a set of real world computational tasks were picked in order to provide evidence that the system is useful for its intended task. This chapter will present an overview of each of these case studies in a structured manner so as to highlight the reasoning, objectives and results obtained from each case study in turn. Finally, an overview of the lessons learnt from the construction of the case studies will be provided.

6.2 Assumptions

All the case studies discussed in this chapter assume that certain facets of the system are kept constant. This ensures, firstly, that the results obtained can be compared against one another and, secondly, that conditions that could potentially influence the accuracy of the results are kept constant. For this reason, only the Condor scheduling system was used to run jobs, as different scheduling systems take different amounts of time to complete different tasks such as job creation and result generation. Furthermore, all case studies, with the exception of Case Study II, make use of the Linux operating system and Intel X86 platform. Thus the filtering by cluster option present in the filtering wizard (see Figure 5.7) was not used as the entire Grid was utilised when running each case study. Finally, the last assumption is that all of the approximately 60 cores were operational and that no other Grid jobs were executing or in the job queue when each of these case studies was run.

6.3 Case Study I : Whetstone Benchmark - Single OS

6.3.1 Overview and Objectives

The whetstone benchmark, a classic computational performance benchmark, formed the first case study. This synthetic benchmark was first described in the 1976 paper by Harold Curnow and Brian Wichmann [Curnow & Wichmann, 1976], and describes one of the most well-known measurements of floating-point computational performance. The speed of a computational device measured using the Whetstone benchmark was reported in Kilo Whetstone Instructions per

Second (KWIPS). As computers became more powerful, this metric progressed to Millions of Whetstone Instructions Per Second (MWIPS). The whetstone benchmark today is measured in more conventional Millions of Instructions per Second (MIPS)

This case study was chosen due to its simplicity as a parameter sweep application. This case study did not serve to illustrate any specific feature of the interface, but was used to test basic interface functionality. Therefore, the main objective was to test the interface submission processes, database functionality and job query subsystem before more complex case studies were attempted. This case study conforms to the set of assumptions as discussed in the previous section and was therefore only run using one operating system.

6.3.2 Process

Executable

The Whetstone executable consists of one source file: `whetstone.C`. The executable does not rely on any libraries, contributing to the trivial nature of this case study.

Data Staging

Since the Whetstone benchmark consists of only one file, the executable, no files other than the executable needed to be staged. The executable was copied to the submission server using WebDAV, but this could just as easily have been done by using the interface file upload functionality.

Parameter Definitions

The Whetstone benchmark takes only one input argument— the loop count. This value specifies how many loops of the benchmark code to run in order to generate machine performance statistics. The more loops run, the more accurate the result will be— however, this is at the expense of compute time. Since the test-Grid consisted of approximately 50 Linux-based cores needing benchmarking, 50 such jobs were submitted to the Grid. The benchmark code was therefore run on every core present in the Grid. Since the benchmark code has no parallel implementation, it was necessary to add an input argument at the argument enumeration step of the wizard in order to generate 50 runs of the application. To do this, the number type with an increment of one was used in conjunction with the the flag type (number of loops). The number type is ignored by the Grid job at run-time as the Whetstone code considers only the final argument. An example is shown below:

- whetstone 1 10000
- whetstone 2 10000
- whetstone 3 10000

6.3.3 Results

Table 6.1 provides an overview of the results obtained when submitting the Whetstone job to the Grid. Since this job is trivially small, the 550 Kb of data represents 50 copies of the executable. Due to this small amount of data, a short submission time of 1.487 seconds is also noted. Since the Condor scheduler submits jobs by creating a spool directory for each job on the submit machine, this value is representative of the time taken to create 50 folders and copy a version of the executable into each of these folders. Again, since the job consists of only 50 runs, the PSDL file generated at the job creation step is small and takes only 0.258 seconds to render.

Table 6.1 also shows an overview of the time taken to submit jobs both over the Grid and on a single machine. Since scheduling systems have to match jobs to resources before jobs are sent off on to the Grid, the process of match-making can take a large amount of time. This is evident from the difference between real time and Grid time, where real time refers to the actual wall clock time taken between submission and completion of a job and Grid time refers to the time spent executing on remote Grid nodes. Finally, the effects of the Grid can be seen from the difference in time between the serial and Grid instances. For all case studies presented in this chapter, the serial time is calculated by adding together the individual real times for the entire run of the case study. For example, if a case study consists of ten sub-jobs each running for two seconds of wall clock time, the serial time would be approximately 20 seconds.

Event	**Time**
PSDL Generation	0.3
Data Size	550 Kb
No. of runs	50
Submit Time	1.5
Grid Time	∼1867
Real Time	∼2248
Serial Total	∼45757

Table 6.1: Whetstone Single OS performance data (time reported in seconds)

6.3.4 Reflection

Submission of the Whetstone benchmark to the Grid using the Web interface was a trivial exercise. With the exception of the extra argument that was added in order to generate 50 copies of the benchmark for Grid purposes, no further complications were encountered. The interface could handle such situations more elegantly— however, since most parameter-sweep applications take at least one input argument so as to augment each run of the application with a different set of input data, encountering such situations would be rare.

6.4 Case Study II : Whetstone - Multi OS

6.4.1 Overview and Objectives

The Web interface was designed in such a way so as to allow Grid jobs to run on multiple available operating systems at the same time, given that binaries or executables for each platform exist. This case study, a duplicate of Case Study I, was chosen to highlight this feature of the system. The same submission procedure was followed as for Case Study I.

6.4.2 Process

Executable

In order to run this Grid job on multiple operating systems, the Whetstone source code was compiled for both the Windows and Linux platforms. Furthermore, job submission using the interface requires the compiled binaries to be renamed so as to follow the naming convention: `executableName.$$(OpSys).$$(Arch)`. This convention requires the user to rename the binaries by substituting the variable names `OpSys` and `Arch` with the actual operating system and architecture names chosen in the filtering window. A user does not have to worry about any

other specifics of a heterogeneous job submission apart from manually renaming binaries. The interface will change submission modes automatically depending on the number of operating systems selected at the filtering step.

Data Staging

Since multiple operating systems are used in this case study, two files need to be uploaded to the server, namely the two binaries. Once again, no input data was required.

Parameter Definitions

Job submission to multiple operating systems requires no change to the argument selection as the executables accept the same input arguments irrespective of the operating system on which they are run. The argument enumeration steps discussed for Case Study I therefore also apply to this case study.

6.4.3 Results

Execution times are approximately equal for both Case Study I and Case Study II as only the machines upon which jobs were executed were changed.

Event	Time
PSDL Generation	0.3
Data Size	550 Kb
No. of runs	50
Submit Time	1.5
Grid Time	∼1925
Real Time	∼2279
Serial Total	∼46787

Table 6.2: Whetstone Multi OS performance data (time reported in seconds)

6.4.4 Reflection

Grid job submission to multiple operating systems provides flexibility in a Grid environment consisting of inherently heterogeneous resources. This case study has shown the interface's capability of submitting jobs to multiple operating systems. However, an extension of the interface to allow for job submission to different OSs could also be beneficial.

6.5 Case Study III : Text Indexer

6.5.1 Overview and Objectives

In order to effectively search through large quantities of data, techniques such as indexing are used to speed up this process [Cacheda et al., 2005]. Search engines in particular have popularised such techniques due to the sheer size of data collections that have to be searched through and the time frame, in the order of a few hundred milliseconds, in which such results have to be retrieved. Indexing works by creating an inverted file which contains a list of words (with no duplicates) that appeared in the original file, as well as a numerical value next to each word representing how many times that word appeared in the original file. Once inverted files have been created, document ranking algorithms are applied and searches can be conducted. It should be noted, however, that many important indexing steps have been left out of this discussion. The

job that was submitted for this case study was custom-written and did not contain any logic that would make it more or less suited for use in the Web interface.

Current indexing techniques increasingly move towards distributed and clustered architectures in order to cope with growing volumes of data. The Hadoop [Feldman et al., 2006] approach, for example, makes use of many clustered execute nodes in order to process data in parallel by making use of MapReduce [Dean & Ghemawat, 2008]. This approach works well but requires much investment in the form of dedicated clusters. Since indexing is particularly suited to the parameter sweep model, due to its SIMD nature, it therefore fits the Grid computing paradigm as well. The indexing process can therefore make use of vast quantities of non-dedicated compute resources in order to build the indices necessary for fast query response times.

The main objective in running this case study is to show that a non-trivial indexing job can be formulated into a parameter sweep application and deployed to a Grid using the Web interface. The Grid therefore provides the scalable infrastructure necessary to process vast quantities of data in parallel at a fraction of the cost of a clustered approach. Furthermore, as with each case study to come, the level of complexity of this job has increased from that of Case Studies I and II, thus testing the ability of the interface to handle increasingly complex real-world problems.

6.5.2 Process

Executable

The text indexer case study consists of only one binary, named `indexer`. As with the previous two case studies, this executable does not rely on any libraries.

Data Staging

Since the indexing application is a SIMD application, both the executable as well as input files needed to be copied to the server. The input files used for this case study were text files retrieved from Project Gutenberg [Gutenberg, 2008], including Shakespearean plays and other textual data. The data as well as input files totalled approximately 90Mb, of which 74Mb can be attributed to input files. All files were staged using the WebDAV interface. Although a real-world Grid would typically use a much larger dataset than used for this case study, the aim with the case studies was to test the interface's ability to handle different types of parameter sweep applications and not the ability of the underlying schedulers to manage vast quantities of data.

Parameter Definitions

The text indexer takes two arguments— the first is the name of the inverted file to be created and the second is an input file. For the first argument, an auto-incremented integer value was used to name the inverted file. The input file name could not be used since it is possible that another file in the future might have the same name. The number type was used to specify an increasing integer value starting at one and having an increment of one. For the second argument, the input file wizard was used to specify a directory of input files on the server. The parameter sweep nature of the multiple-file type was utilised in order to assign one input file to each run of the application. This application differed from the benchmark application since the number of runs was not determined by the user but by the number of input files the user specified. The job queue therefore contained more jobs than active Grid nodes. An example of the sweep is shown below:

- indexer 1 input1.txt
- indexer 2 input2.txt
- indexer 3 input3.txt

6.5.3 Results

Table 6.3 provides an overview of the results obtained when submitting the indexing job to the Grid. As can be seen from the table, 500 runs (i.e, 500 text files) were queued for indexing using the Grid. Since the number of runs increased by a factor of 10 from Case Study I to this case study, the PSDL generation time has increased accordingly. The increase in time, however, is expected due to the generation procedure having $O(MN)$ time complexity. Even with the increase in time, however, 1.8 seconds is still acceptable for a job of this size. Table 6.3 also shows the time recorded for the submission of this job to the Grid. This time increased substantially from the previous case studies due to the overall footprint of the job having increased, both in terms of argument complexity and size. Submit time contributed to the longest delay in the job creation/launching process.

Event	**Time**
PSDL Generation	1.9
Data Size	88 MB
No. of runs	500
Submit Time	25
Grid Time	∼1024
Real Time	∼1649
Serial Total	∼6832

Table 6.3: Text indexer performance data (time reported in seconds)

6.5.4 Reflection

Apart from the extra wizard step of having to add in an input file directory, the job creation process is the same as for the previous case studies. Even though the complexity of the job was increased, the interface did not require much additional effort from the user.

6.6 Case Study IV : Audio Converter

6.6.1 Overview and Objectives

Many applications, such as word processors, allow for the conversion of one document format to another [Sommerer, 2004]. Similarly, Web applications often convert data into language neutral, structured formats, such as XML in order to communicate with other services. Furthermore, data conversions are typically done in bulk as part of some process. Since such conversions are usually SIMD in nature as well as computationally intensive, it is possible to parallelize such a bulk process and in so doing leverage the power of the Grid.

This case study was used to illustrate how the Web interface can be used to convert files. A batch conversion from one proprietary audio format (.wma) to an open format (.mp3) was performed using the interface. Furthermore, the level of complexity of this case study when compared to the previous case studies increased substantially.

6.6.2 Process

Executable

Unlike the previous case studies, this case study makes use of a bash script as the "executable", called `wmamp3.sh`. Since the audio converter written for this case study makes use of existing conversion software, it was necessary to find a way of running these pieces of software in sequence. The bash script for this job is shown below:

```
#!/bin/bash

#extract supporting files and software
tar -xvf encoders.tar

#convert wma to a temporary wave with Mplayer
./mplayer -vo null -vc dummy -af resample=44100 -ao pcm:waveheader $1

#convert wave to mp3 using toolame
./toolame -m s audiodump.wav -o $1

#remove wma extension and replace with mp3
mv "$1" "‘basename "$1" .wma‘.mp3"

#remove temporary wave file
rm audiodump.wav
```

The bash script shows how two pieces of software (Mplayer and toolame) are used to convert from one audio format to another by making use of an intermediate `WAVE` file. However, at the executable selection step in the wizard, neither Mplayer nor toolame were specified, as only one executable may be submitted per job. The bash script is therefore the only executable that needed to be specified at the executable selection step in the wizard.

Data Staging

This case study in essence relied on the existence of three executables: the bash script, Mplayer and toolame, as well as the supporting library files needed by these tools. Furthermore, since the aim of the case study was to convert `wma` files to `mp3` files, a set of 500 input files needed to be transferred to the submission server. The entire job was therefore broken down into three sets of files: the executable, the supporting software and the input files. The supporting software, along with associated libraries was packaged into a `tar` archive. This archive is extracted as soon as the script is run and therefore does not rely on the execute nodes having the software pre-installed.

As can be seen from Table 6.4, the size of the job increased considerably from the previous case studies, to an overall footprint of approximately 4.8GB. Of this 4.8GB, 2.8GB consisted of `wma` audio files and 2.0GB can be attributed to 500 copies of the bash script and `encoders` archive which is approximately 4MB in size. As with the first case study, the scheduler creates a spool directory containing all files needed for each sub-job. Since there are 500 sub-jobs, a spool directory is created for each before distributing the job to the remote Grid nodes. This is only done in non-NFS environments such as the Grid installed for these case studies. Once again, all files were staged using the WebDAV interface.

Parameter Definitions

The bash script was written to accept one argument, an input `wma` file, as is evident from the $1 entries in the bash script above. In order to specify these input files, the directory of `wma` files was selected at the input file specification step in the job creation wizard. At the input argument enumeration step of the wizard, the files in this directory were split up among 500 runs of the audio converter using the multiple-file type. Once again, the number of runs was calculated by counting the number of files present in the input directory. Although the script only takes one input argument, the input file, the `encoders` archive needed to be sent along with each job. To do this, a single-file argument was specified as the second argument to the bash script. By doing this, the archive was assumed to be an input file by the interface and was then bundled with each run of the Grid job. An example of a few runs is shown below:

- wmamp3.sh music_file1.wma encoders.tar
- wmamp3.sh music_file2.wma encoders.tar
- wmamp3.sh music_file3.wma encoders.tar

6.6.3 Results

Table 6.4 provides an overview of the results obtained when submitting the audio conversion job to the Grid. The PSDL generation time once again increased in direct proportion to the increase in complexity of the arguments specified, as expected. The most noticeable decrease in performance, however, is attributed to the submit time. Since the amount of data that needs to be spooled for submission is 4.8GB, the submission process took approximately 17 minutes. Although 17 minutes is considered to be a long time for any computational process to complete, the long-term benefits of running this job on the Grid as opposed to a single machine is evident from the wall clock speedup by a factor of 21, or 0.67 hours as opposed to 14.6 hours.

Event	**Time**
PSDL Generation	3.0
Data Size	4750 MB
No. of runs	500
Submit Time	1036
Grid Time	∼1718
Real Time	∼2438
Serial Total	∼53166

Table 6.4: Audio conversion performance data (time reported in seconds)

6.6.4 Reflection

This case study has shown how common data processing problems can be "Gridified" with relative ease and submitted using the Web interface developed as part of this research. The case study has demonstrated how applications with non-trivial software dependencies can be submitted using the interface with only a few minor changes to the way in which jobs are specified. Apart from the notion of a bash script as an alternative executable, all interface operations remained the same for this job as they did for the previous case studies.

6.7 Case Study V : Distributed Rendering

6.7.1 Overview and Objectives

Rendering of a scene in Computer Graphics refers to the conversion of a high-level scene description into a 2D image [Angel, 2001]. Such scene descriptions contain definitions of objects in the scene as well as their locations, lighting effects, shadow effects, animation and other such artefacts. Such scenes are typically complex in terms of their specification and interactions among scene elements and therefore require much compute time to render. Scene rendering is, however, just one facet of Computer Graphics. Animations, or consecutive sets of scenes, are more widespread. In order to render an animation, it is possible to split up such an animation into consecutive blocks and process each block individually. Once each block is processed, the final rendered animations can be merged. Since rendering applications are generally SIMD, it is possible to create a Grid job to render an animation.

This case study illustrates how a Grid job can be specified in order to render an animation. The animation is split up into sets of frames, where each set is sent to a Grid node for processing. This case study is the most complex in terms of its software dependencies and the way in which jobs are specified.

6.7.2 Process

Executable

As with the audio conversion case study, this case study makes use of a bash script as the executable, called `blender.sh`. The difference between this case study and the audio converter is that only the blender [Blender, 2008] software is required for rendering the animation. The blender executable was not used as the sole executable because libraries have to be passed along with the application, and some files need to be removed before the rendered frame is copied back. By removing these files, the size of the data that needs to be copied back is reduced, which therefore shortens the retrieval time as well as the disk space requirements of the output. These operations cannot be performed by the blender executable, and therefore a bash script was created to achieve these goals. The bash script for this job is shown below:

```
#!/bin/bash
#

#unzip tar archive
gunzip files.tar

#extract tar archive
tar xvf files.tar

#set the library path to the local library files needed by blender
export LD_LIBRARY_PATH=.

#render the duck avi from frame $1 to frame $2
./blender -b duck11_peaking.blend -x 1 -o outputOfRender -F MOVIE \
-s $1 -e $2 -a

#remove all files no longer needed so avoid copying back to server
rm blender
rm lib*
```

```
rm files.tar
rm files.tar.gz
rm duck11_peaking.blend
```

The script is similar to that of the audio converter script, except for the exporting of the library path. Blender requires that certain libraries exist on the target machine. Since the existence or correct version of these libraries cannot be guaranteed in a Grid environment, the libraries were bundled with the Grid job and were set in the local environment using the `LD_LIBRARY_PATH` environment variable.

Data Staging

In terms of input files, this case study was slightly less complex than that of the audio converter as only one animation from the open movie Elephants Dream [Elephant's Dream, 2008] was used during the rendering process. Since only one animation file was rendered by splitting it into sets of frames, the source file was simply added into the `tar` archive. This eliminates the need to specify a single-file type. In terms of data staging, only one copy of each of the bash script and data archive was transferred to the server.

The total size of the job was approximately 790MB. This size is solely attributed to the size of the bash script and supporting tar archive and therefore shows the importance of removing extraneous data files once the job has completed. Once again, all data was staged using the WebDAV interface.

Parameter Definitions

Since the input animation file was bundled with the application, only two parameters were needed by the script, namely the starting frame and ending frame to render. To specify these ranges, the number-type in the argument enumeration step of the wizard was used with the same increment, but with differing starting and ending values. The job was split into 89 runs of five frames each, giving a total animation frame count of 445 frames for the `duck11_peaking.blend` animation. Figure 6.1 shows the final output of each of these runs as an illustration, as well as an enlarged snapshot of one of these frames. Finally, as with the audio conversion case study, the file archive was added as a single-file type. An example of a few runs is shown below:

- blender.sh 1 5 files.tar.gz
- blender.sh 6 10 files.tar.gz
- blender.sh 11 15 files.tar.gz

6.7.3 Results

Table 6.5 provides an overview of the results obtained when submitting the distributed rendering job to the Grid. Since the arguments specified did not contain the multiple-file enumeration as in the audio conversion case, the PSDL generation time was reduced substantially. The job took 106.736 seconds to submit to the Grid; a much more acceptable time than that of the audio converter, attributed to the small overall job size. Once again, Grid time was substantially less than the time for execution on a single machine, as expected.

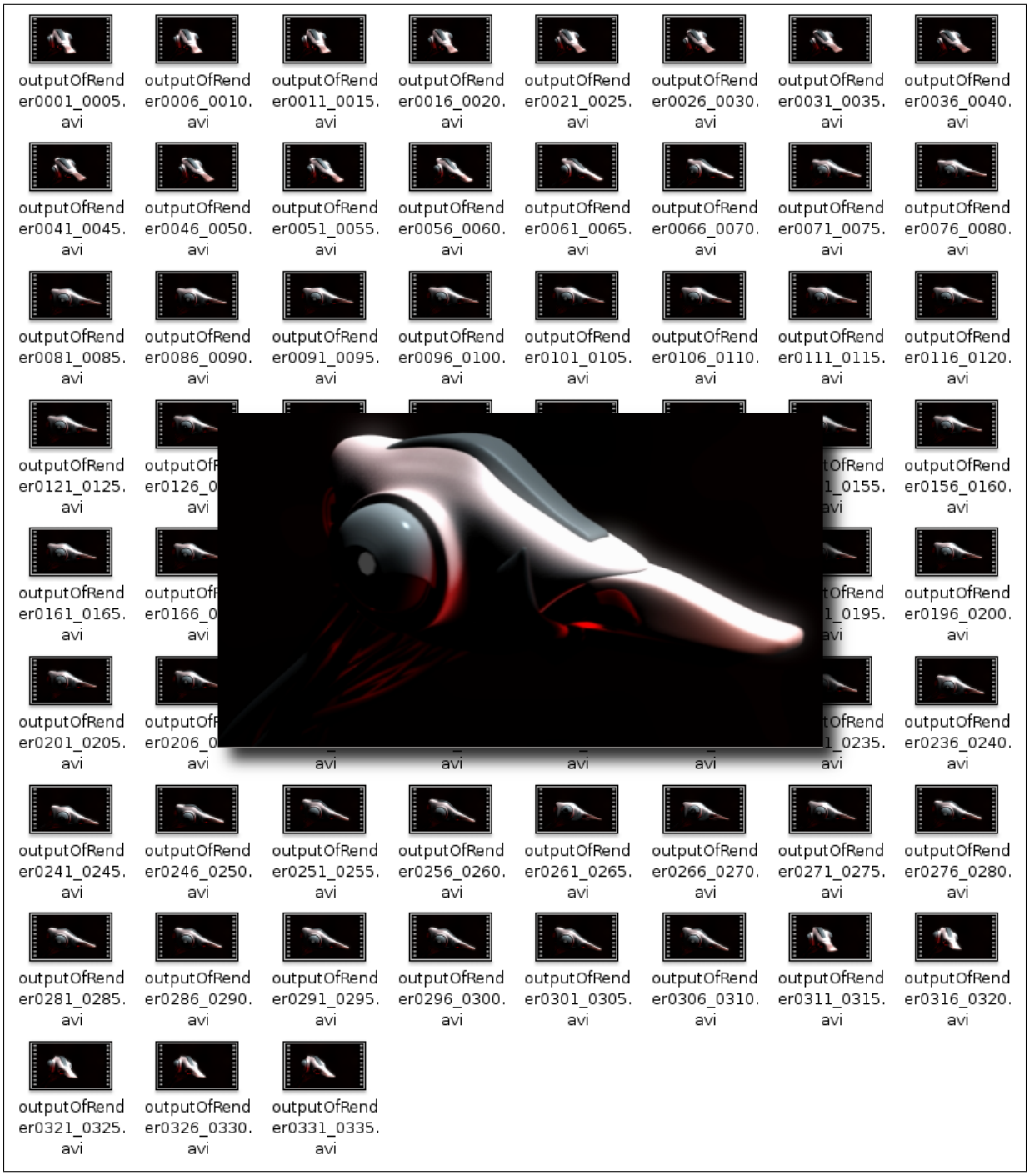

Figure 6.1: Output of distributed rendering case study with enlarged snapshot of a single frame

Event	Time
PSDL Generation	0.4
Data Size	783 MB
No. of runs	89
Submit Time	106
Grid Time	∼1320
Real Time	∼1680
Serial Total	∼13920

Table 6.5: Distributed rendering performance data (time reported in seconds)

6.7.4 Reflection

The specification of the distributed rendering Grid job was accomplished without any difficulties. Even though this job was very different to the audio converter in terms of the way it accepted arguments, the process of creating and submitting these jobs was similar. As with the audio converter, the only possible complexity was the creation of a wrapper script that orchestrates the running of the job on the Grid.

6.8 Scope and Limitations

The case studies discussed in this chapter were run under controlled conditions on a stable campus intranet. Although such an environment is necessary to provide consistent test results, the reality is that Grid systems are volatile and can lead to unexpected delays or even application errors due to timeouts, data corruption and network element failures. Since Grid jobs usually take a long time to complete, turnaround time is generally assumed to be long. This is especially true in the case of parameter-sweep applications. These applications typically consist of tens of thousands of runs and can take a long time to return results from the various Grid nodes for the various reasons described above.

Since the case studies in this chapter were relatively small and since Grid nodes were dedicated solely to running these case studies, turnaround time was short. However, the turnaround time of Grid jobs is independent of the performance of the Web interface. The interface is merely a portal to the Grid job submission system and is therefore not affected by Grid jobs that take a long time to complete.

Some of the case studies presented in this section make use of wrapper scripts. These scripts are written in order to execute a sequence of operations that make up the Grid job itself. It can be argued that these scripts require some level of sophistication. Although this is true, writing a wrapper script to perform a complex task is akin to writing a C++ or Java application to do perform the same task. The purpose of the script in the case studies presented in this chapter, serves only to act as the executable for the Grid task.

One final aspect that deserves mentioning is that of data volumes. With all the case studies evaluated in this chapter, a measure of submission time was presented. The submit time was dependent on the speed of the underlying scheduler's ability to create new job images. In other words, for Grid jobs containing little data, the scheduler would be able to create a new job image in a few seconds. However, as the size of the Grid job increases, the interface tends to perform poorly as it waits for the scheduler to complete its task. In production Grid environments, such a latency could not be tolerated and this is a known limitation of the interface. The Future Work section will elaborate on some of the ways in which these limitations could be overcome.

Case Study	PSDL Gen.	Data Size (MB)	Runs	Submit	Grid Real	Grid Total	Serial Total
Whetstone Benchmarks	0.26	0.6	50	1.5	∼1867	∼2248	∼45757
Whetstone Multi-OS	0.25	0.6	50	1.5	∼1925	∼2279	∼45887
Text Indexer	1.9	90	500	25	∼1024	∼1649	∼6832
Audio Converter	3.0	4750	500	1036	∼1718	∼2438	∼53166
Distributed Rendering	0.7	783	89	107	∼1320	∼1680	∼13920

Table 6.6: Summary of case study performance data (time reported in seconds)

6.9 Summary

This chapter has provided an overview of five case studies that were conducted by making use of the Web interface. The aim of these studies was to provide evidence for the "completeness" of the interface as a tool for the creation of real-world parameter sweep applications on a Grid. Since it is not possible to prove that the interface is complete, nor possible to claim this without supporting evidence, these case studies provide strong evidence in support of the interface's parameter sweep capabilities. The case studies have shown the flexibility of the interface in terms of its capacity to handle varying job types. Although the ways in which each of the case studies has been specified are similar, the software dependencies and arguments differed greatly. Furthermore, the sizes of the jobs were varied in order to test various elements of the interface for robustness. The ability of the interface to handle each of these varying requirements, with very little change to the process of creating and submitting jobs, provides evidence as to its capability of successfully handling different classes of parameter sweep applications. For comparison, Table 6.6 reports the timing results obtained from all case studies.

Chapter 7

Evaluation

7.1 Introduction

Chapter 6 gave an overview of 5 case studies that were used to evaluate the Web interface designed as part of this research in terms of its ability to accommodate varying types of real-world applications. By increasing the complexity of each case study in turn, the interface was evaluated in terms of completeness. As mentioned in Section 6.1, completeness of the system in this instance is defined as a system that includes all the necessary parts or elements needed to deploy a typical parameter-sweep Grid application. For a system designed with human interaction in mind, however, such a set of case studies, although important, is not sufficient by itself. It was therefore necessary to perform a thorough set of user evaluations in order to determine if the design was successful in terms of usability and intuitiveness. Furthermore, although such an interface might be found to be usable, intuitive and support all the features of parameter sweep applications expected by users, it might still perform poorly. Since one of the aims of this research was to create a lightweight interface, it was deemed necessary to evaluate this aspect of the system.

This chapter will present the evaluation of the Web interface from two perspectives, namely user and performance evaluation. The results from both of these evaluations will be discussed in isolation followed by a summary of the results in their entirety.

7.2 User Evaluations

One of the questions this research attempts to answer is if it is possible to create an interface to a Grid scheduling system with a high degree of usability. In an attempt to answer this question, various rounds of user consultations, prototyping and evaluations were conducted. These processes were, however, conducted before the software implementation phase. In order to verify that the implementation of the Web interface resulted in a system with a high level of usability and intuitiveness, a final round of user evaluations was conducted. This section will provide an overview of the test subject selection process, a detailed discussion of the experimental design, statistical analysis techniques as well as the results obtained from these analyses.

7.2.1 Population and Evaluation Environment

Before any user evaluation strategy was contemplated, test subjects had to be chosen. Highly computer literate students from the Science and Engineering domains were recruited. This study has assumed that most research scientists have an average to above average degree of computer literacy, therefore the sole use of Science subjects is unlikely to bias results significantly.

24 students were selected to participate in the user evaluations. According to Nielsen et al. [Nielsen & Landauer, 1993], only 16 evaluations would be worth their cost in finding between 75% and 100% of the usability problems with an interface. The number of test subjects selected for the experiment is therefore well above the recommended minimum. Furthermore, Nielsen also states that more evaluations should be conducted depending on the level of usability the system is aiming for. By selecting 24 subjects, 57% more than necessary according to Nielsen, the number of potential usability problems that could be identified was maximised.

Of the 24 test subjects, 23 were Computer Scientists and one was an Electro-Mechanical-Engineer. Selection was made in advance by considering Computer Science course results and selecting only the top 25% of students. These students were contacted via email and self-selected on a first-come-first-served basis. All subjects were at least in their second year of tertiary level study. Furthermore, subjects who participated in pre-implementation prototype evaluations (see Chapter 4) were excluded from this evaluation process. These qualifications and level of study of test subjects are reported in Table 7.1

Discipline	Computer Science	23
	Electro-Mechanical-Engineering	1
Academic Level	BSc. 2nd year	5
	BSc. 3rd year	8
	BSc. Hons	7
	MSc.	3
	MEng.	1

Table 7.1: Test subject discipline and academic levels

The environment in which the test subjects conducted the evaluations needed to be kept constant in order to produce consistent results. Test subjects should never be distracted by other people and should always be in a quiet environment in order to concentrate on the task at hand. For this reason, an experiment room was used in order to keep sound levels to a minimum. Furthermore, test subjects were placed with their backs to the rest of the objects in the room, thereby eliminating any distractions.

7.2.2 Experimental Design

Section 4.2.1 provided an overview of the minimum requirements that research scientists consulted at the beginning of this study believe an interface to a Grid should have. These include a mechanism for being able to view the status of the Grid, submit a job to the Grid and monitor the status of such jobs. The evaluation of the interface was therefore split up into an evaluation of each of the components providing this functionality. To do so, three independent tasks were designed, each focusing on one of these initial requirements. Figure 7.1 provides a high-level view of these three main tasks, along with the sub-tasks that need to be performed for each independent component evaluation (see Chapter 5 for information on each sub-task). Furthermore, each task was followed by a short questionnaire which enabled test subjects to assess these components individually. The questions required both written answers as well as answers to Likert-scale questions. Scaled questions allow subjects to rate levels of usability, intuitiveness and response time, while questions requiring written answers allow subjects to report on aspects of the system they liked and disliked as well as allowing for subjective comments on aspects such as aesthetics and possible improvements.

Even though the three tasks mentioned above formed the basis of the user evaluations, other important information needed to be gathered at various stages of the evaluation process. For

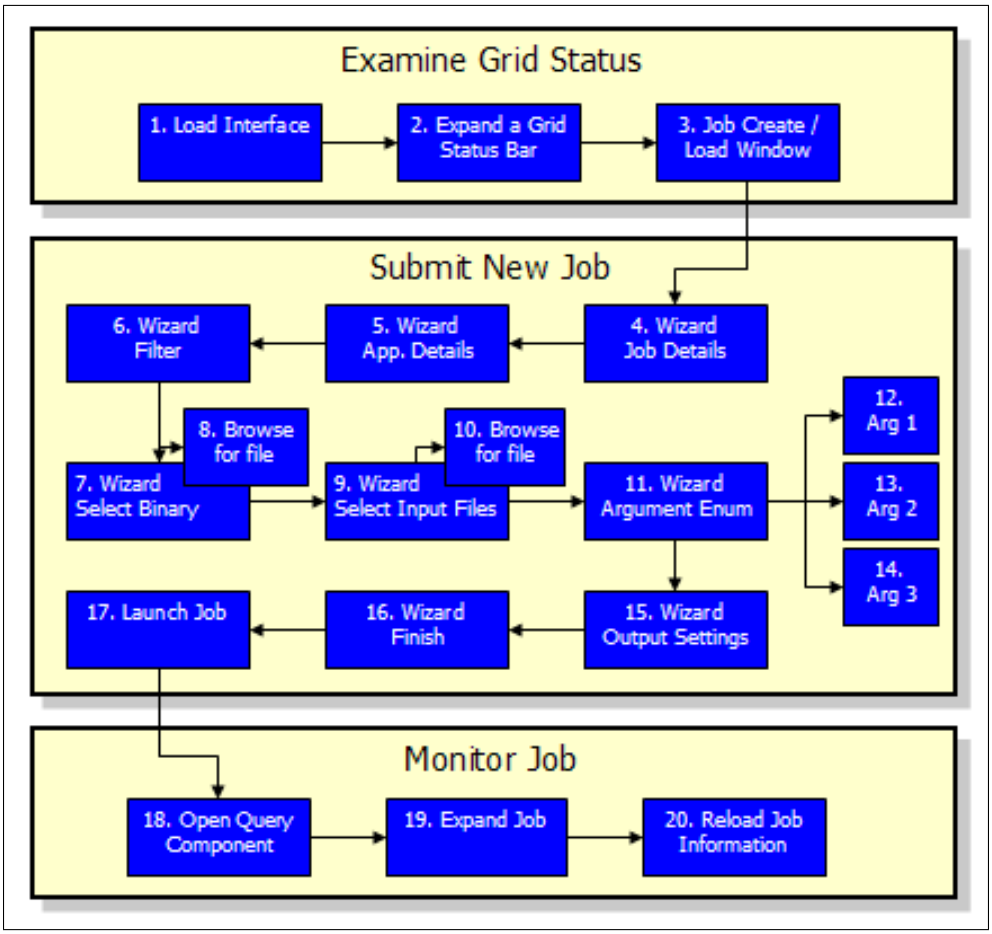

Figure 7.1: User evaluation tasks represented as a flowchart of sub-tasks

this reason, a 15 page questionnaire (see Appendix C) was devised, that allowed test subjects to complete the evaluation in a structured manner. Each section of the experiment / questionnaire is discussed in more detail in the sections to follow.

Background

The first part of the questionnaire gathered data on the qualifications, research interests and background information of test subjects. Background on HPC as well as Grid knowledge of test subjects was gathered in this section. Finally, data on how often test subjects make use of Web applications such as Gmail, Flickr and Facebook was gathered.

Presentation

At the outset of the experimentation phase, it was expected that test subjects would not be familiar with the concept of a computational Grid. For this reason an introductory slideshow on Grids, scheduling systems and parameter sweep applications was presented prior to the evaluation. The presentation took the form of an automated, pre-recorded Powerpoint slideshow of approximately 13 minutes in length. Although an objective of the Web interface is to show that users without Grid-specific knowledge can make use of a Grid, all users need to understand basic Grid concepts. The presentation was conceived in such a way that the results of the study are not affected by the information provided in the presentation. This is due to the nature of the content present in the presentation being informational and at a high-level.

On completion of the presentation, a questionnaire collected information from test subjects that assessed their understanding of the information presented in the slideshow.

Grid Status Task

As already mentioned, the assessment of the Web interface was split up into three tasks, each with its own questionnaire. The Grid status task is the first of these. For this task, test subjects were instructed to find the Grid status component displaying information related to one of the Grid pools. Once subjects had found the component, they were required to fill in a table by extracting the relevant information from the values presented in the component. Finally, test subjects were asked to provide their interpretation of the icons present in the summary view of the status component. This task was then followed by a questionnaire.

It should be mentioned that a live Grid status window was not used— instead, only a snapshot of the Grid at a previous point in time was provided. This approach provides all users with the same experience and also removes the variable nature of a Grid environment from the study.

Job Submission Task

The second task required test subjects to create and submit a job to the Grid using the Web interface. The task was outlined in the form of a real-world problem statement and no step-by-step instructions for completing the task were given. This approach forces test subjects to understand the task at hand and not simply test their ability to follow instructions.

The audio conversion job presented in Section 6.6 was modified slightly for use in this task and all input files and binaries were pre-staged. As with the Grid Status task, no jobs were actually submitted to the Grid, although it appeared this way to the test subjects. Once test subjects completed the task they were asked to complete a questionnaire similar in structure to that of the Grid Status task.

Job Query Task

The final task required test subjects to monitor the job they had just created and submitted to the Grid. Even though the job was not actually run on a live Grid, the database entries were still created at the time the job was launched. Since test subjects were expected to notice status updates and record these on the questionnaire, it was necessary to update these database entries while observing the subjects performing the evaluation. This technique is known as Wizard of Oz testing [Maulsby et al., 1993]. To accomplish this, a script was written to update the database as the user uncovered certain information, thereby creating the illusion of a fully functional system.

On completion of the task, test subjects were once again requried to complete a questionnaire with questions similar to those found in the previous two tasks.

General Feedback

The final section of the questionnaire allowed test subjects to comment on the general appearance of the interface, outline any possible improvements as well as give an indication of any aspects they liked or disliked. Likert-scale questions that gauged overall responsiveness and similarity of the interface to other Web applications such as Facebook and Gmail also formed part of this section.

7.2.3 Analysis Techniques

The data collected during the user evaluation sessions is both quantitative and qualitative. The quantitative data, as already mentioned, was collected by using Likert scales. Likert scale data, however, is ordinal (categorical) data, which therefore means that the results from Likert-scale evaluations need to be analysed by using non-parametric data analysis techniques. Such techniques are generally used for studying data that can take on a ranked ordering. Furthermore, non-parametric statistical techniques make fewer assumptions about normality of the data being analysed, thereby providing more accurate results for this type of data [Thomas W. MacFarland,].

The results section to follow will make use of two main statistical methods to analyse the data gathered from the user evaluations. The first method, the Wilcoxon-Mann-Whitney U test, is used for hypothesis testing. This test is a t-test for non-parametric data, used for comparing two independent samples. The aim of the test is to determine whether a significant difference exists between two groups. To accomplish this, a null hypothesis is formulated. This hypothesis, denoted H_0, is that two populations have no significant difference between their medians. In addition to the null hypothesis, the alternative hypothesis is formalised, denoted H_1. This hypothesis is the converse of the null hypothesis and expresses the hypothesised result of the test. In order to reject the null hypothesis, the significance level (p) generated by the Wilcoxon-Mann-Whitney test is evaluated. This study will use the standard 95% significance level [Underhill & Bradfield, 2001] as a measure of how well the two distributions converge. If the calculated *P-Value* is less than the significance level of p=0.05, the null hypothesis is rejected in favour of the alternative hypothesis.

The second method that will be used, to determine if correlations between variables exist, is the Spearman Rank Order Correlation method, also known as Spearman's Rho. Values of such correlations range between —1 and 1 where the extremes represent very strong correlations between the datasets. A correlation of 0 implies no relation between the datasets. The analysis will assume that any correlation above 0.7 is significant at p<0.05.

7.2.4 Results

This section will provide an overview of the results of the user evaluations by providing discussions of the results from each of the sections outlined in the experimental design. A discussion of the results for the interface as a whole will be presented along with supporting evidence in the form of statistical analyses.

Test Subject Background Analysis

The main goal of gathering background information on test subjects was to determine how much knowledge they had of HPC and Grid technologies. To this end, questions specifically requiring subjects to rate their knowledge of such systems were present in this section. Questions relating specifically to Grid computing were used to gather data on whether test subjects had used such environments before or not. The responses to these questions are summarised in Table 7.2.

Knowledge of	**Responses**				
	Excellent	**Very Good**	**Neutral**	**Poor**	**Very Poor**
HPC	0	3	13	7	1
Volunteer Computing	0	8	7	5	4
Grid Computing	0	5	12	4	3

Table 7.2: Test subject responses on familiarity with HPC and Grid technologies; n = 24

Statistical Analysis	**Descriptive Statistics**			
	Median	**Minimum**	**Maximum**	**Std_dev**
HPC	3	1	4	0.73
Volunteer Computing	3	1	4	1.10
Grid Computing	3	1	4	0.93

Table 7.3: Descriptive statistics of HPC and Grid knowledge; n = 24

A statistical analysis of the responses reported in Table 7.2 is presented in Table 7.3. The results indicate that a large number of test subjects had a below average understanding of Grid computing concepts. Furthermore, 66% of subjects reported that they were familiar with some Grid theory, while none had any practical experience with such systems. If it can be shown that the interface is intuitive and easy to use by the test subjects with little Grid knowledge who participated in this study, one of the main research questions will have been answered.

When comparing responses to the questions on volunteer computing and Grid computing knowledge, a Spearman Rank Order Correlation test found a strong correlation between knowledge on these two topics. In other words, given that a person has some level of knowledge on the topic of Grid computing, it is likely that they have a similar level of knowledge on volunteer computing. This analysis was then extended to include knowledge of HPC. The Spearmen test showed that there is no correlation between HPC and volunteer computing / Grid knowledge. This indicates that having some level of HPC knowledge does not necessarily imply a similar level of Grid or volunteer computing knowledge. Since some of the test subjects participating in this study had done a course in HPC, it is unlikely that such subjects would be able to perform

better than those that had no HPC knowledge since the results of the Spearman test show a lack of correlation between HPC in general and Grid knowledge.

The final section of the background questionnaire required test subjects to indicate if they had ever heard of AJAX, as well as indicate how often they use Web applications such as Gmail, Facebook or Flickr. Since these applications are heavily AJAX-based, these questions set out to determine whether test subjects would be able to recognise dynamic elements of an AJAX-based interface, as well as how often they are subjected to such interfaces on the Web. Of the 24 subjects, 96% had heard of AJAX. This is not surprising, since test subjects were all Computer Scientists. Finally, of the 24 test subjects, 92% indicated that they made use of Web applications such as Gmail or Facebook daily, while the other 2 use such applications less frequently.

Presentation Analysis

In order to provide test subjects with some level of basic Grid knowledge necessary to understand the tasks which they would subsequently be given, a presentation was shown to all test subjects. Upon completion of the presentation, subjects were asked to answer four questions relating to the presentation. The first two questions asked subjects to rate how well they understood the content of the presentation in terms of Grid concepts and parameter sweep applications. The second two questions, which can be found in Appendix C, required subjects to recall information from the presentation. These questions consisted of statements to which test subjects were asked to indicate a level of agreement. The results from these four questions are summarised in Table 7.4.

Understanding of	**Responses**				
	Excellent	**Very Good**	**Neutral**	**Poor**	**Very Poor**
Grid Computing Concepts	2	18	4	0	0
Parameter Sweeps	10	8	5	1	0
Statement 1 (positive)	5	11	3	5	0
Statement 2 (negative)	1	4	5	**10**	**4**

Table 7.4: Test subject understanding of concepts presented in an informative presentation; n = 24

To determine whether the test subjects understood the information in the presentation, the results from each question were summed and an overall average taken. The average across all test subjects for all questions was 78%. This high overall average provides evidence to suggest that test subjects understood the information they were presented with. The toughest statement that subjects had to validate received an average score of 79%. Of 24 test subjects, 58% managed to infer the correct answer for the question, and 21% selected a neutral answer which is technically incorrect (see the bold figures in Table 7.4). Even with this low value, the overall results provide further evidence to suggest that test subjects understood the contents of the presentation.

Task 1 : Grid Status

After completion of the presentation, test subjects were asked to perform three tasks in line with the requirements set out initially by the scientists interviewed at the beginning of the study (see Section 4.2.1). The Grid status task was the first of these tasks. This task consisted of two sub-tasks, each of which will be discussed in turn.

Sub-task 1 - Information Extraction

The first task required test subjects to locate the Grid status component (see Figure 5.5). They were then prompted to extract four pieces of information from the component, pertaining to the status of one of the Grid pools. The first observation that was made during the evaluations was that many test subjects did not see the component expansion icon (+), thereby getting stuck at this point. This icon changes the view of a pool from its summary view to its expanded view. After alerting test subjects to the function of the icon, they were all able to complete the task. The second observation was that test subjects had difficulty finding the first piece of information required by the task. They were asked to find the number of machines idle or awaiting jobs. This refers to the number of machines that are "up" in a particular pool. Test subjects got confused between this value and the number of cores available in the summary view (see Figure 5.5).

No. Machines Idle	**No. Machines Down**	**No. Architectures avail.**	**No. OSs avail.**
13	24	18	18

Table 7.5: Grid Status Task 1 : Number of correct extractions per task; n = 24

The other information required in this task was easily found by most test subjects. 79% of subjects managed to obtain three or more correct answers for this question. Table 7.5 shows the number of correct overall data extractions for the first Grid status task.

Sub-task 2 - Icon Interpretation

The second task required test subjects to give their interpretation of the icons present in the summary view(s) of Grid pools (see Figure 5.5). Five icons, each representing a different pool statistic, are presented in each summary view. Table 7.6 shows how many test subjects correctly identified each of the icons.

Pool Status	**Total Pool Load**	**Total Pool Memory**	**Total Pool Storage**	**Total Pool Speed**
5	14	22	22	19

Table 7.6: Grid Status Task 2 : Number of correct icon interpretations; n = 24

As can be seen from the table, pool status and pool load have much lower values than the other results. The reason for this can be seen in Figure 5.5. Since the first two icons are situated close together, test subjects did not realise that they were separate. For this reason, many overlooked the pool status icon and simply gave no interpretation for it. Since test subjects assumed that the two icons were joined, they could not make sense of what the icon was representing. A further observation was that these two icons, unlike the rest, do not have any units of measurement attached to them. Test subjects were able to infer what the other icons were

representing with more certainty due to these units being present.

Even with the poor results for the first two icons in this task, the median for number of correct observations is four out of five. This is an 80% success rate for correctly identifying the status icons. This provides evidence to suggest that test subjects found the icons intuitive.

Component Evaluation

After the test subjects performed the two tasks discussed above, they were asked to evaluate the Grid status component in terms of intuitiveness, response time and sensible data presentation. Tables 7.7 and 7.8 provide a summary and descriptive statistics of test subject ratings respectively.

Rating of	**Responses**				
	Excellent	**Very Good**	**Neutral**	**Poor**	**Very Poor**
Intuitiveness	2	14	7	1	0
Response Time	3	13	5	3	0
Sensible Data Presentation	3	15	6	0	0

Table 7.7: Grid status component ratings; n = 24

Statistical Analysis	**Descriptive Statistics**			
	Median	**Minimum**	**Maximum**	**Std_dev**
Intuitiveness	4	2	5	0.69
Response Time	4	2	5	0.87
Sensible Data Presentation	4	2	5	0.61

Table 7.8: Descriptive statistics of Grid status component ratings; n = 24

As can be seen from the responses, the majority of the scores for all three ratings are in the "Very Good" category. The standard deviations for each response are less than one, indicating a high degree of consensus among the ratings of different test subjects. Since the results of the three separate ratings were so similar, a Spearman correlation matrix was applied to the data to see if a correlation existed between intuitiveness and sensible data presentation, for example. The tests showed no correlation, leading to the conclusion that intuitiveness is not dependent on either response time or sensible data presentation. One can only speculate why this is the case—a possible explanation for this observation could be that test subjects evaluated intuitiveness by comparing the Grid interface to other more familiar interfaces.

Discussion

The results for the Grid Status task indicate that test subjects not only found the Grid status component intuitive to use, but that even with limited Grid knowledge, they were able to make sense of the information presented in the component. As already mentioned, a few problems

were encountered during the evaluation sessions. These problems, however, did not lead to poor ratings of the component, thereby indicating that these problems were minor. Furthermore, test subjects were requested to provide feedback in the form of comments and suggestions for improving the component. Approximately half of the test subjects indicated that tooltips attached to icons on the Grid status page would help significantly in improving the readability of the summary bar. Furthermore, subjects indicated that adding appropriate spacing between the icons, as well as automatically scaling the units of measurement associated with each icon (i.e KB to MB) would make the component more readable. Other suggestions included the provision of a mechanism for making the expansion icon (+) appear clickable, providing a timestamp on the last update of Grid status information as well as a loading "spinner" to alert users to the fact that components are being updated or retrieved. Since the nature of these comments are restricted to component improvements, there is further evidence to suggest that the design of the Grid status component was well received by test subjects.

Task 2 : Job Submission

Sub-task 1 : Job Creation and Launching

The second step in the usage scenario, depicted in Figure 7.1, is the creation and submission of a Grid job. For this task, test subjects were provided with a short problem statement which outlined the nature of a Grid job which they were to submit to the Grid. Furthermore, no step-by-step instructions were provided, thereby making it possible to identify problem areas in the design of the interface. No explicit results were expected from test subjects other than completing the task. However, observational data was gathered.

All test subjects, apart from one, were able to complete the task. The reasons for this test subject not being able to complete the task are unclear as the subject did not provide any indication as to why she failed to complete it. The subject did indicate that she understood the problem statement, so this can be ruled out as the cause. Due to the nature of the failure being unclear, and taking the ambiguity of her responses into account, these responses were excluded from the study.

During completion of the tasks, a few interface usability problems were observed. The first problem was that many test subjects neglected to select an architecture from the filtering wizard (see Figure 5.7). Although all subjects quickly selected the "Linux" operating system as per the problem statement, it seemed as though they did not spot the architecture list. A possible explanation for this is that the selection boxes are in close proximity to one another, thus causing confusion as to where one set of functions ends and another starts. After informing subjects that they were to select an architecture from the list, many noted that they simply did not see the option. The second problem was noted with the file browsing window. Since files in the window were not highlighted when selected, test subjects were unclear as to whether the files were indeed selected or not. This problem, however, can be solved easily. The last major problem occurred during the argument enumeration step (see Figure 5.11). Since the argument fields make use of textboxes in "read-only" mode, test subjects assumed they had to enter the argument information into these textboxes. On attempting to enter data into these fields, nothing happened and test subjects became confused. By changing the appearance of the fields, this problem can be overcome.

On completion of the task, test subjects were asked to indicate how well they understood the problem statement. The majority of test subjects answered positively, with 92% of subjects understanding the task with greater than an 80% level of confidence. One test subject, however, rated his understanding of the task as "very poor", but was able to complete it successfully.

The ambiguity present in his response, as well as the large deviation from the sample mean, led to his results being excluded as an outlier.

Component Evaluation

As with the previous task, upon completion, test subjects rated the task according to the categories presented in Table 7.9. In addition to the three ratings present in the previous task, test subjects also were asked to rate the level of intuitiveness of the filtering and argument enumeration wizards. Since these wizards are the most complex in terms of the Grid job creation process, their level of intuitiveness is important in relation to the entire wizard. Tables 7.9 and 7.10 provide a summary and descriptive statistics of test subject responses respectively.

Rating of	**Responses**				
	Excellent	**Very Good**	**Neutral**	**Poor**	**Very Poor**
Overall Intuitiveness	2	9	9	1	1
Response Time	6	8	8	0	0
Intuitiveness of Filtering Wizard	7	9	5	1	0
Intuitiveness of Argument Enumeration Wizard	2	11	5	4	0
Sensible Data Presentation	4	9	8	1	0

Table 7.9: Job creation component ratings; n = 22

Statistical Analysis	**Descriptive Statistics**			
	Median	**Minimum**	**Maximum**	**Std_dev**
Overall Intuitiveness	3	1	5	0.91
Response Time	4	3	5	0.81
Intuitiveness of Filtering Wizard	4	2	5	0.87
Intuitiveness of Argument Enumeration Wizard	4	3	5	0.91
Sensible Data Presentation	4	2	5	0.82

Table 7.10: Descriptive statistics of job creation component ratings; n = 22

Although test subjects rated the most complex components with a higher than average level of intuitiveness, the results make it clear that something else in the interface caused them to rate it lower overall. The cause for this low rating is most likely to be the problems already mentioned. One test subject rated the interface as having a "Very Poor" overall level of intuitiveness. This test subject struggled with the argument enumeration step in the wizard, evident from his "Poor" rating of the component. It is therefore likely that a bad experience with one step in the wizard leads to lower overall ratings for the entire component. 17% of test subjects rated the argument enumeration wizard as "Poor". Since this wizard was designed specifically with parameter sweep applications in mind and attempts to mimic a UNIX-like command line, it is likely that test subjects were unable to identify with this metaphor. This is evident from the

large number of test subjects attempting to enter the arguments manually in the textboxes as discussed at the beginning of this section. A lack of Linux or command line scripting knowldge could also be the reason for the "Poor" rating.

Discussion

Even with the low component ratings mentioned so far, the statistical analysis presented in Table 7.10 shows that an overall rating of "Very Good" was obtained. This suggests that the Job Submission wizard is intuitive to use, even in the face of a new type of component that most test subjects would never have encountered before, namely the argument enumeration wizard. The response time and data presentation were all highly rated, providing further evidence in support of this claim.

As with the previous task, test subjects were asked to comment on the interface and provide suggestions as to where the component could be improved. As expected, 46% of test subjects suggested improving the file browser to allow for highlighting of files when clicked. As mentioned at the beginning of this section, many test subjects got confused when file names were not highlighted and needed reassurance that the correct file was selected. 13% of test subjects indicated that strategic placement of tooltips would help to prevent confusion in the face of many available options. The filtering wizard (see Figure 5.7) would be an ideal candidate for the addition of tooltips, for example. Finally, a few subjects indicated that improvements to the argument type names would help prevent confusion between the different file types.

Apart from the obvious usability problems highlighted by test subjects, possible future enhancements were also pointed out. Features such as a final summary of the job before it is created, automatic addition of spaces between arguments, allowing for manual argument entry and improvements to current buttons were noted. Finally, some test subjects thought it would be a good idea to make the progress icons at the bottom of the job creation wizard clickable. This would allow these buttons to be used for navigation and also for quickly moving among the various steps in the wizard.

As with the previous task, the majority of the comments are restricted to component improvements. Since no test subjects mentioned any drastic changes to the design of the interface, as well as the high ratings the component obtained, it can be concluded that the design was reasonably successful.

Task 3 : Job Monitoring

Sub-task 1 : Monitoring

The final task in the usage scenario, depicted in Figure 7.1, is the monitoring of the job submitted to the Grid. Test subjects were asked to locate the job status component and then record information present in the component. After the status was recorded, they were to refresh the component and record any further status changes. Before each refresh, the contents of the database was updated by the evaluator (recall the Wizard of Oz approach discussed earlier). Once test subjects were satisfied that the system provided a clear indication that the job had completed with no errors, the task was deemed complete. If errors were encountered, subjects were instructed to remove the offending sub-job(s). As with the previous task, no explicit results were expected from test subjects other than the successful completion of the task. However, observational data was once again gathered.

All test subjects were able to successfully locate the job status component. However, some

subjects took some time to realise that jobs in the status window were collapsed by default and, as with the Grid status task, did not see the expansion icon (+). Furthermore, since the job monitoring component makes use of two job deletion mechanisms— one to delete an entire job and one to delete sub-jobs— test subjects got these two confused. Test subjects attempting to remove a failed sub-job made use of the complete job removal feature, thereby deleting their entire job. This was the biggest problem noted during the completion of this task, but can easily be fixed by way of tooltips and help functionality, or a restructuring of the deletion components.

Component Evaluation

Upon completion of the job monitoring task, test subjects were once again instructed to rate the task. Tables 7.11 and 7.12 provide a summary and descriptive statistics of test subject responses respectively.

Rating of	**Responses**				
	Excellent	**Very Good**	**Neutral**	**Poor**	**Very Poor**
Intuitiveness	5	12	7	0	0
Response Time	2	13	6	3	0
Sensible Data Presentation	7	16	1	0	0

Table 7.11: Job monitoring component ratings; n = 24

Statistical Analysis	**Descriptive Statistics**			
	Median	**Minimum**	**Maximum**	**Std_dev**
Intuitiveness	4	3	5	0.70
Response Time	4	2	5	0.83
Sensible Data Presentation	4	3	5	0.53

Table 7.12: Descriptive statistics of Job monitoring component ratings; n = 24

As can be seen from the tables, the results are similar to those of the previous two tasks. Since the median is categorised as "Very Good", there is evidence to support the claim that the Grid monitoring component is intuitive and has a sensible data layout. The response time scores, when looking specifically at the standard deviation, are not as compelling. Since the component takes a long time to refresh, test subject responses reflected this.

Discussion

As mentioned in the previous section, one of the main problems with the job monitoring interface was its slow response time when being refreshed. Since test subjects were Computer Scientists, many mentioned some possible solutions to this. The first solution is to have the component automatically refresh as the status of jobs change. It was also suggested that this idea be taken further by refreshing only the jobs that have updated. Although this is by far a more complex solution when compared to the current implementation, it will have the positive effect of decreasing overall bandwidth as well as improving the responsiveness of the interface upon such a refresh. Since the slow speed has to do mainly with the choice of toolkit widgets, namely a grid component, choosing a component with less overhead also could potentially in-

crease the refresh speed.

Deleting jobs was another major problem that was identified with this component. The interface allows for two ways of removing jobs. The first method allows a user to manually select sub-jobs and delete them by making use of the "Apply Changes / Refresh" button (see Figure 5.14). The second method allows for the deletion of an entire job by specifying the job number (see Figure 5.14). As already mentioned, test subjects found this mechanism ambiguous and it was often observed that the latter method was used to attempt to delete individual sub-jobs. For this reason an alternative unambiguous deletion mechanism needs to be instituted.

Many of the other suggestions by test subjects were made to improve the usability of the component. Many test subjects noted that providing an overall status of a job as opposed to the status of all sub-jobs within a job would be more useful, decrease refresh time and also make better use of screen space. Furthermore, it also was suggested that the interface display notifications on the status of the job in the right-hand panel. If a job is started or cancelled, for example, such a notification could inform a user without the user having to make use of the job monitoring component. The job status component would therefore only be used for the purpose of troubleshooting and retrieving more detailed job information. Further suggestions include the use of colour as opposed to merely textual information to display the status of a job and listing jobs in error at the top of the list.

Overall Results

The above sections have presented an overview of the results from the user evaluations of each of the interface components in isolation. This final section aims to tie together the results from the previous sections as well as present the final thoughts of test subjects with respect to the interface as a whole. This section will start by evaluating overall intuitiveness, response time and sensible data presentation for the system as a whole. Finally, an overview of the entire interface from the perspective of test subjects, in the form of general feedback, will be presented.

Usability

In order to determine how the interface performed overall in terms of both usability and performance, the average result for intuitiveness, response time and sensible data presentation across all three tasks was calculated. The descriptive statistics for these averages can be found in Table 7.13. These results also can be seen in the Box and Whisker plot displayed in Figure 7.2.

Statistical Analysis	**Descriptive Statistics**			
	Median	**Minimum**	**Maximum**	**Std_dev**
Intuitiveness	3.66	2.33	4.66	0.51
Response Time	3.83	2.33	4.66	0.67
Sensible Data Presentation	4.0	3.0	4.66	0.46

Table 7.13: Descriptive statistics for average over all tasks; n = 24

As can be seen from the plot as well as the statistics, the standard deviation for all three metrics is reasonably low (less than 1). This indicates that there was little variation in the responses from test subjects. Furthermore, as can be seen from the plot in Figure 7.2, there also is little variation among the three independent variables across all three tasks. The high ratings provide strong evidence to suggest that the interface has an above average degree of usability and response time. Furthermore, the ratings suggest that the way in which data is presented in the

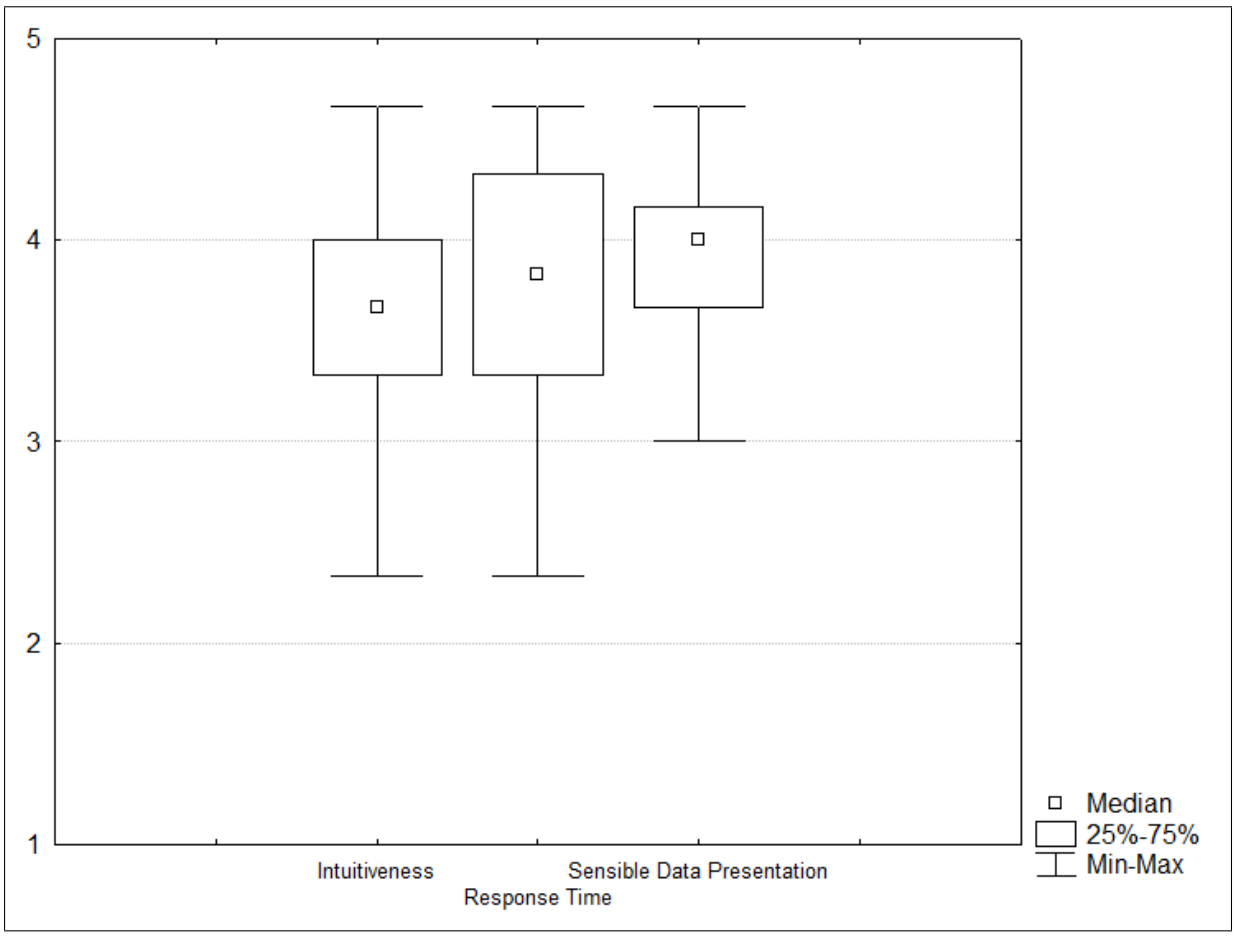

Figure 7.2: Box and Whisker plots of intuitiveness, response time and data presentation for the entire Grid interface

interface is sensible and easy to interpret, thus providing further evidence to support the high usability ratings presented in the previous sections.

General Feedback

Once all three tasks were completed, test subjects were asked to answer a few questions on issues such as aesthetics, features they did and did not like, overall responsiveness, as well as overall comments and suggestions. The results from this section are presented in Table 7.14. Note that in this case, n = 22, as two test subjects neglected to answer the general feedback questions. As can be seen from the statistical results presented in Table 7.15, test subjects re-

Rating of	**Responses**				
	Excellent	**Very Good**	**Neutral**	**Poor**	**Very Poor**
Overall Responsiveness	1	14	4	3	0
Level of dynamic functionality	4	10	6	1	1

Table 7.14: Overall interface ratings; n = 22

ported above average results for the overall responsiveness of the interface. The result reported for overall responsiveness is in line with the average response times over all three tasks reported

Statistical Analysis	Descriptive Statistics			
	Median	**Minimum**	**Maximum**	**Std_dev**
Overall Responsiveness	4.0	2.0	5.0	0.789542
Level of dynamic functionality	4.0	1.0	5.0	0.984732

Table 7.15: Descriptive statistics overall interface ratings; n = 22

in Table 7.13. This result therefore strengthens the analysis presented in the previous section and provides further evidence to support the claim that the overall responsiveness of the interface was deemed to be more than adequate by test subjects.

Recall from the test subject background analysis that 96% of test subjects had heard of the AJAX Web development paradigm. This result is important at this point in the analysis in order to determine if the interface successfully made use of AJAX principles in order to present users with a more intuitive interface. From the results presented in Table 7.15, it can be seen that test subjects deemed the interface to be highly dynamic. Since test subjects were able to observe the dynamic elements of the interface in action while using the interface to complete the tasks set out in the questionnaire, this rating provides evidence to suggest that the use of AJAX was well received by test subjects.

In addition to the Likert scale questions, test subjects were asked to provide general suggestions and comments about the interface as a whole. The questions in this section were split up into a few groups, namely: general aesthetics; improvements; and likes and dislikes of interface components. The results from each of these groups will now be discussed in more detail.

General Aesthetics

The general consensus on the part of test subjects was that the interface is clean, easy to understand due to its structure and, to quote a user, "very attractive and appealing, clear menu, workspace and areas familiar to even a new user". Other comments made by test subjects were that the interface fits in well with that of the Web browser's own interface and that effects such as those present on tabs make the system appear professional and "finished off". On the negative side, some test subjects noted that since the interface is a very specialised tool, novices might have trouble understanding it due to it being somewhat technical. Since the interface is a scientific tool that is not intended to be used by people without some basic level of HPC knowledge, the presence of technical information is unavoidable. What has been determined by test subjects, however, is that the layout and presentation of this technical information has been done in an intuitive way. Finally, one test subject also noted that the size of the font could be increased. Since the interface requires that a large amount of information be displayed, the font size was decreased in order to allow for more efficient use of screen space. This tradeoff cannot easily be overcome without impacting the usability and sensibility of the data displayed on the interface.

Improvements

As mentioned in the previous section, the interface was well received by test subjects. However, some possible improvements were brought to light. These were mostly of a minor nature. The main concerns had to do with notifications and alerts. Test subjects noted that the interface should dynamically alert users to job status changes by displaying updates in the right-hand

pane. During the design phase of this research project, this pane was envisioned to perform this function, but this feature was not implemented due to time constraints. The next major improvement suggested by many test subjects was the use of documentation to assist with the use and understanding of the functionality present in the various components. Furthermore, the prolific use of tooltips and legends to make the understanding of icons and buttons clearer also was suggested. Other minor suggestions included improving the visibility of links so as to be able to tell them apart from static text such as labels.

Likes and Dislikes

As already mentioned, the most appealing aspects of the interface to test subjects are that it has a clean look, is dynamic and responsive. Furthermore, test subjects mentioned that the data presentation was clear, again providing evidence in support of the findings presented earlier (see Figure 7.2).

In terms of dislikes, one of the most commonly mentioned problems with the interface is that it did not have enough help functionality. As mentioned earlier, tooltips and more general documentation on the use of each component would make the interface much more usable. Similar comments were made about the icons used in the interface. Without appropriate tooltips it can become difficult to make out what each one does. Comments made during this section were focused more on the improvement of the interface than actual problems.

7.3 Performance Evaluation

The user evaluations have only provided evidence to support the hypothesis that the Web interface is intuitive and usable as a tool for submission and monitoring of parameter sweep Grid applications. Although such an interface can be shown to be easy and intuitive to use, it can also perform poorly in terms of perceived speed, bandwidth and latency when compared to other design paradigms such as the traditional Web development paradigm. It is for this reason that evaluation of performance was conducted.

A common selling point of the AJAX-based approach as an alternative development technique is that it is more bandwidth efficient. The claim is that by reducing the amount of traffic attributed to traditional full page refreshes, as well as a perceived speed increase due to its dynamic loading properties, the AJAX approach is superior. One cannot, however, simply take such claims on face value. In practice, many different toolkits are used for the development of AJAX-enabled Web applications. Since a toolkit could be poorly designed or make use of inefficient algorithms, it is possible that the full potential of an AJAX approach would not be realized. For these reasons, a study was conducted to measure the performance of the Web application developed as part of this research, primarily in terms of bandwidth efficiency and latency. Such performance evaluations are generally used to show how one system compares to another in terms of some pre-defined set of criteria. In order to evaluate the Web interface, this was not possible. Since only one version of the interface exists, i.e the AJAX version, there is no suitable candidate to which to compare the interface. Furthermore, building a duplicate system using a traditional development approach was considered wasteful, therefore prompting the use of the simulated analysis technique. Such an analysis simulates how the AJAX interface would operate if the traditional development approach was used instead.

This section will discuss the techniques used to quantitatively evaluate the performance of the AJAX Web interface as well as a simulated analysis of a Web interface developed using a traditional approach. Finally, the results obtained from these evaluations will be discussed.

7.3.1 Methodology

Before any performance evaluations could begin, it was necessary to outline a usage scenario for a typical Grid interface. By returning to the initial set of requirements outlined by the scientists interviewed at the beginning of this research project (see Section 4.2.1) a Grid job submission and monitoring scenario was decided upon (see Figure 7.1). This scenario consists of three tasks, namely: Grid status examination; job submission; and finally job monitoring. Each of these tasks was then broken down even further into sub-tasks which are numbered accordingly. The usage scenario depicted in Figure 7.1 is identical to that used during the user evaluations discussed in the previous section.

Once the usage scenario was finalised, each sub-task was evaluated in terms of its constituent data components. To do this, the Firebug [Lerner, 2007] tool was used. Firebug is a Mozilla Firefox [Hipson, 2005] plugin that provides a wealth of information as a page is loading, by breaking up the data communication into its constituent components. Figure 7.3 shows a typical Firebug display as a Web application is loaded into the browser. The page elements that are loaded also are visible as well as the times taken to load each individual element. As can be seen from the toolbar present at the top of Figure 7.3, Firebug provides data on the size of each of the HTML, CSS, JavaScript (JS), AJAX (XHR), image and Flash components of a client-server communication. Each of these components was recorded separately for each of the sub-tasks.

For the AJAX interface, obtaining this information was as trivial as loading the Web appli-

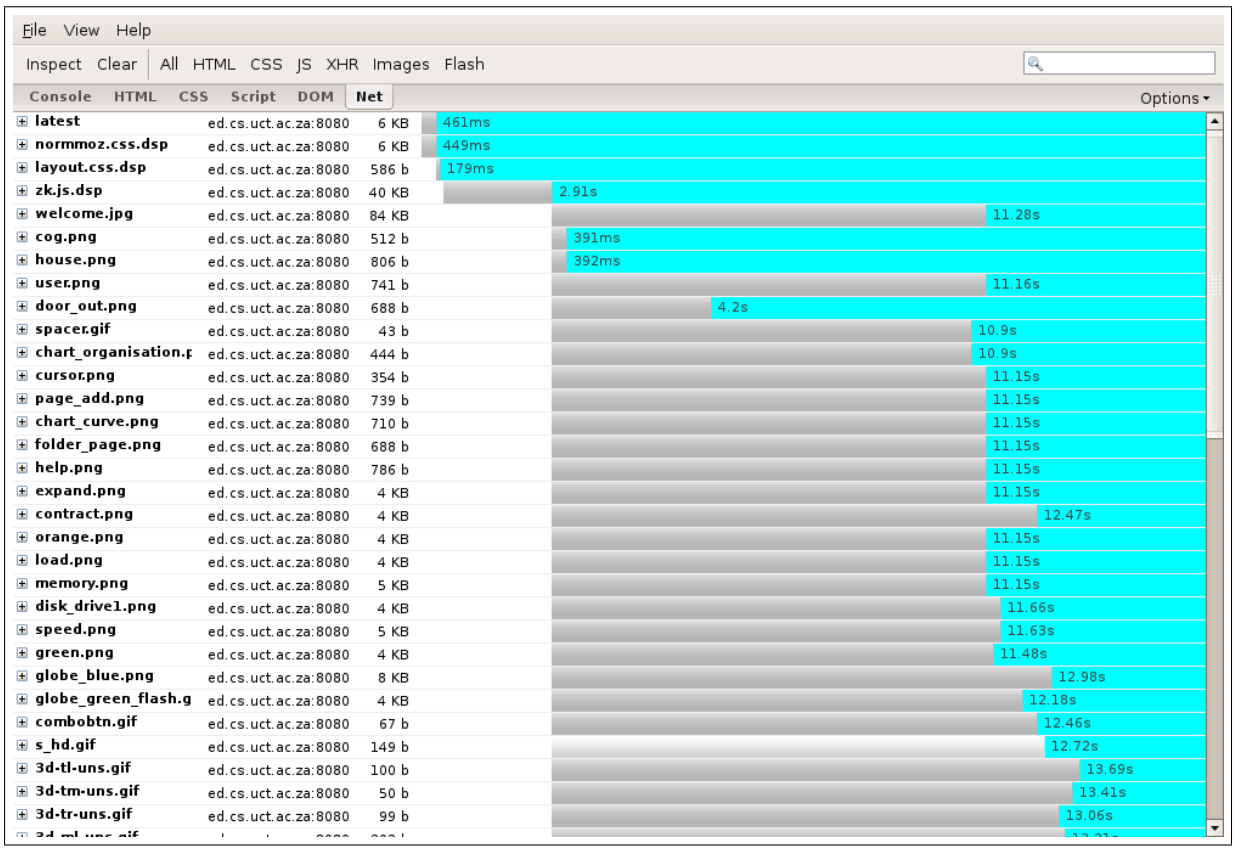

Figure 7.3: Firebug in action during interface loading

cation and recording the values present for each of the components displayed by Firebug. For the simulated analysis, however, it was necessary to make a number of assumptions in order to generate data based on a traditional Web application development model. It was therefore assumed that had the interface been HTML-only:

1. the menu pane would be located in a separate frame on the left side of the interface and would therefore not be loaded each time a page is updated
2. no JavaScript or dynamic components would be present
3. the initial CSS file would be served only once and would then be resident in the browser's cache
4. as with the AJAX interface, each unique image would be taken into account only once and would then be retrieved from cache
5. unlike the AJAX interface, no overlapping windows would be used and therefore the main display area would be updated after each task
6. certain elements such as the argument selection popups would be opened in new windows thereby reducing page load times

With the above assumptions in place, the same Firebug-based evaluation procedure was followed as with the AJAX interface. The data generated by Firebug was then used to build a model of a traditional Web interface by excluding extraneous parts and considering only those parts that would contribute to the particular task being displayed. In other words, all dynamic content was removed, leaving only data attributed to the static content for each sub-task. The size of these filtered pages was then recorded instead of those reported by Firebug.

	Requests	HTML	CSS	JavaScript	XHR	Images	Total
1	67 / -1	42.2 / 28.9	6 / 6	64 / 0	0 / 0	137 / 137	211 / 171.9
2	2 / -1	0 / 22.4	0 / 0	0 / 0	2 / 0	0.679 / 5.74	2.679 / 27.98
3	2 / -1	0 / 13.21	0 / 0	0 / 0	2 / 0	0.679 / 0.69	2.679 / 13.89
4	21 / -1	0 /13.29	0 / 0	7 / 0	2 / 0	286 / 34	293 / 47.29
5	1 / -1	0 / 13.75	0 / 0	0 / 0	0.2 / 0	0 / 0	0.2 / 13.75
6	3 / -1	0 / 26.84	0 / 0	0 / 0	1 / 0	0 / 43	1 / 69.84
7	1 / -1	0 / 12.94	0 / 0	0 / 0	0.71 / 0	0 / 37	0.71 / 47.94
8	12 / -1	0 / 21.99	0 / 0	0 / 0	5 / 0	9 / 11.735	14 / 33.725
9	1 / -1	0 / 39.39	0 / 0	0 / 0	0.144 / 0	0 / 40.733	0.144 / 80.123
10	3 / -1	0 / 21.99	0 / 0	0 / 0	4 / 0	0 / 0	4 / 21.99
11	1 / -1	0 / 49.34	0 / 0	0 / 0	0.143 / 0	0 / 40.618	0.143 / 89.96
12	6 / -1	0 / 27.14	0 / 0	0 / 0	4 / 0	0.303 / 0	4.303 / 27.14
13	3 / -1	0 / 28.55	0 / 0	0 / 0	3 / 0	0 / 0.678	3 / 29.23
14	7 / -1	0 / 29.96	0 / 0	0 / 0	7 / 0	0 / 0	7 / 29.56
15	2 / -1	0 / 12.94	0 / 0	0 / 0	0.282 / 0	0 / 40	0.282 / 52.94
16	3 / -1	0 / 12.95	0 / 0	0 / 0	1.09 / 0	0 / 0	1.09 / 12.95
17	38 / -1	0 / 20.23	0 / 0	0 / 0	13 / 0	0 / 4	13 / 20.23
18	12 / -1	0 / 43.54	0 / 0	10 / 0	18 / 0	4 / 4	32 / 47.54
19	2 / -1	0 / 50.54	0 / 0	0 / 0	0.229 / 0	0.064 / 0.064	0.293 / 50.54
20	1 / -1	0 / 43.54	0 / 0	0 / 0	18 / 0	0 / 0	18 / 43.54

Table 7.16: Performance data for the AJAX and traditional interfaces.

7.3.2 Results

The data generated during the Firebug-based user evaluations is shown in Table 7.16. The values on the left side of the "/" are associated with the AJAX interface, whereas the values on the right refer to the hypothetical simulated HTML-only interface. Since the simulated interface, as per the assumptions, has no dynamic components, the JavaScript and XHR columns have zero values. Similarly, there is no data associated with the requests column for the simulated analysis due to such data being unavailable, hence the "-1" values.

To make sense of the data present in Table 7.16, Figure 7.4 shows a cumulative bandwidth plot for the interface usage scenario for both the AJAX and simulated interfaces. From the graph it is clear that the AJAX interface rapidly begins to outperform the simulated interface. The reason for this is that the AJAX interface does not reload data that doesn't need to be updated. The XHR calls, together with JavaScript's capability of manipulating the DOM tree in the browser dynamically, reduces the overall bandwidth consumption of the Web application. Therefore, by the time the user is three quarters of the way through a job creation procedure (sub-task 10), the interface becomes more efficient than its HTML-only counterpart. One thing to note in the figure, however, is the peak at sub-task 4. Since the way in which the AJAX application was developed causes the job creation wizard to be downloaded in its entirety when the wizard is activated, a large peak can been seen at this point in the graph. The reason for choosing such an approach is to allow for an uninterrupted job creation procedure, thereby preventing a user from losing focus on the task at hand while creating his/her job.

Since the bandwidth reduction techniques of the AJAX-based development approach are compounded over time, Figure 7.5 shows the effect of a longer usage session on overall bandwidth usage. The graph shows four consecutive iterations of the usage scenario described above. The vertical lines in the graph indicate the beginning of a new iteration. As can be seen from the

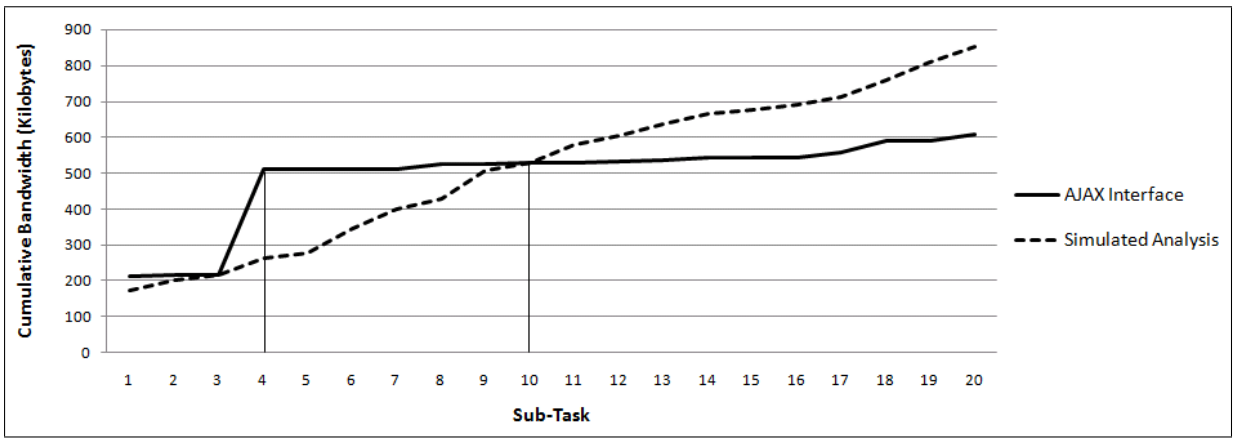

Figure 7.4: Cumulative bandwidth usage of the AJAX interface vs. an identical hypothetical non-dynamic HTML-only interface for the interface usage scenario

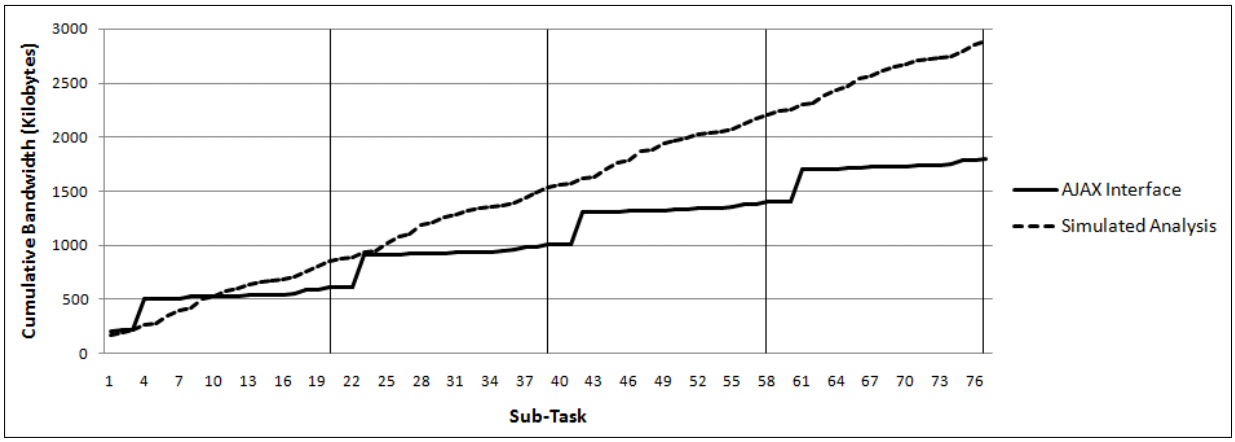

Figure 7.5: Cumulative bandwidth usage of the AJAX interface vs. an identical hypothetical non-dynamic HTML-only interface for four consecutive repetitions of the interface usage scenario

graph, the bandwidth usage over a longer period of time is greatly reduced by using an AJAX-based approach. In the graph, images and CSS were assumed to be cached for each successive iteration of the usage scenario.

The graphs so far have shown the benefits of the AJAX-based approach in terms of bandwidth usage. However, with lower bandwidth consumption comes decreased latency, especially over slower network connections. Figure 7.6 shows the non-cumulative bandwidth consumption of the usage scenario outlined above. From the figure it is clear that by using the AJAX approach, a fully loaded interface (as from step 5 onwards) shows a significant decrease in data transfer from the server to the browser. This decreases latency and provides a pleasanter user experience. It is, however, the case that AJAX makes a larger number of requests than traditional techniques and could therefore lead to an increase in latency (see Table 7.16). To offset such large numbers of requests, the way in which the Web application is designed can overcome this problem. By ensuring that the bulk of requests are made before a user starts interacting with the application, i.e. as the page is loading, latency issues are not as noticeable.

The results presented in this section show that the AJAX-based approach to Web application development has significant advantages in terms of decreased overall bandwidth usage and latency. The results show that an AJAX-based approach, although seemingly more heavyweight in terms of its dynamic nature and underlying JavaScript core, is in fact more lightweight than

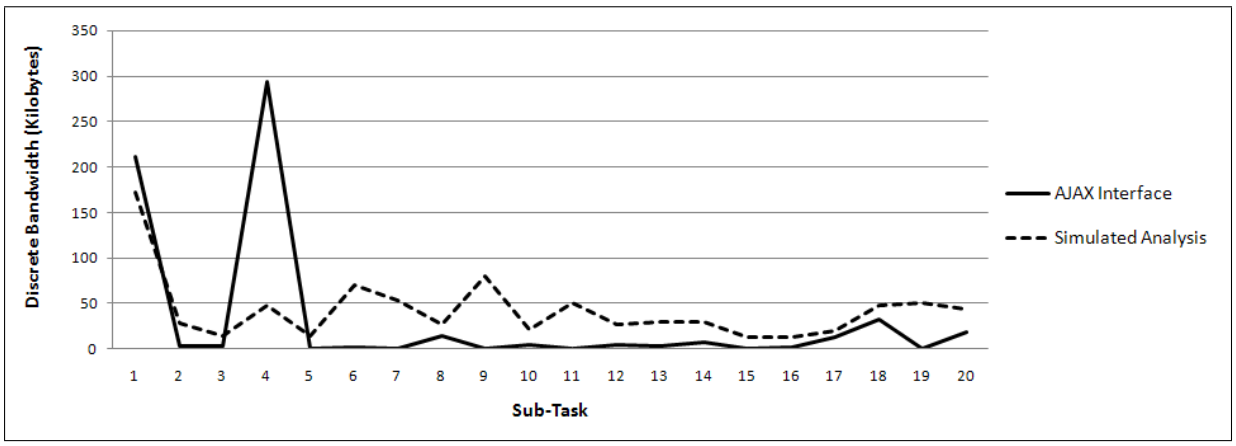

Figure 7.6: Discrete bandwidth consumption of the AJAX interface vs. an identical hypothetical non-dynamic interface for the interface usage scenario

traditional development paradigms.

7.4 Summary

This chapter has presented two methods used to evaluate the interface designed as part of this research. The first method— the user evaluations— aimed to gauge user responses towards the interface in terms of usability, response time and data presentation. Test subjects were asked to complete a series of tasks and then provide feedback by filling out a questionnaire. The results from these evaluations were positive and provide evidence to suggest that the initial objectives set out in Chapter 1 were met. Test subjects gave above average ratings for usability, sensible data presentation as well as response times for each of the tasks in isolation and similar responses for the interface as a whole indicating a good degree of consistency in the results.

The second method— the performance evaluations— were used to determine how well the interface responded to user requests by making use of a typical usage scenario. This scenario took the form of a job submission and query cycle and was designed to provide results indicative of real-world usage. The results of these tests were then compared to a simulated analysis of an HTML-only interface. The results from these experiments provide conclusive results which prove that the AJAX-based interface is more efficient than an HTML-only interface during the course of a typical usage scenario, both in terms of response time and bandwidth efficiency. The lightweight objective set out in Chapter 1 has therefore been achieved.

Chapter 8

Conclusion

The usability of scientific software has often fallen by the wayside as the design and implementation of new features is considered to be a more important objective. The lack of usability of such systems therefore hampers their uptake, somewhat ironically, due to an abundance of features making the system as a whole difficult to use. This was and currently still is the case with Grid middleware. For this reason, this research project has investigated how Grid technology can benefit from lightweight Web technologies in order to abstract away the complexities inherent in a Grid by providing users with access to a high-level interface to such systems. In order to achieve this goal, a Web interface was researched, designed and built using a lightweight AJAX-based approach. A number of experiments— case studies and user evaluations— were then conducted in order to determine whether the initial objectives of the research project were met.

This chapter will provide an overview of the findings from all experiments and evaluations and draw together the steps taken during the research project to paint an overall picture of the research as a whole. Firstly, however, the objectives and the way in which they were met will be discussed.

8.1 Realisation of Objectives

8.1.1 Design and implement a Web interface to a local Grid computing system with a high degree of usability

Designing and building an interface is a trivial task. However, building an interface that can easily be used by novices is non-trivial. For this reason, a process of interviews, paper-prototype construction and user evaluations of these prototypes was conducted. Only after the prototypes were evaluated and the appropriate modifications made to the design, was a prototype of the actual interface developed as software. This design lifecycle was decided upon in order to maximise user input during the design process.

Once the final system was developed, a series of prototype evaluations was once again conducted by making use of the final system. The evaluations consisted of three tasks each with its own sub-tasks. After completion of each task (with or without assistance), users were asked to score the interface according to three pre-defined metrics, namely: intuitiveness, response time and sensible data presentation. Furthermore, at the end of the evaluation session, users were asked to score the interface as a whole according to these same metrics. The aggregate results across all three tasks was found to be consistent with the scores for the interface as a whole, thereby strengthening the results generated from the study. These results, reported in Chapter 7, provide evidence to suggest that the interface has a high degree of usability.

8.1.2 Ensure that computer literate non-Grid experts are able to make use of the interface

A crucial part of the success of the interface was to enable non-Grid experts to easily make use of it without having seen a similar interface before. Since more than half of the test subjects had little to no knowledge of HPC, the high usability result reported by users, discussed in the previous section, provides evidence to suggest that the use of the interface is not limited to Grid experts only. In addition, the high data presentation ratings of the system as well as the clean graphical nature of the interface appealed to test subjects, accounting for these above average scores.

8.1.3 Implement the interface in such a way that it is lightweight and makes use of Web 2.0 techniques

In order to construct an interface that attempts to look and feel like a desktop application, Web 2.0 techniques were utilised to satisfy this requirement. In order to meet the high usability objective discussed so far, the interface was designed to look like, act and respond like popular Web applications such as Facebook, Flickr and Gmail.

In order to achieve the Web 2.0 look and feel as well as meet the objective of building a lightweight system, an AJAX toolkit was utilised as the core of the system. An AJAX approach to Web development allows for asynchronous communication between the browser and the server in order to update only relevant parts of a Web application dynamically. Such an approach, if implemented correctly, promises to increase the efficiency of the Web application, reduce server load and decrease request-response times by decreasing the overall bandwidth requirements of the application. Experiments and simulated analyses confirmed this premise and showed that the Grid interface built during this research project was, over the course of a typical Grid job submission and monitoring cycle, more efficient than an HTML-only interface, in terms of both response time and bandwidth usage. The Web 2.0 techniques, such as the use of AJAX, have shown that a lightweight interface can be built to an inherently dynamic Grid job submission and monitoring environment.

8.1.4 Ensure that the interface is extensible by allowing for inclusion of different schedulers as "plug-ins" by utilising a component-based approach

Grid computing and scheduling systems, although different, are primarily based on the same technology. In other words, the process of submitting, monitoring and viewing the status of a Grid system is identical across most scheduling systems. However, each implementation and the way in which each of these tasks is achieved is different. For this reason, some vendors (be they proprietary or open-source) have built portals and interfaces (Web-based and Desktop-based) for their particular system. Such an approach does provide a much higher level of usability to the systems in question. However, in real-world environments, many different schedulers managing different equipment can exist. For this reason, it was deemed important to develop the system in such a way that different scheduling systems can be "plugged-in" to the interface by writing a set of scripts. This goal was achieved during the development of this interface, allowing a user to change from one scheduling system to another by selecting the scheduler of choice from a drop-down list. This approach has the advantage of providing a central point of development for scheduling system-independent interface technology as well as providing users with a central point from which to submit all jobs, irrespective of the scheduler used.

8.2 Reflection

Before any of the research objectives could be researched, a test Grid had to be built on which to base the research. Subsequent to this, an interface to such a Grid system had to be designed and implemented without having much literature in the Grid domain on which to base decisions. Current Grid portals and interfaces are merely designed as a means to an end with, what seems to be, very little research conducted into creating an interface that is both intuitive and has a high level of usability. For this reason, the process of building such an interface was rather daunting, particularly since AJAX implementations also are fairly new and have only been used for the past few years. Bearing all this mind, however, the experience of conducting research where there were so many unknowns has been a positive one, from both an academic and practical point of view. This section will present various views of AJAX from both the development perspective and from a higher level as well as discuss some problems encountered during the course of this research.

8.2.1 Is AJAX development really different?

As discussed earlier in this dissertation, AJAX is comprised of various technologies, both old and new. In this regard, development of software using the AJAX paradigm is simply a term used to describe a Web application developed using these technologies. In this sense, AJAX development is no different to existing Web development techniques. In terms of technologies alone, the term AJAX can be considered to be a buzz word. However, the concept of an AJAX Web application goes further than just the core technologies upon which it relies. The term AJAX is synonymous with usability, high levels of user interaction, collaboration, lightweight applications, rich dynamic properties of such applications, efficiency, rich GUIs and much more. AJAX is therefore an end-user centric paradigm that is used to enhance and deliver content to the user in a dynamic, unobtrusive and aesthetic manner. The principles upon which the AJAX paradigm is built therefore makes its use in practice very different from existing development methodologies.

8.2.2 Will AJAX last?

The future of AJAX, as with any technology, is uncertain and can only be speculated on. However, as the Web continues to become increasingly popular as a platform for hosting complex applications with the need for enhancing usability, user centric paradigms such as AJAX will be a necessity. In this regard AJAX will certainly be a popular choice for development of Web applications in the future. The question as to whether AJAX will prevail over technologies such as Flash, for example, is however difficult to answer. Although both AJAX and technologies such as Flash aim to create rich user interfaces with high levels of interaction and dynamism, each technology has benefits for the context in which it is used. However, in terms of support, AJAX provides developers with the peace of mind that most modern day browsers support all the technologies making up the core of AJAX. The same cannot be said for Flash, for example, as Flash requires a third-party plugin to be installed before Flash-based applications will run in the browser.

8.2.3 General Project Problems and Issues

Throughout this research project there have been many problems encountered. These have been minor problems ranging from the recruitment of test subjects and small programming snags to major problems such as the complete failure of the primary research cluster. Many of the experiments and the Grid deployment discussed in this thesis made use of this cluster. Such unforeseen problems inevitably delay the research process, but provide valuable experience.

In terms of the development of the interface itself, the use of AJAX, and in particular the toolkit, provided a unique set of challenges. Firstly, hand-coded AJAX is extremely tedious to write due to its complexity and other factors such as cross-browser incompatibilities. For this reason it was decided to make use of the toolkit already mentioned. Since many of these toolkits exist, much research had to be conducted by reading articles, blogs and feature lists in order to make an informed decision on the choice of toolkit. Once the toolkit used in this research was chosen, the next step was exploring its features and use. Many challenges presented themselves in this part of the process. Firstly, the toolkit was developed in China and therefore documentation was not always clear. Secondly, the examples provided on the Web are mainly for beginners thereby requiring one to make use of forum posts in order to determine how more advanced features can be used. Thirdly, during development one realises the shortcomings of a toolkit that could not be anticipated before. All the afore-mentioned obstacles take time to overcome and thereby increase development time substantially. The conclusion drawn from developing in AJAX is that one must be careful when choosing a toolkit, especially since it is practically impossible to convert from one to the other without incurring a significant loss in time. Furthermore, the level of support provided by the developers of such a toolkit is of utmost importance. Making use of an outdated or poorly supported framework will almost certainly lead to problems during the development process. Lastly, one must consider the cost of utilising a toolkit as opposed to writing hand-coded AJAX. For smaller projects the use of a toolkit is almost certainly advantageous.

8.2.4 General Conclusions

This research project has shown that it is possible to display a large amount of technical HPC information in a limited space by making use of the dynamic features provided by the AJAX development paradigm. Of course, HPC is not the only scientific field that requires much data to be displayed on screen. The fields of high energy physics, climate modelling and GIS as well as many other such disciplines often require data to be summarised and displayed to users in an intuitive fashion. This research has shown, by way of an implementation of an actual system, the benefits that an AJAX-based Web application can have for use in these fields. Not only does an AJAX solution allow for information to be hidden and displayed dynamically, but its ability to update only specific page elements as desired makes it a sound development alternative to existing methodologies.

8.3 Future Work

This research project has much scope for future additions as well as completely new research topics altogether. An overview of each possible addition and research topic follows:

8.3.1 Interface improvements

Due to the prototypical nature of the interface there is still much room for improvement. One of the main areas of improvement lies with the argument enumeration step of the wizard. This component currently takes a directory of input files and uses these files to build a list of parameters for each run of a parameter sweep application, if so desired by the user. However, the wizard is incapable of recursively traversing directories in order to build runs off of child directories of the parent directory from which input files are selected. This would be a powerful feature of the interface as it is often the case that input files have the same file name, but can obviously not reside in the same directory, necessitating the use of the hierarchical directory structure. A further extension envisaged would be the ability of the interface to apply a pre-defined operation or set of operations upon a directory or directories of files. Such an operation, for example, to apply algorithm A, B, C and D to each of 100 files located in 50 subdirectories would then be possible. Currently, in order to achieve such an operation using the interface, one would have to create 50 jobs (1 for each sub-directory) and write a wrapper script to apply each algorithm to each file in turn. The amount of effort currently required to obtain output from the system for this type of job would most certainly dishearten users from using the interface.

8.3.2 Interface additions

Load Balancing

The current design of the Grid interface allows users to select the scheduler of choice from a drop-down menu and then proceed to submit and monitor jobs pertaining to that scheduler. However, as has been mentioned, an environment can consist of many different schedulers. A possible future extension to the interface would allow the system to automatically select the best scheduler for a particular job or have the system automatically submit a job to the scheduler with the maximum available free resources, provided the job can run on those resources.

Visualization

The Grid status component of the interface built as part of this research is relatively simplistic and serves to provide a general overview of the status of the pools or clusters of which the Grid is comprised. However, in order to make this component more visually appealing and intuitive to users, a visualization of the status of the Grid could be incorporated into the interface. This visualization could take the form of a 3D-tree view of the Grid itself or a novel view for the Grid could be created. Due to the dynamic properties of the Grid itself, a visualization for the Grid could potentially be modelled in a similar way to that of a peer-to-peer network, with the exception that nodes committed to the Grid are typically committed for a long period of time. Intelligent grouping of nodes making up one “cluster” or “pool” would be needed to ensure that users are able to select the approapriate co-located resources upon which their job is to execute.

Notification Panes and Feeds

Many features that were initially scheduled for integration into the interface were not implemented due to time constraints. Features such as dynamic notifications in the right-hand notification pane of the interface and RSS feeds, for example, were never implemented. Such features would make the interface more usable and also provide more of the Web 2.0 functionality that

was initially envisioned for the system— functionality that people have grown accustomed to on the Web.

Mobile Devices

It is often the case that one does not always have access to a PC when travelling or going out for the day. For these cases, a mobile version of the interface, either pre-loaded onto a capable handset or streamed via HTTP, would be useful for viewing the progress of jobs. This feature poses many interesting research questions such as how to display the complex Grid job data on devices with small displays and how the updating of such data will occur.

8.3.3 Additional Research

Grid Middleware Usability

This research project has abstracted away much of a complete Grid infrastructure by focusing usability research on local scheduling systems only. In doing so, the scope of the project was reduced and produced results which suggest that the full suite of Grid middleware could be made more usable, based on the findings from this abstracted view of a Grid. Future research could potentially investigate how usability of local scheduling systems differs from that of a complete Grid.

Browser Desktops

In the current state of Web technology, the browser is used as a mechanism for displaying mostly static text. Although some Web applications such as Gmail have succeeded in overcoming a reliance on Desktop email clients, there is still a long way to go in realising a fully browser-based desktop. Such a desktop would allow one to make use of word processors, email clients, photo-editing software, CD-burning software, etc, all from within the browser. Such applications could still leverage the power of a user's processor but run such applications in a dynamic Web-based environment. Such an environment would not suffer from the security problems that desktop computers currently face and would allow updates to occur instantaneously as soon as a new version of a program is released. As the future of computing becomes more closely intertwined with the Web, research into this area should start to become more widespread.

Appendix A

Paper Prototypes

Chapter 4 discusses the prototype evaluations that were conducted as part of this research. The aim of these evaluations was to gather user feedback on the Web interface, and in so doing, manipulate the design to satisfy a more general audience. Only interface components present in the final design (albeit in a different form) are included in this appendix.

As can be seen from Figure A.1, not much has changed from the initial prototype to the final implementation. Since the menu on the left of the interface was well received by users, this design feature was not altered. Users liked the simplicity and cleanness of the interface and it was therefore not altered in any way. Some of the components visible in the menu pane are no longer present in the final interface as these components were not implemented.

The main changes that were made after the prototype evaluations focused on the area of job submission. Figure A.2 shows some of the initial ideas for choosing a job type at the beginning of the submission process and thereby providing parameter sweep-like functionality in the interface. This idea was deemed too clumsy and was extended to the argument enumeration wizard (see Figure A.6) which will be discussed shortly. Furthermore, the job submission window (see Figure A.3) was modified to take the form of a wizard. This change streamlines the job submission process into discrete steps and does not leave the user feeling overwhelmed with a multitude of settings characterised by the initial design. The filtering window (see Figure A.4), a popup launched from the original submission window, also was modified to form part of the submission wizard found in the current interface. The panes of which the filtering window was comprised, however, are still clearly discernible.

One of the major modifications to the design of the interface, and catering exclusively for parameter sweep applications, was to the argument enumeration window (see Figure A.6). This window was initially designed to allow a user to build the arguments expected by the application from basic components. The components include different file types, flags and numerical values. This window too was integrated into the job submission wizard, but kept its look and feel.

Figure A.7 shows the initial design of the job query window. Not many interface changes were made, but the window was changed from a popup window to an embedded tabbed window during implementation. Finally, Figure A.8 shows a first attempt at a Web-based file management interface. The interface seen in the figure was not developed as the WebDAV approach discussed in Section 5.3.3 was favoured.

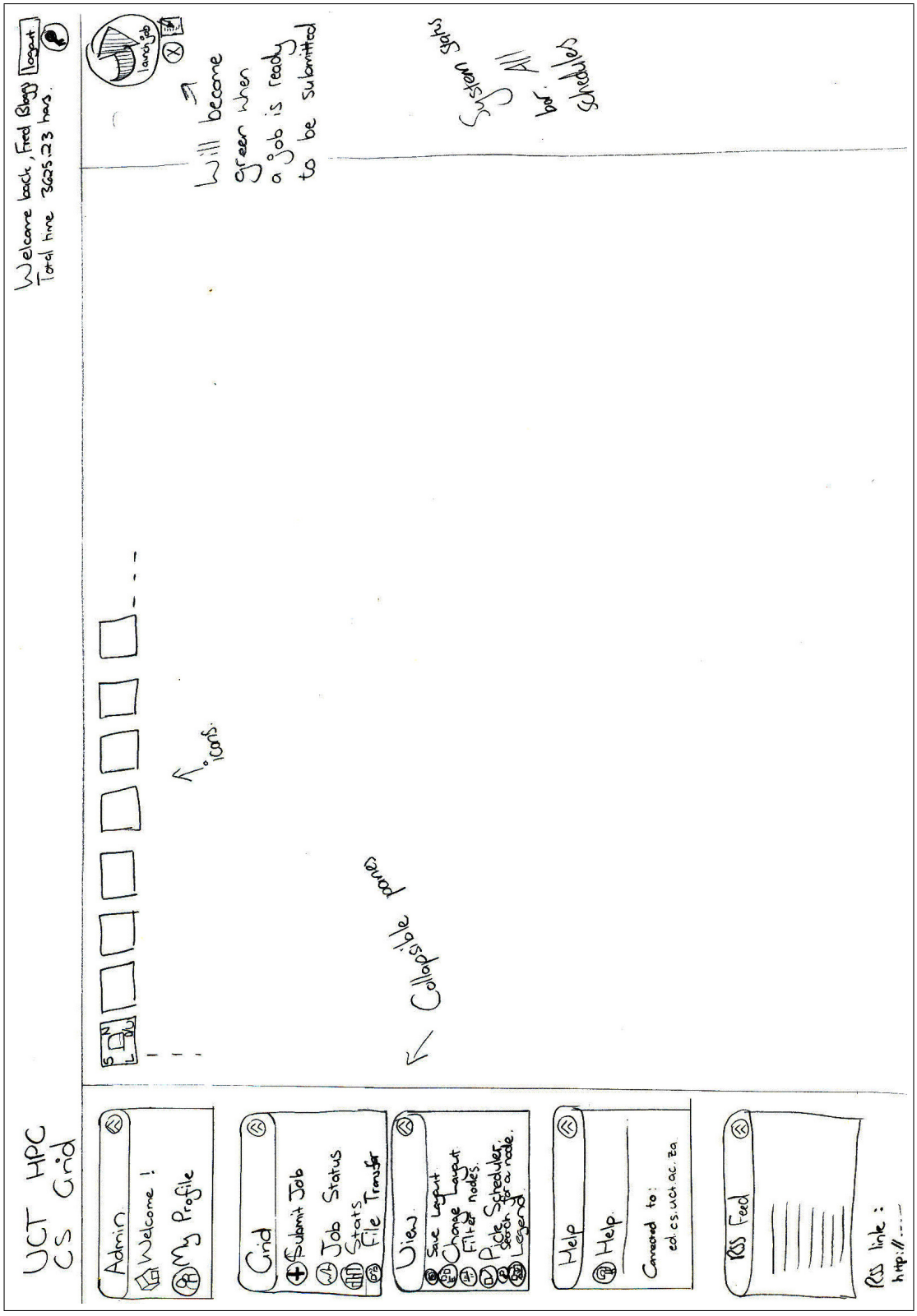

Figure A.1: Main Interface Layout

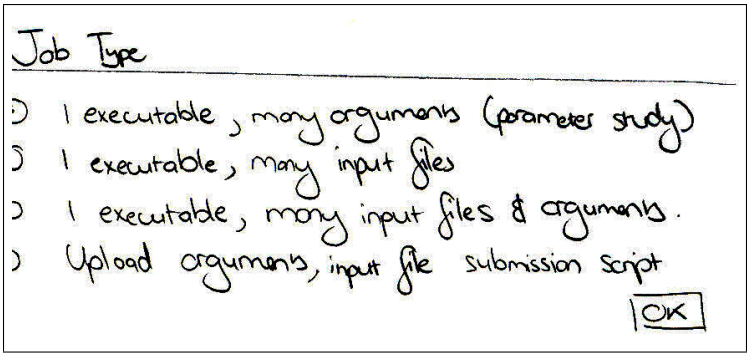

Figure A.2: Job Type Selection Menu

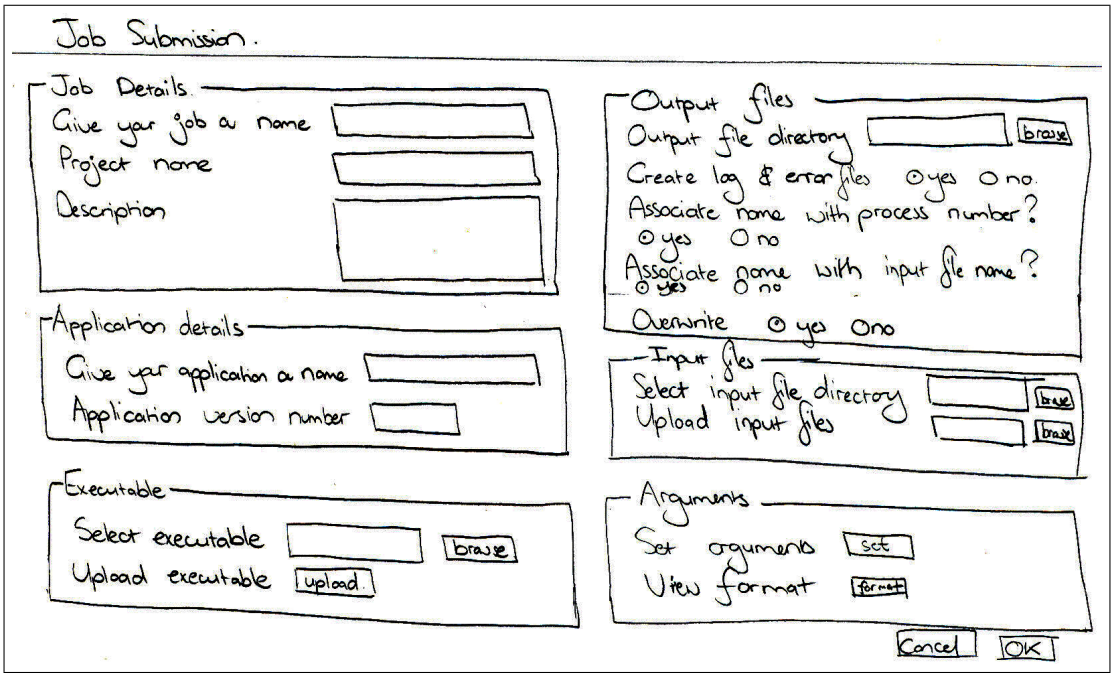

Figure A.3: Job Submission Wizard

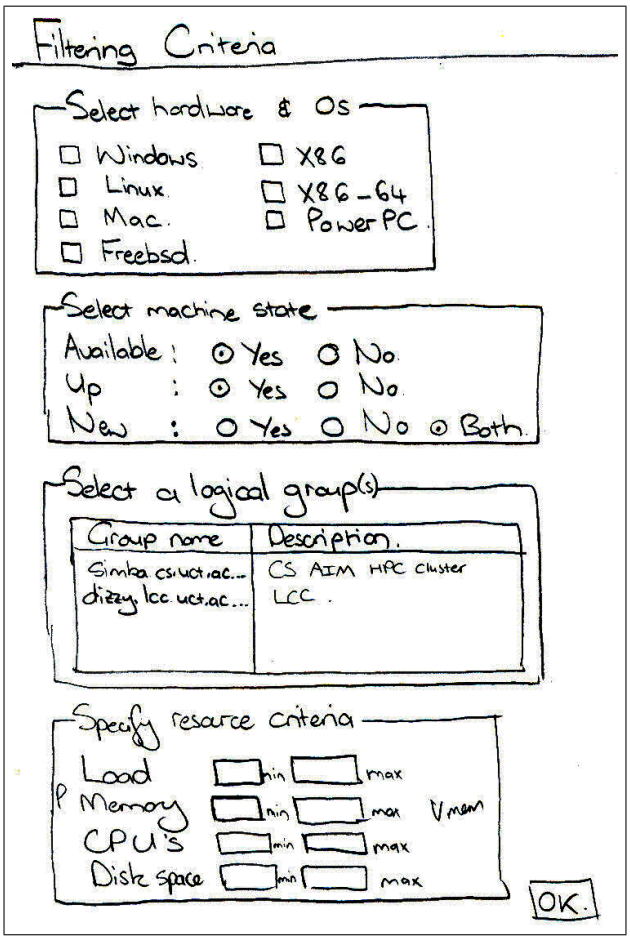

Figure A.4: Filtering Wizard

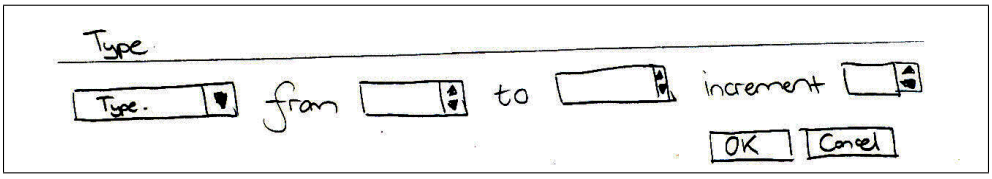

Figure A.5: Argument Settings Popup

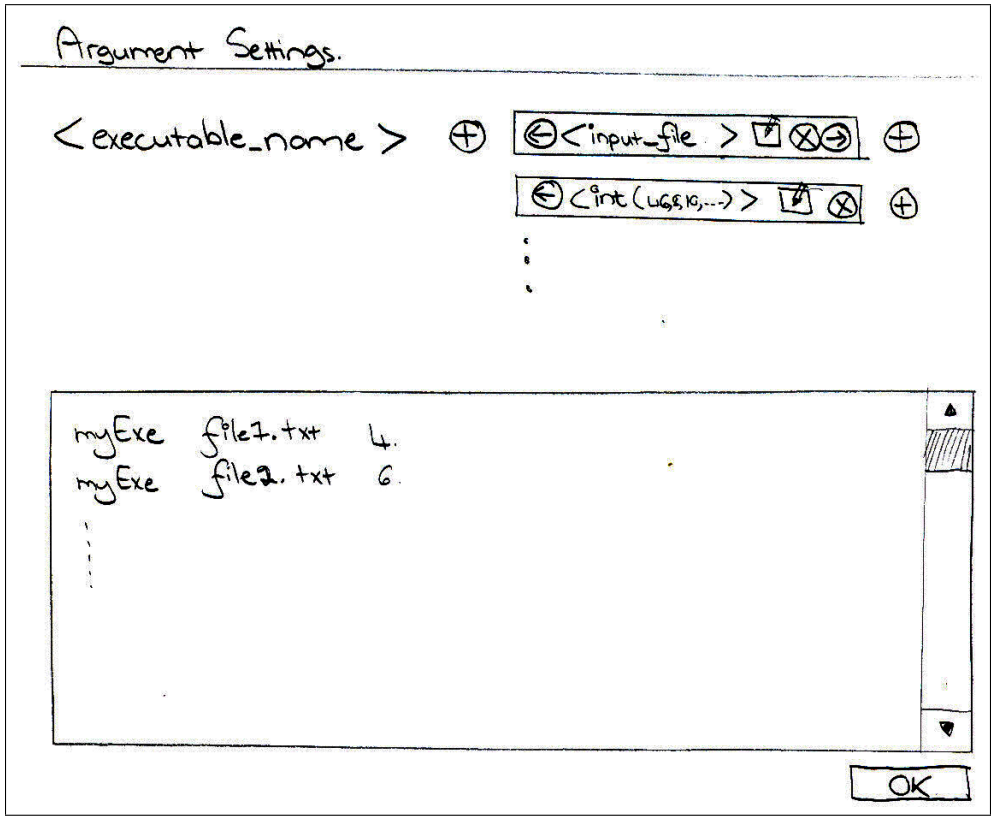

Figure A.6: Argument Enumeration Wizard

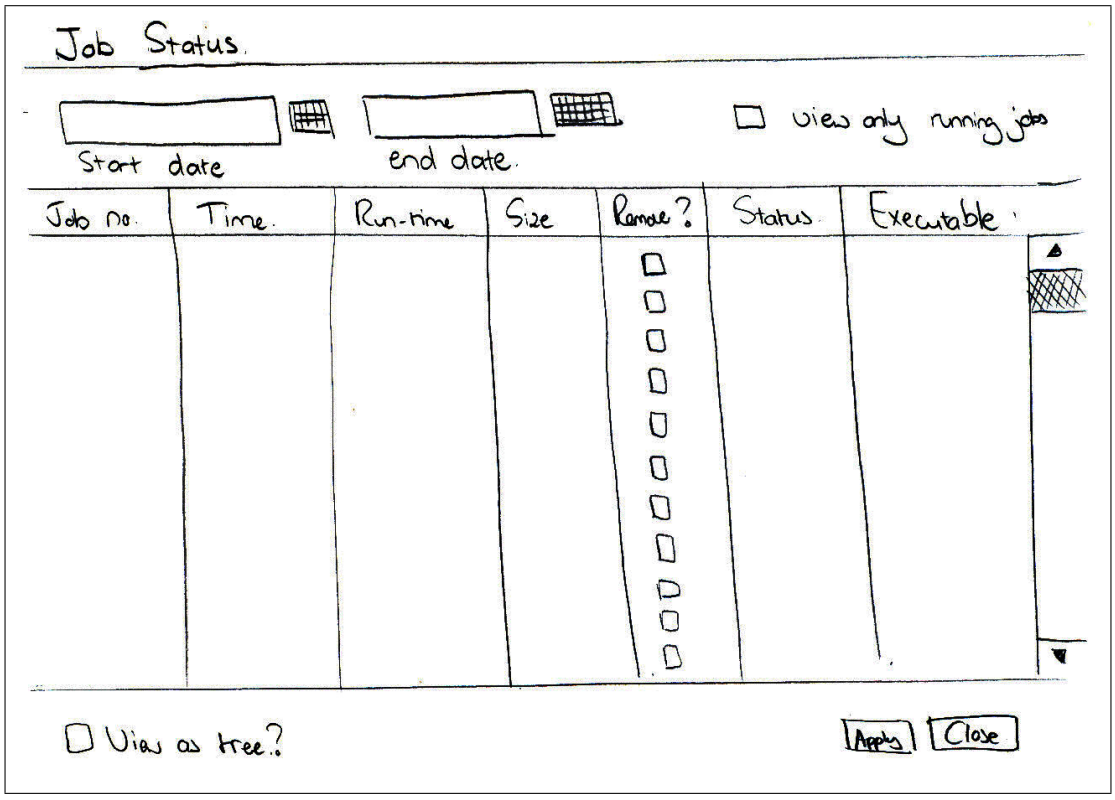

Figure A.7: Job Query Component

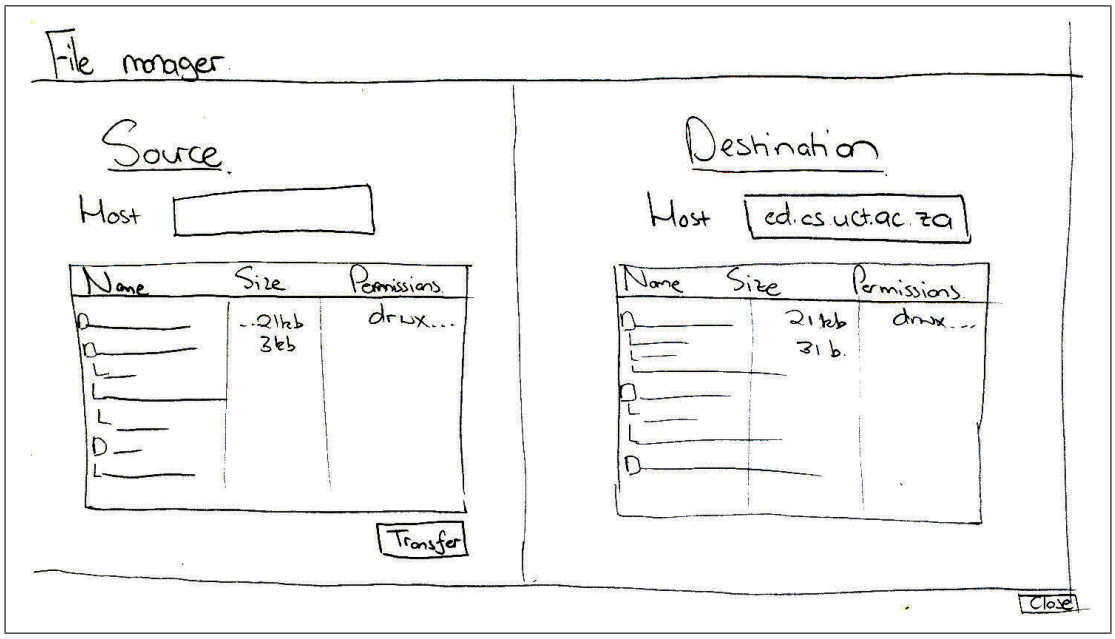

Figure A.8: File Manager

Appendix B

Interview Questionnaire

B.1 Knowledge assessment

- How familiar are you or your department with Grid computing technology?
- Briefly describe your, or your department's level of experience in either cluster computing, Grid computing or both.

B.2 Current parallel applications, tools and hardware

B.2.1 Applications

- What kind of parallel problems are you currently running?
- Do these problems parallelize well (i.e embarrassingly parallel) or are they communication intensive jobs?
- What language(s) is the source code for the above mentioned applications written in?
- What size data collections do these applications process in terms of:
  - input size?
  - output size?
- How long does the average program run take?
- How long does the longest program take to run?
- How long does a single batch job currently take to complete?
- How many CPU's are currently being utilized during the run?
- How many resources are consumed on average (particularly disk space and memory)?

B.2.2 Tools

- Do you make use of any batch processing or scheduling software such as LSF, PBS or Condor?
  - If yes, which software suite is used and what type of applications is it used for?
- Do you make use of any Grid software such as Globus or Sun Grid Engine?
  - If yes, which software suite is used and what type of applications is it used for?

- If you run both cluster software and Grid software, or run multiple types of cluster and or Grid software, do you have a standard interface from which to run the different software suites or does each software suite have its own independent interface and or command line client?
- What kind of cluster tools or protocols do you make use of heavily during the course of running a parallel application (please include cluster or Grid specific utilities, system utilities and 3rd party products)?

B.2.3 Cluster architecture

- What type of cluster are you currently utilizing (Beowulf, Rocks, Oscar, home-grown solution)?
- What operating systems are you currently running on you cluster(s)?
- What system architectures are you currently making use of?
  - PowerPC
  - X86 (32 bit)
  - X86 (64 bit)
  - Mac OS
  - Windows
- Which, if any, local or distributed data storage solutions or systems do you make use of (SRB, NFS etc).

B.3 User Interfaces

- Given a choice, would you rather make use of a front-end to a HPC environment or login to a console and use native OS commands?
- Why would you prefer this option?
- Would you prefer a combination of console-based and interface-based tools or either?
- Do you use any user interfaces that make executing a parallel job any easier? (Web-based or GUI based)?
- If you make use of some sort of user interface, what do you like most about it?
- If you do not make use of some sort of user interface, what key functionality would you require or like to see in such an interface?

B.4 Importance of HPC Computing

- How important is HPC to your research?
- Do you think that the use of a campus wide Grid be beneficial to your particular application(s)?

B.5 Clusters

- What is the size of your department's cluster and what are the node configurations?
- What percentage of the time, on average, are your clusters utilized?
- Would you be interested in participating in a campus Grid effort (note the conditions below)?
  - Clusters are under complete owner control
  - Profiles can be set that allows a job to run after specific conditions are met.

Appendix C

User Evaluation Exercise

A lightweight interface to local Grid scheduling systems

Evaluation Exercise

· Approximate Duration : 30mins ·

C.1 Introduction

This evaluation exercise consists of a number of tasks. In order for these exercises to be meaningful, some background information on Grid computing, scheduling techniques and parameter sweep applications will be provided in the form of a self-paced presentation. After the presentation, you will be asked a few questions in order to evaluate your understanding of the concepts you saw in the presentation. You will then have an opportunity to ask questions, after which you will complete a series of two tasks. These tasks will evaluate components of a Web-based Grid interface developed as part of this research project and each task will be followed by a short questionnaire. Finally, you will be asked to answer a few questions on certain aspects of the interface not related to any of the tasks in particular.

The questionnaires that you will be prompted to complete after each task will require written answers as well as questions where the most appropriate answer needs to be selected.

Please note that you are taking part in this exercise on a voluntary basis, purely for academic purposes, and the observations that will be recorded by means of pencil/pen and paper, as well as the results thereof, will be confidential. This analysis is strictly based on the tools provided, and at no point will the focus of the analysis be on you or your computer literacy. Feel free to ask questions at any time while performing the tasks and you are not obliged to complete any or all of the tasks, should there be a need for you not to do so.

C.2 Background Information

This section of the questionnaire will gather some basic information about yourself and your level of experience with various technologies. Once again, bear in mind that this is NOT a test.

C.2.1 Qualifications

Please provide some information about your major(s) and/or research interests and year of study if you are a student.

C.2.2 Parallel Computing

1. How do you rate your knowledge of parallel computing techniques and/or software?

Excellent	Very Good	Neutral	Poor	Very Poor

2. If you answered positively to the previous question, indicate which techniques or software tools you are acquainted with (please give a high-level description, i.e courses attended, projects done, etc.).

3. How do you rate your knowledge of volunteer computing projects?

Excellent	Very Good	Neutral	Poor	Very Poor

4. How do you rate your understanding of the concept of Grid computing?

Excellent	Very Good	Neutral	Poor	Very Poor

5. If you answered positively to the previous question, indicate whether your knowledge is constrained to theory only, or theory and practical application of Grid technology.

6. Do you know what a parameter sweep application is?

Yes	**No**

7. Have you ever heard of AJAX?

Yes	**No**

8. How often do you use Gmail, Facebook and/or Google Maps?

C.3 Overview of Grid computing

In order to prepare you for the tasks ahead, please ask the evaluator to start an automated presentation that will give you some background information on Grid computing. The presentation will also provide some information that you will need to submit a job to the Grid in the tasks to come. When you are done please feel free to ask the evaluator questions relating to the presentation, then turn the page over and complete the questionnaire. Please do not turn the page over until you have completed the presentation.

C.3.1 Questionnaire : Presentation

1. Please rate your understanding of basic Grid computing concepts in light of the information presented in the automated lecture.

Excellent	Very Good	Neutral	Poor	Very Poor

2. In light of the presentation, how do you rate your understanding of a parameter sweep application?

Excellent	Very Good	Neutral	Poor	Very Poor

3. The presentation highlighted the importance of schedulers in Grid environments. How do you rate the validity of the following statement? "Grid middleware hands off work to schedulers and does not schedule jobs at the organisational level."

Excellent	Very Good	Neutral	Poor	Very Poor

4. The presentation highlighted the importance of schedulers in Grid environments. How do you rate the validity of the following statement? "Execute nodes situated far apart need not necessarily be connected by Grid middleware since the public internet can connect them easily."

Excellent	Very Good	Neutral	Poor	Very Poor

C.4 Grid Status Task

Grid computing status systems are important to users as they provide information on the status of the Grid at the current point in time. Without this status information, users would be submitting jobs to a "black box" as they would have no way of evaluating the status of the Grid before they submit jobs. The Grid portal you are to to use provides a mechanism to view the current status of the Grid. The aim of this task is for you to find this information and answer some questions before filling out the questionnaire on your experiences with the task.

C.4.1 Task

1. A typical Grid is composed of a number of independent pools which operate as a whole to solve some problem. These pools, however, can be treated as independent entities and are therefore treated as such by the Grid interface. Your task is to find status information on the "HPC Cluster" pool and evaluate the operational status of this pool. Assume that you have noticed that some of your Grid jobs are taking longer than they should on this pool and you would like to find out why this is so. The table below will give you an indication of what information is required. Fill in the table using information gathered from the interface.

No. of machines Idle / Awaiting Jobs	**No. of machines Down**	**No. of architectures avail.**	**No. of OSs avail.**

2. The status tool provides a way of viewing statistics on each pool in the collapsed view (i.e before expansion of a component to the summary view. Which statistic do you think each of the five icons represent? If you do not know what an icon means, please indicate this.

C.4.2 Questionnaire : Status Task

1. Were you able to complete this task successfully?

Yes	No

2. If you did not manage to complete this task, state why you were not able to do so.

3. How do you rate the intuitiveness of the Grid status component?

Excellent	Very Good	Neutral	Poor	Very Poor

4. How do you rate the response time or speed of the Grid status component?

Excellent	Very Good	Neutral	Poor	Very Poor

5. How easy was it to make sense of the information displayed in the Grid status component?

Excellent	Very Good	Neutral	Poor	Very Poor

6. Do you think that the Grid status component could be improved in any way?

C.5 Job Submission Task

Your friend Peter is an avid music fan and he has thousands of audio files. He has just purchased a new computer but, alas, he has no money for Microsoft Windows as he bought a new set of speakers as well. Due to these unfortunate monetary circumstances, he has decided to install Linux on his new computer. He has, however, come to realise that the concept of open-source software is a great one. For this reason he is getting rid of all his proprietary audio file formats and, in so doing, needs to convert all his Windows media files to the `mp3` format. He does not know how to do this, so he has approached you. Since you are lazy, you downloaded a script with some supporting software that would do this for you automatically. However, when Peter brings you his music collection of 40435 `wma` files, you realise that you will have to use the university Grid to convert these as the conversions will take too long on a single computer. To do this, you have decided to do a test run, and perfect the Grid job before sending off all 40435 files to the Grid. You have selected 50 wma files and will now proceed to create a Grid job for these files.

Your task is therefore to build a Grid job using the Grid interface to create a parameter sweep application to submit Peter's job to the Grid. To do this you have read the manual for the script you downloaded and have written the following instruction set so you will remember how to use this script in the future:

"Run the script (written for a Linux operating system on an Intel architecture), called wmamp3.sh on the command line with the following arguments - 1) the name of the audio file to be converted - 2) a bit rate in kbps (Peter likes high quality so use 320) - 3) the tar archive with the supporting files, all space-separated of course. Example : wmamp3.sh beethoven_adagio_01.wma 320 encoders.tar"

Now that you have all the information you need, create a new job using the wizard on the interface. To do this, choose the correct option on the left pane of the interface and follow the instructions. Once you have created the job, the wizard will exit and you will be able to launch the job.

C.5.1 Questionnaire : Job Submission Task

1. How well did you understand the task that you were given?

Excellent	Very Good	Neutral	Poor	Very Poor

2. Were you able to complete this task successfully?

Yes	No

3. If you did not manage to complete this task, state why you were not able to do so.

4. How do you rate the intuitiveness of the job creation wizard in general?

Excellent	**Very Good**	**Neutral**	**Poor**	**Very Poor**

5. Some of the wizard components you encountered operated in an unconventional manner, i.e a manner not commonly encountered in similar systems. Indicate below how intuitive you found these "unconventional" interfaces in the context of a parameter sweep application.

   (a) Filtering wizard

Excellent	**Very Good**	**Neutral**	**Poor**	**Very Poor**

   (b) Argument specification wizard

Excellent	**Very Good**	**Neutral**	**Poor**	**Very Poor**

6. How do you rate the response time or speed of the job creation wizard?

Excellent	**Very Good**	**Neutral**	**Poor**	**Very Poor**

7. How easy was it to make sense of the information displayed in the job creation wizard?

Excellent	**Very Good**	**Neutral**	**Poor**	**Very Poor**

8. Do you think that the job creation wizard could be improved in any way?

C.6 Job Query Task

Now that you have specified and launched your job on the Grid, your friend Peter asks you to see how far the Grid job is. Use the interface to answer his question. To do this, you will need to find the part of the interface that displays job information. Once you have found this component, find the job you have submitted and record its status in the space provided below. Once you have done this, find the button on the interface that will refresh the window, thereby updating the information on the job you submitted. Repeat this process until all jobs have completed. Once you are satisfied that ALL jobs have completed successfully, indicate in the space below what lead you to this conclusion. If a subjob(s) has resulted in an error, delete the job(s). Indicate below which subjob(s) failed and which functionality you used to delete the job(s).

1. Were you able to find the information related to your Grid job?

Yes	**No**

2. If you did not manage to find this information, state why you were not able to do so.

3. How do you rate the intuitiveness of the job status window?

Excellent	**Very Good**	**Neutral**	**Poor**	**Very Poor**

4. How do you rate the response time or speed of the job status window?

Excellent	**Very Good**	**Neutral**	**Poor**	**Very Poor**

5. How easy was it to make sense of the information displayed in the job status window?

Excellent	**Very Good**	**Neutral**	**Poor**	**Very Poor**

6. Do you think that the job status window could be improved in any way?

C.7 General Feedback

1. Comment on the general aesthetics of the interface.

2. Could the aesthetics be improved in any way?

3. What features/aspects did you like about the interface?

4. What features/aspects did you NOT like about the interface?

5. How do you rate the overall responsiveness of the interface?

Excellent	**Very Good**	**Neutral**	**Poor**	**Very Poor**

6. How strongly would you associate this interface and its behaviour to that of applications such as Gmail/Flickr/Facebook?

Excellent	**Very Good**	**Neutral**	**Poor**	**Very Poor**

7. If you have any other comments/suggestions/criticisms, please provide them here.

· THE END ·
· Thank you for participating in this study ·

Bibliography

[Abbas, 2004] Abbas, A. (2004). *Grid Computing: A Practical Guide to Technology and Applications*. Networking Series. Hingham, MA: Charles River Media.

[Agarwal & Levy, 2007] Agarwal, A. & Levy, M. (2007). The KILL Rule for Multicore. In *Proceedings of DAC Conference* (pp. 750–753).: IEEE Computer Society.

[Anderson, 2004] Anderson, D. P. (2004). BOINC: A System for Public-Resource Computing and Storage. In *5th International Workshop on Grid Computing (GRID 2004), 8 November 2004, Pittsburgh, PA, USA, Proceedings* (pp. 4–10).: IEEE Computer Society.

[Anderson et al., 2002] Anderson, D. P., Cobb, J., Korpela, E., Lebofsky, M., & Werthimer, D. (2002). SETI@home: an experiment in public-resource computing. 45(11), 56–61.

[Anderson & Fedak, 2006] Anderson, D. P. & Fedak, G. (2006). The Computational and Storage Potential of Volunteer Computing. *CoRR*, abs/cs/0602061.

[Angel, 2001] Angel, E. (2001). *Interactive Computer Graphics: A Top-Down Approach With OPENGL primer package-2nd Edition*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

[Bar, 2003] Bar, M. (2003). OpenMosix, a Linux Kernel Extension for Single System Image Clustering. In *Proceedings of Linux Kongress: 10th International Linux System Technology Conference* (pp. 94–102). Saarbrucken, Germany.

[Baru et al., 1998] Baru, C. K., Moore, R. W., Rajasekar, A., & Wan, M. (1998). The SDSC storage resource broker. In S. A. MacKay & J. H. Johnson (Eds.), *CASCON* (pp.5̃).: IBM.

[Beckles, 2005] Beckles, B. (2005). Re-factoring Grid computing for usability. In *Proceedings of the UK e-Science All Hands Meeting 2005* Nottingham, UK.

[Berners-Lee et al., 1994] Berners-Lee, T. et al. (1994). The world-wide web. *Communications of the ACM*, 37(8), 76–82.

[Blender, 2008] Blender (2008). Blender Animation Software. www.blender.org/.

[Bruin et al., 2006] Bruin, R., White, T., Walker, A., Austen, K., & Dove, M. (2006). Job submission to Grid computing environments. In *Proceedings of the UK e-Science All Hands Meeting 2006* (pp. 754–761). Nottingham, UK.

[Butt et al., 2003] Butt, A. R., Zhang, R., & Hu, Y. C. (2003). A Self-Organizing Flock of Condors. In *Proceedings of the IEEE/ACM Supercomputing Conference (SC2003)* (pp.4̃2). Phoenix, AZ, November 15-21: IEEE Computer Society Press.

[Cacheda et al., 2005] Cacheda, F., Plachouras, V., & Ounis, I. (2005). A case study of distributed information retrieval architectures to index one terabyte of text. *Inf. Process. Manage*, 41(5), 1141–1161.

[Codling, 2003] Codling, J. (2003). Using Macromedia Flash to Produce Rich Multimedia Content on the World Wide Web (v.1.1).

[Condor, 2007a] Condor (2007a). Condor GCB. http://www.cs.wisc.edu/condor/gcb/.

[Condor, 2007b] Condor (2007b). Condor version 7.0.4 manual. http://www.cs.wisc.edu/condor/manual/v7.0/.

[Condor, 2008] Condor (2008). HPC vs. HTC. http://www.cs.wisc.edu/condor/htc.html [Last Accessed 26/05/09].

[Cormode & Krishnamurthy, 2008] Cormode, G. & Krishnamurthy, B. (2008). Key differences between Web 1.0 and Web 2.0. *First Monday*, 13(6).

[Crane et al., 2005] Crane, D., Pascarello, E., & James, D. (2005). *AJAX in Action*. Greenwich, CT, USA: Manning Publications Co.

[Curnow & Wichmann, 1976] Curnow, H. J. & Wichmann, B. A. (1976). A Synthetic Benchmark. *The Computer Journal*, 19(1), 43–49.

[Czajkowski et al., 1998] Czajkowski, K., Foster, I., Karonis, N., Kesselman, C., Martin, S., Smith, W., & Tuecke, S. (1998). A Resource Management Architecture for Metacomputing Systems. *Lecture Notes in Computer Science*, 1459.

[Dahan et al., 2004] Dahan, M., Thomas, M., Roberts, E., Seth, A., Urban, T., Walling, D., & Boisseau, J. R. (2004). Grid Portal Toolkit 3.0 (GridPort). In *High performance Distributed Computing, 2004. Proceedings. 13th IEEE International Symposium* (pp. 272–273).: IEEE Computer Society.

[Dean & Ghemawat, 2008] Dean, J. & Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1), 107–113.

[Delic & Walker, 2008] Delic, K. A. & Walker, M. A. (2008). Emergence of the academic computing clouds. *Ubiquity*, 9(31), 1–1.

[Elephant's Dream, 2008] Elephant's Dream (2008). Elephant's Dream. www.elephantsdream.org/ [Last Accessed 26/05/09].

[Feldman et al., 2006] Feldman, J., Muthukrishnan, S., Sidiropoulos, A., Stein, C., & Svitkina, Z. (2006). On the Complexity of Processing Massive, Unordered, Distributed Data. *CoRR*, abs/cs/0611108.

[Foster & Kesselman, 2004] Foster, I. & Kesselman, C. (2004). *The Grid 2 : Blueprint for a new Computing Infrastructure*. Morgan Kaufmann Publishers, second edition.

[Foster et al., 2001] Foster, I., Kesselman, C., & Tuecke, S. (2001). The Anatomy of the Grid: Enabling Scalable Virtual Organization. *The International Journal of High Performance Computing Applications*, 15(3), 200–222.

[Foster, 2005] Foster, I. T. (2005). *A Globus Primer*. Argonne National Laboratory.

[Foster, 2006] Foster, I. T. (2006). Globus Toolkit Version 4: Software for Service-Oriented Systems. *J. Comput. Sci. Technol*, 21(4), 513–520.

[Frey et al., 2002] Frey, J., Tannenbaum, T., Livny, M., Foster, I. T., & Tuecke, S. (2002). Condor-G: A Computation Management Agent for Multi-Institutional Grids. *Cluster Computing*, 5(3), 237–246.

[Garrett, 2005] Garrett, J.-J. (2005). Ajax : A new approach to Web Applications. http://www.adaptivepath.com/publications/essays/archives/000385.php [Last Accessed 23/05/09].

[Global Grid Forum, 2007] Global Grid Forum (2007). Job Submission Description Language (JSDL) Specification, Version 1.0. http://www.ogf.org/documents/GFD.56.pdf.

[Gutenberg, 2008] Gutenberg (2008). Project Gutenberg. www.gutenberg.org/ [Last Accessed 23/05/09].

[Hipson, 2005] Hipson, P. (2005). *Firefox and Thunderbird: Beyond Browsing and Email*. Indianapolis, IN, USA: Que Corp.

[Jazayeri, 2007] Jazayeri, M. (2007). Some Trends in Web Application Development. In *FOSE '07: 2007 Future of Software Engineering* (pp. 199–213). Washington, DC, USA: IEEE Computer Society.

[Lerner, 2007] Lerner, R. (2007). At the Forge: Firebug. *Linux Journal*, 2007(157), 8.

[Linderoth et al., 2000] Linderoth, J., Goux, J.-P., & Yoder, M. (2000). *Metacomputing and the Master-Worker Paradigm*. Technical Report ANL/MCS-P792-0200, Mathematics and Computer Science Division, Argonne National Laboratory.

[Litzkow et al., 1988] Litzkow, M., Livny, M., & Mutka, M. (1988). Condor - A Hunter of Idle Workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems* (pp. 104–111).

[Mackinlay, 1988] Mackinlay, J. D. (1988). Applying a Theory of Graphical Presentation to the Graphic Design of User Interfaces. In *ACM Symposium on User Interface Software and Technology* (pp. 179–189).

[Mahemoff, 2006] Mahemoff, M. (2006). *AJAX Design Patterns*. O'Reilly Media, Inc.

[Maulsby et al., 1993] Maulsby, D., Greenberg, S., & Mander, R. (1993). Prototyping an intelligent agent through Wizard of Oz. In *Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems* (pp. 277–284). New York, NY, USA: Communications of the ACM.

[Message-Passing Interface Forum, 1997] Message-Passing Interface Forum (1997). *MPI-2.0: Extensions to the Message-Passing Interface*, chapter 9. MPI Forum.

[Moore, 1965] Moore, G. E. (1965). Cramming more components onto integrated circuits. *Electronics*, 38(8), 114–117.

[Nielsen & Landauer, 1993] Nielsen, J. & Landauer, T. K. (1993). A Mathematical Model of the Finding of Usability Problems. In S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel, & T. White (Eds.), *Proceedings of the Conference on Human Factors in computing systems* (pp. 206–213). New York: Communication of the ACM.

[OMII UK,] OMII UK. Gridsam. http://www.omii.ac.uk/wiki/GridSAM [Last Accessed 26/05/09].

[O'Reilly Media, 2007] O'Reilly Media (2007). Web2.0. http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html [Last Accessed 26/05/09].

[Papadopoulos et al., 2003] Papadopoulos, P. M., Katz, M. J., & Bruno, G. (2003). NPACI Rocks: tools and techniques for easily deploying manageable Linux clusters. *Concurrency and Computation: Practice and Experience*, 15(7-8), 707–725.

[Preece et al., 2002] Preece, J., Rogers, Y., & Sharp, H. (2002). *Interaction Design: Beyond Human-Computer Interaction*. John Wiley & Sons. OCLC 48265540.

[Scott, 2001] Scott, S. L. (2001). OSCAR and the Beowulf Arms Race for the "Cluster Standard". In *Proc. 2001 IEEE International Conference on Cluster Computing (3rd CLUSTER'01)* (pp. 137). Newport Beach, California, USA: IEEE Computer Society.

[Sebu & Ciocarlie, 2006] Sebu, L. & Ciocarlie, H. (2006). The Design of Stateful Web Services Based on Web Service Resource Framework Implemented in Globus Toolkit 4. In *SYNASC* (pp. 309–316).: IEEE Computer Society.

[Sefelin et al., 2003] Sefelin, R., Tscheligi, M., & Giller, V. (2003). Paper prototyping - what is it good for?: a comparison of paper- and computer-based low-fidelity prototyping. In *CHI '03: CHI '03 extended abstracts on Human factors in computing systems* (pp. 778–779). New York, NY, USA: ACM.

[Shu et al., 2005] Shu, C., Yu, H., Xiao, L., Liu, H., & Xu, Z. (2005). Towards an end-user programming envrionment for the Grid. In *Proceedings of the Grid and Cooperative Computing Conference* (pp. 345–356). Beijing, China.

[Sommerer, 2004] Sommerer, R. (2004). *Presentable Document Format: Improved On-demand PDF to HTML Conversion*. Technical Report MSR-TR-2004-119, Microsoft Research (MSR).

[Sunderam, 1990] Sunderam, V. S. (1990). PVM: A Framework for Parallel Distributed Computing. *Concurrency - Practice and Experience*, 2(4), 315–339.

[Thomas W. MacFarland,] Thomas W. MacFarland. Mann Whitney U-test. http://www.nyx.net/~tmacfarl/STAT_TUT/mann_whi.ssi [Last Accessed 26/05/09].

[Underhill & Bradfield, 2001] Underhill, L. & Bradfield, D. (2001). *IntroSTAT Second Edition*. Juta.

[Underwood et al., 2004] Underwood, K. D., Ligon, III, W. B., & Sass, R. R. (2004). An Analysis of the Cost Effectiveness of an Adaptable Computing Cluster. *Cluster Computing*, 7(4), 357–371.

[W3C, 2004] W3C (2004). Web services architecture. http://www.w3.org/TR/ws-arch/.

[Weinreich et al., 2008] Weinreich, H., Obendorf, H., Herder, E., & Mayer, M. (2008). Not Quite the Average: An Empirical Study of Web use. *TWEB*, 2(1).

[Wikipedia, 8 09] Wikipedia (2007-08-09). History of the Graphical User Interface.

[Zkoss.org, 2008] Zkoss.org (2008). The ZK Toolkit. http://zkoss.org [Last Accessed 26/05/09].