A Lightweight Interface to Local Grid Scheduling Systems

Christopher Parker[a], Kyle Williams[a,], Hussein Suleman[a]

[a]*Department of Computer Science, University of Cape Town, Private Bag X3, Rondebosch, South Africa, 7701*

Abstract

Many computationally intensive research problems can be addressed using a Grid architecture. However, Grid use is restricted to those who have an in-depth knowledge of its complex architecture and functionality. To make Grid computing more accessible, a lightweight Web 2.0 interface to the scheduling systems on which Grids rely, and which can serve as an abstraction of a large Grid environment, was built. The purpose of this interface was to simplify many of the complexities associated with using Grid architectures. A case study is used to demonstrate the applicability of the interface to a problem that can be solved using a Grid, while a user study demonstrates how users, with little or no experience using Grids, were able to accomplish tasks using the Grid. Lastly, it is shown that the Web 2.0 interface can outperform traditional static interfaces in terms of response time and bandwidth efficiency.

Keywords: lightweight interface, scheduling systems, AJAX, Web 2.0, Grid computing

Email addresses: cparker@cs.uct.ac.za (Christopher Parker), kwilliams@cs.uct.ac.za (Kyle Williams), hussein@cs.uct.ac.za (Hussein Suleman)

1. Introduction

Grid computing middleware is often complex and makes use of low-level command line utilities in order to create, launch and monitor jobs on the Grid. The use of such utilities requires extensive knowledge of low-level Grid operations in order to know when, and in which context, certain utilities need to be used. As a result, the Grid is not as accessible as it could be.

In parallel to development of Grid computing technology, use of the World Wide Web has grown significantly. The growth of Web 2.0 technologies has resulted in 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 [1]. Furthermore, core Grid middleware has transitioned from a monolithic architecture to a component-based Web-service model [2]. 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. As a result, and in order to overcome the usability problems currently plaguing Grid technology, a Web-based and Web 2.0-inspired system, capable of abstracting away many of the complexities associated with grid computing, was built.

There are several key objectives that this Web-based system set out to address: determine if a Web interface to a local Grid computing system can be both functional and usable; determine if computer-literate non-Grid experts are able to make use of the interface; determine whether lightweight, Web 2.0 techniques live up to claims of increased usability, speed and decreased response time; and ensure that the interface is extensible by allowing for inclusion of different schedulers as "plug-ins" or components.

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. The approach and results are applicable to large-scale Grids as well. Since Grid middleware acts as a communication medium among 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.

The rest of this article is set out as follows. AJAX, a technology not typically related to Grid computing, will be introduced in Section 2 and the infrastructure and design of the system will be discussed in Sections 3 and

4. Section 5 will make use of a case study to demonstrate the applicability of the Web interface to a real-life parameter sweep Grid computing problem, followed by a user evaluation of the Web interface in Section 6 and a performance evaluation in Section 7. Related work will be discussed in Section 8 and conclusions will be discussed in Section 9.

2. AJAX

Asynchronous JavaScript and XML (or AJAX) represents a relatively new approach to Web development. 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, which increase usability and make 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 and XML/Document Object Model (DOM); and, most importantly, the XMLHttpRequest object, which allows for asynchronous communication [3]. The term AJAX is then used to represent an application development paradigm that makes use of existing technologies already present in many browsers.

In a traditional Web application 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 1). 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 sends data back to the client application. 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

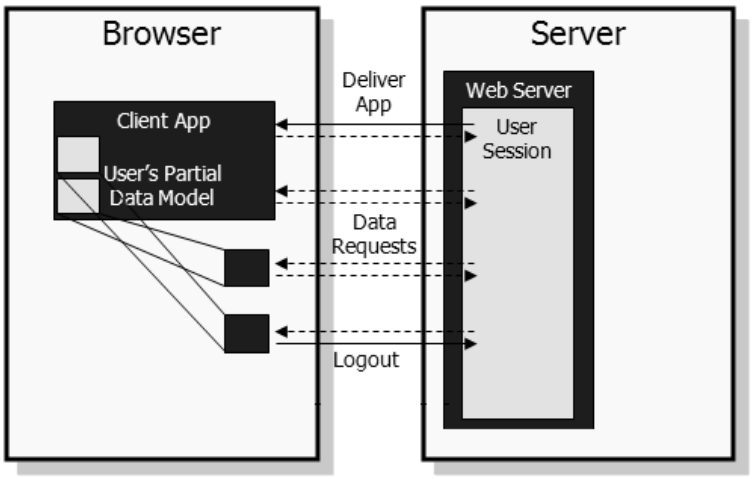

Figure 1: AJAX Web Application

the application has been delivered to the client - only small amounts of data.

By making pages dynamic with the use of AJAX, a number of efficiencies and characteristics that websites have strived for over the years is achieved. 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 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 improved overall user experience.

3. Infrastructure

The infrastructure upon which the interface was built involved a test grid, containers, schedulers, a database and a directory structure.

The test grid infrastructure consisted of three computing clusters, one Grid server and a handful of Windows workstations, all running the Condor scheduler [4]. The clusters consisted of dedicated machines running the Linux and FreeBSD operating systems and the Windows workstations were utilised on a volunteer basis. All machines were connected using the Condor [4] flocking mechanism. The LoadLeveler scheduler [5] was installed on one of

the clusters as well.

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. 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 Parameter Sweep Description Language (PSDL), a customised form of the Job Submission Description Language (JSDL) [6] for parameter sweep applications, into a format understood by each scheduling system. These converters each implement a set of methods that the various interface components call in order to populate the relevant on-screen panels. The converters not only enable different scheduling systems to be added over time, but also allow for more complex interface features to be added.

It was found in the requirements gathering sessions that a way of displaying the status of the Grid was important. In order to accomplish this, scripts were built to gather and store status information. These scripts, custom-written for each scheduler, ensure that the multi-scheduler design approach is realised by separating scheduler logic from interface logic. 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. Finally, WebDAV shares were created to allow users to bulk-upload files to the Grid server.

4. Design

4.1. Overview

An overview of the layout of the Web interface is presented in Figure 2 and, as can be seen from the figure, the interface consists of four main panes, namely the information header at the top, the menu pane on the left-hand side, the status pane in the centre and the notification pane on the right-hand side. The status pane is one of the most important parts of the interface as it 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 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.

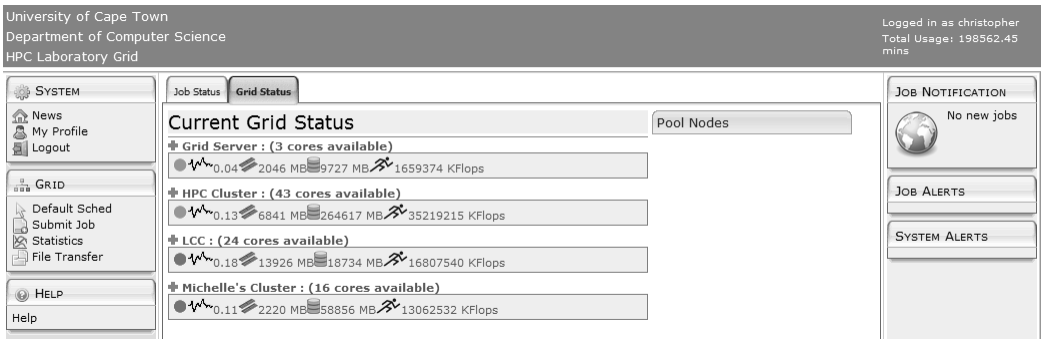

Figure 2: Grid interface layout

The design of the Web-based interface was derived from a process of requirements gathering in which Grid users were interviewed. The main finding was that Grid users wanted a system that simplified Grid computing and emphasised submission and monitoring tools as being the most important. Furthermore, the parameter sweep approach was chosen as a fundamental Grid job type to be supported by the interface, where an application is run multiple times with different input files and input arguments for each individual run. A resource-centric view of the system, which keeps certain information visible at all times, was chosen during the interface design. As such, the status of the Grid is visible at all times, except when a user changes from Grid status mode to job status mode.

4.2. Interface Components

There are a number of key components that make up the system, including: the grid status component, the job specification component, launching and querying components and miscellaneous components. An overview of each of the major components in terms of design and functionality as well as the way in which they communicate with the server are presented below.

4.2.1. Grid Status

The Grid status component displays the current status of the Grid and is updated by Python scripts that wrap around the low-level command line utilities. Since the way in which scheduling software is written differs 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.

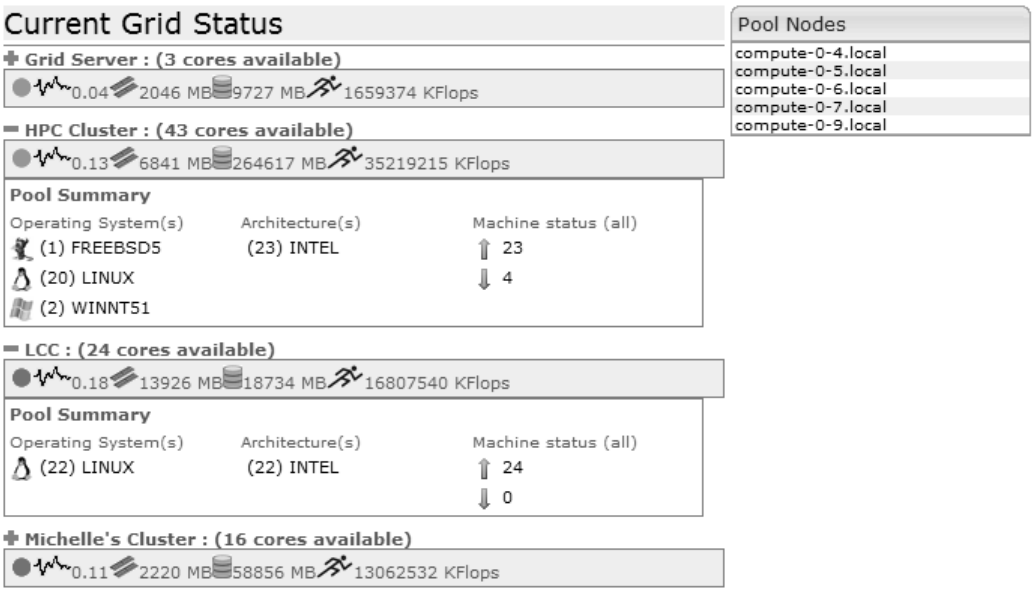

Figure 3: Grid status component

Figure 3 shows a typical Grid status snapshot in both collapsed and expanded view. From the figure, four pools are visible and the figure also shows status information pertaining to each pool. This summary provides a breakdown of the architectures and operating systems present within the pool as well as an indication of the numbers of machines that are up and 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. Lastly, the final element in the status display is the "Pool" Nodes box and contains the names of the machines in a selected 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 who are developing applications that are to be run on the Grid. 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, given the current state of the Grid.

4.2.2. Job Submission

The second major interface component is for Job Submission component. The job submission process is split into two steps, namely: job specification and job launching. The job specification process takes the form of a wizard. This wizard allows a user to choose whether to create a new job or load a PSDL template from a file, and includes the following steps:

1. **Job and application specific information** prompts 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 to be created on the server.
2. **Resource filtering** allows a user to choose the systems that his/her job is to be executed on and automates the process of matching the available resources to applications.
3. **Executable selection** prompts the user for the binaries or executables needed for the job.
4. **Input file selection** prompts the user for a set of input files.
5. **Input argument enumeration** allows the user to specify a dynamic set of changing argument types to be passed to each run of the application. The first type is the number type, a value that is incremented for each run of the application. The next type, the flag type, is a static type. A flag is merely repeated verbatim for each run of the application. The single-file type specifies the same file for each run of the application. The final argument type is the multiple-file type that allows a user to dynamically allocate a file from the directories of input files selected in the previous step. An important feature of this wizard page is a sample command-line window. This window displays the result of the argument enumeration procedure and 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. Figure 4 shows the input argument enumeration wizard.
6. **Output-specific settings** involves the selection of output parameters such as whether or not log files are produced and the output file names.

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 and the PSDL XML document is then written to the user's job templates

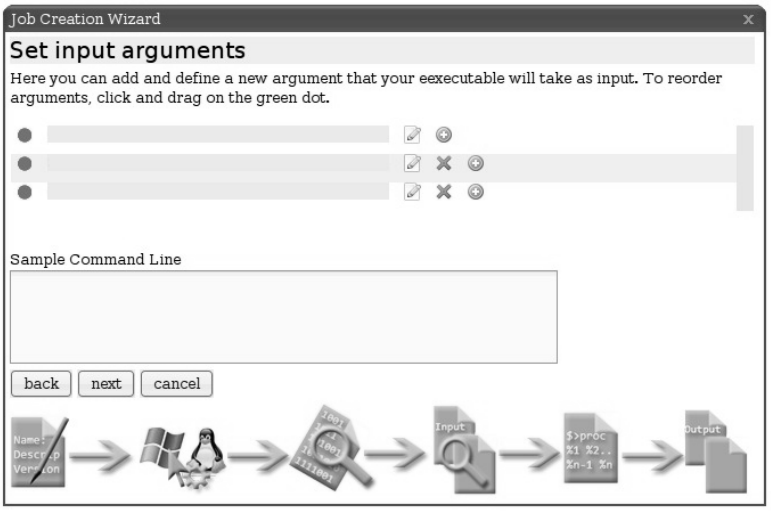

Figure 4: Input argument enumeration

directory. The presence of the PSDL file allows the user to load a pre-existing job into the browser using the job submission wizard.

4.2.3. Job Launching

Job launching is the second step in the job submission process. A user is notified when a job is ready to be launched via an icon in the Job Notification panel on the right side of the interface (see Figure 2). When this icon is clicked, the process of submitting the job to the Grid begins and an AJAX-based progress indicator is shown at the top of the screen.

4.2.4. Job Querying

The third major interface component is the job querying component (see Figure 5), which allows users to view the status of jobs submitted to the Grid.

The query window gives the user a number of options options. Firstly, if a job has been "held" due to some error, the user is able to delete the entire job or the offending sub-job. The interface also provides a way for users to filter jobs by date range or status (or both) and when a job completes, the output files can be retrieved and moved to a data directory on the server.

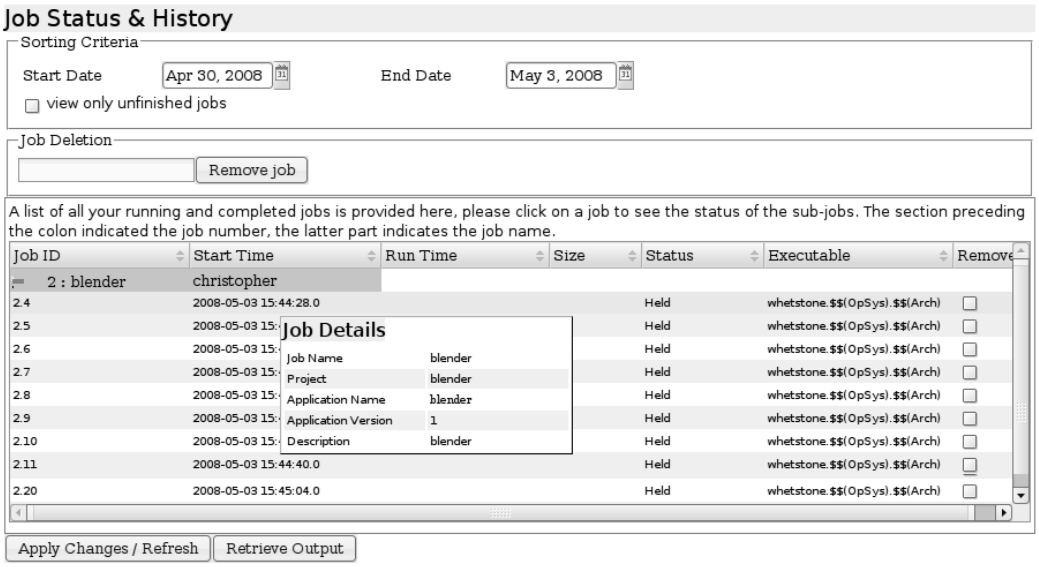

Figure 5: Job Status Window

5. Case Study: Audio Converter

The aim of the Web interface is to abstract and simplify the use of Grid computing tools. It is therefore necessary 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. A case study representing a real world computational task was used to provide evidence that the system is useful for its intended task.

Many applications, allow for the conversion of one document format to another. Data conversions are typically done in bulk as part of some process. Since such conversions are usually Single Instruction Multiple Data (SIMD) in nature as well as computationally intensive, it is possible to parallelize such a bulk process. This case study was used to illustrate how the Web interface can be used to perform a batch conversion from one audio format (.wma) to an another (.mp3).

5.1. Process

This case study makes use of a bash script as the "executable'. Since the audio converter written for this case study makes use of existing conversion

software (Mplayer and toolame), the bash script was used to run these pieces of software in sequence.

This case study in essence relied on the existence of three executables: the bash script, Mplayer and toolame, the supporting library files needed by these tools, as well as the files to be converted. The entire job was therefore broken down into three sets of files: the executable, the supporting software and the input files. The supporting software and libraries, was packaged into a tar archive that is extracted as soon as the script is run and therefore does not rely on the execute nodes having the software pre-installed.

The bash script was written to accept one argument, an input wma file. 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. 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 command lines for 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

5.2. Results

The amount of data that needed to be spooled for the submission was 4.8GB and as such the submission process took approximately 17 minutes. 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 of 40.63 minutes and Grid time of 28.63 minutes, 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. These figures are reported in comparison to a serial time, which is calculated by adding together the individual real times for the entire run of the case study, of 886.1 minutes. The difference is a Grid speedup by a factor of 21.81.

This case study has shown how a common data processing problem can be "Gridified" with relative ease and submitted using the Web interface and how applications with non-trivial software dependencies can be submitted using the interface and successfully be deployed to the Grid. Additional case studies investigating the applicability of the Web interface to real world problems are reported in [7].

6. User Evaluation

Research scientists consulted at the beginning of this study believed an interface to a Grid should have mechanisms for viewing the status of the Grid, submitting jobs to the Grid and monitoring 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. Furthermore, each task was followed by a short questionnaire that enabled test subjects to assess these components individually and required both written answers as well as answers to Likert-scale questions. Other important information was gathered at various stages of the evaluation process using a 15 page questionnaire.

The user evaluation was separated into 6 phases: user background analysis and introductory presentation, a grid status monitoring task, a job submission task, a job query task and a general feedback session.

6.1. User Background Analysis

The first part of the questionnaire gathered data on the qualications, research interests and background information of test subjects. Background on HPC as well as knowledge on volunteer and Grid computing of test subjects was gathered. The main goal of gathering background information on test subjects was to determine how much knowledge they had of HPC and Grid technologies.

It was found that a large number of test subjects had a below average understanding of Grid computing concepts. 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 then the objective to make the Grid usable by computer literate non-Grid experts would have been achieved.

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. 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 used such applications less frequently.

At the outset of the experimentation phase, it was expected that test subjects would not be familiar with the concept of a computational Grid and an introductory, pre-recorded slideshow on Grids, scheduling systems and parameter sweep applications was presented prior to the evaluation. 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 it provided.

6.2. Task 1: Grid Status Monitoring

The grid status task was set to analyse the ability of users to monitor the status of the grid.

The first sub-task required test subjects to locate the Grid status component (Figure 3). 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 number of machines idle or awaiting jobs as they got confused between this value and the number of cores available in the summary view. The other information required in this task was easily found by most test subjects and 79% of subjects managed to obtain three or more correct answers for this question.

The second sub-task required test subjects to give their interpretation of the icons present in the summary view(s) of Grid pools. Five icons, each representing a different pool statistic, are presented in each summary view.

Rating	Ex.	V. Good	Neutral	Poor	V. Poor
Intuitiveness	2	14	7	1	0
Response Time	3	13	5	3	0
Sensible Data Rep.	3	15	6	0	0

Table 1: Grid status component ratings; n = 24

The median for number of correct observations is four out of five. This is an 80% success rate for correctly identifying the status icons and suggests that the icons are intuitive and easy to understand.

6.2.1. 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. Table 1 provides a summary of test subject ratings and shows that the majority of the scores for all three ratings are in the "Very Good" category.

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 most of the information presented in the component.

6.3. Task 2: Job Submission

The second step in the usage scenario is the creation and submission of a Grid job. For this task, test subjects were provided with a short problem statement that outlined the nature of a Grid job that they were to submit to the Grid. 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 completion of the task. However, observational data was gathered. All test subjects, apart from one who indicated that she did understand the problem, were able to complete the task. Due to the nature of the one subject's failure being unclear and the ambiguity of her response, her responses were excluded from further analysis.

During completion of the tasks, a few interface usability problems were observed. The first was that many test subjects neglected to select an architecture from the filtering wizard. Test subjects also became confused because selected files were not highlighted in the file selection wizard and they tried to enter text into the read-only text fields during argument enumeration.

Rating	Ex.	V. Good	Neutral	Poor	V. Poor
Intuitiveness	2	9	9	1	1
Response Time	6	8	8	0	0
Filtering Intuitiveness	7	9	5	1	0
Arg. Enum. Intuitiveness	2	11	5	4	0
Sensible Data Rep.	4	9	8	1	0

Table 2: Job creation component ratings; n = 22

These shortcomings, however, can be corrected by making minor interface appearance changes. 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.

6.3.1. Component Evaluation

As with the previous task, test subjects rated the task according to the categories presented in Table 2. In addition to the three ratings present in the previous task, test subjects also were asked to rate the levels of intuitiveness of the filtering and argument enumeration wizards since these wizards are the most complex in terms of the Grid job creation process.

17% of test subjects rated the argument enumeration wizard as "Poor." Since this wizard was designed specically 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. A statistical analysis showed 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 - the argument enumeration wizard. The response time and data presentation were all highly rated, providing further evidence to support this claim.

Rating	Ex.	V. Good	Neutral	Poor	V. Poor
Intuitiveness	5	12	7	0	0
Response Time	2	13	6	3	0
Sensible Data Rep.	7	16	1	0	0

Table 3: Job monitoring component ratings; n = 24

6.4. Task 3: Job Monitoring

The final task in the usage scenario 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. 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 job/sub-job(s).

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 did not see the expansion icon (+). Furthermore, test subjects also got confused between the job and sub-job deletion mechanisms.

6.4.1. Component Evaluation

Upon completion of the job monitoring task, test subjects were once again instructed to rate the task. Table 3 provides a summary of test subject responses.

As can be seen from the table, the results are similar to those of the previous two tasks. There is evidence to support the claim that the Grid monitoring component is intuitive and has a sensible data layout. The response time scores are not as compelling. Since the component takes a long time to refresh, test subject responses reflected this. The median for the job monitoring component was "Very Good." However, there were a number of problem areas that were highlighted, such as slow response times and ambiguous deletion mechanisms. These, however, can easily be addressed.

6.5. 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 section aims

Statistical Analysis	Med.	Min.	Max.	Std. Dev.
Intuitiveness	3.66	2.33	4.66	0.51
Response Time	3.83	2.33	4.66	0.67
Sensible Data Rep.	4.0	3.0	4.66	0.46

Table 4: Descriptive statistics for average over all tasks; n = 24

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.

6.5.1. 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 4.

The standard deviation for all three metrics is reasonably low (less than 1). 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 interface is sensible and easy to interpret, thus providing further evidence to support the high usability ratings presented in the previous sections.

6.5.2. 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. Test subjects were also asked to rate the Web interface in terms of its overall responsiveness and level of dynamic functionality, the results of which are presented in Tables 5 and 6. As can be seen from the tables, test subjects reported 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 in Table 4. 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.

The test subject background analysis showed that 96% of test subjects had heard of the AJAX Web development paradigm. This result is important

Rating of	Ex.	V. Good	Neutral	Poor	V. Poor
Responsiveness	5	12	7	0	0
Level of Dynamic Func.	2	13	6	3	0

Table 5: Overall interface ratings; n = 22

Statistical Analysis	Med.	Min.	Max.	Std. Dev.
Responsiveness	4.0	2.0	5.0	0.789542
Level of Dynamic Func.	4.0	1.0	5.0	0.984732

Table 6: Descriptive statistics of overall interface ratings; n = 22

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 Tables 5 and 6, 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 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 general consensus on the part of test subjects was that the interface is clean and easy to understand due to its structure. Other comments made by test subjects were that the interface fits in well with that of the Web browser's own interface and appears professional and "finished off." Some test subjects noted that, since the interface is a very specialised tool, novices might have trouble understanding it. However, since the interface is a scientic 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.

Some possible improvements were brought to light and mainly had to do with text size, notications and alerts and 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, among other minor suggestions.

7. Performance Evaluation

The user evaluations 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. For this reason, an evaluation of performance was conducted.

A study was conducted to measure the performance of the Web application, 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. This was not possible in when evaluating the Web interface since only one version of the interface exists, i.e the AJAX version, and there is no suitable candidate to compare the interface to. Building a duplicate system using a traditional development approach was considered wasteful, therefore prompting the use of a simulated analysis of how the AJAX interface would operate if the traditional development approach was used instead.

7.1. 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, a Grid job submission and monitoring scenario was decided upon, consisting 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. This usage scenario 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[1] tool was used. Firebug is a Mozilla Firefox plugin that provides a wealth of information as a page is loading, by breaking up the data communication into its constituent components. Firebug displays the page elements that are loaded as well as the times taken to load each individual element. Furthermore, Firebug provides data on the size of each of the HTML, CSS, JavaScript

[1] http://getfirebug.com

(JS), AJAX (XHR), image and Flash components of a client-server communication. Each of these components was analysed separately for each of the sub-tasks.

For the AJAX interface, obtaining this information was as trivial as loading the Web application 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: 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; no JavaScript or dynamic components would be present; the initial CSS file would be served only once and would then be resident in the browser cache; as with the AJAX interface, each unique image would be taken into account only once and would then be retrieved from cache; unlike the AJAX interface, no overlapping windows would be used and therefore the main display area would be updated after each task; and 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.

7.2. Results

Sub-tasks 1-20 in Figure 6 show the 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 as the AJAX interface does not reload data that does not need to be updated. Therefore, by the time the user is three quarters of the way through a job creation procedure, the interface becomes more efficient than its HTML-only counterpart. Beyond this, Figure 6 shows the effect of a longer usage session on overall bandwidth usage for 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 graph, the bandwidth usage over a longer period of time is greatly reduced by using an AJAX-based approach.

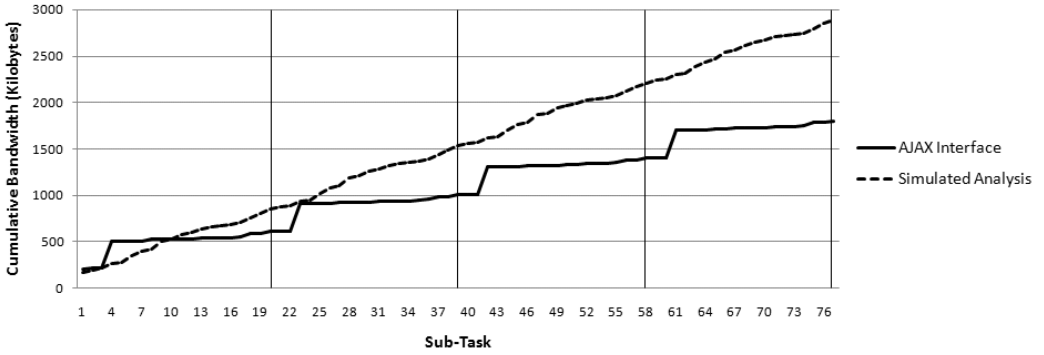

Figure 6: Cumulative bandwidth usage of the AJAX interface vs. an identical hypothetical non-dynamic HTML-only interface for four consecutive repetitions of the interface usage scenario

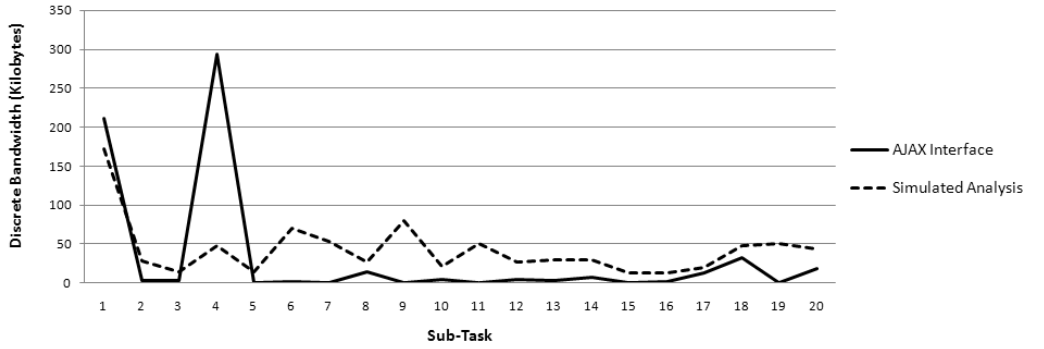

Figure 7: Discrete bandwidth consumption of the AJAX interface vs. an identical hypothetical non-dynamic interface for the interface usage scenario

Figure 7 shows the non-cumulative bandwidth consumption of the usage scenario outlined above. From the figure it is clear that by using the AJAX approach there is a signicant decrease in data transfer from the server to the browser. This decreases response time and provides more pleasant user experience.

The results presented in this section show that the AJAX-based approach to Web application development has signicant advantages in terms of decreased overall bandwidth usage and response time. 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 traditional development paradigms. These experiments provide conclu-

sive results to 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.

8. Related Work

Numerous Grid portal systems and development toolkits are available to enable high-level access to Grid computing environments. Toolkits such as GridPort [8] and containers such as Gridsphere [9] allow developers to quickly build and deploy Grid-based tools. The aim of these systems is to allow for a highly customisable, standards-compliant design by allowing developers to write small portlets that plug into a container.

The GridPort toolkit [8] is one of the more well known Grid-portal development toolkits. This toolkit takes a layered approach, the first layer representing the actual Web applications (scheduler interfaces, file managers, etc.) and container. The second layer - the service layer - consists of third party Grid software services (schedulers, Grid middleware, etc.) upon which the Web applications rely. The third layer - the resource layer - consists of the actual hardware resources, comprising of both computational and storage resources.

Examples of Web-based portals to Grids include WebCom [10], a Web-based computational Grid portal built and tested at the Xian Jiaotong University, the Pegasus Portal [11], a portal for submitting workflows to the Grid that allows for the generation of abstract workflows based on metadata descriptions and the NPACI HotPage portal [12] that provides access to NPACI resources via a simple interface based on the GridPort toolkit. The lightweight interface in this study differs from these other interfaces in that it explicitly attempts to abstract away many of the complexities associated with Grid computing and utilizes AJAX to mimic a desktop-like paradigm, with the goal of making the Grid accessible to users with little or no experience using Grids.

9. Conclusions

The usability of scientic 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 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. The system was designed to allow for additional schedulers to be "plugged" into the system using a set of scripts.

A case study demonstrated the ability of the lightweight interface to simplify and execute a real-world parameter sweep application. Feedback from a user evaluation study showed that non-expert Grid users, with limited knowledge of HPC, were able to use the Web interface to complete tasks and make use of the grid. Users gave above average ratings for usability, sensible data presentation as well as response time for each of the jobs in isolation and similar responses for the system as a whole. Performance evaluations were used to determine how well the interface responded to user requests by making use of a typical usage scenario. The results from these experiments provide conclusive results to prove that the AJAX-based interface is more efficient than a static HTML interface during the course of a typical usage scenario, both in terms of response time and bandwidth efficiency.

It is believed that an interface of this nature greatly simplifies the process of working with a Grid, both for expert and non-expert users, and has the potential to increase the reach of the Grid in modern day computing.

References

[1] P. J. Deitel, H. M. Deitel, AJAX, Rich Internet Applications, and Web Development for Programmers, Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2008.

[2] M. Brown, Build grid applications based on SOA, 2005, http://www.globusconsortium.org/journal/20060502/brown.html Accessed: 12 March 2012.

[3] M. Mahemoff, Ajax Design Patterns, O'Reilly Media, Inc., 2006.

[4] M. Litzkow, M. Livny, M. Mutka, Condor - a hunter of idle workstations, in: Proceedings of the 8th International Conference of Distributed Computing Systems, 1988.

[5] IBM Tivoli workload scheduler LoadLeveler, http://www-01.ibm.com/software/tivoli/products/scheduler-loadleveler/ Accessed: 12 March 2012.

[6] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. Mcgough, D. Pulsipher, A. Savva, Job Submission Description Language (JSDL) Specification, Version 1.0, Technical Report, Global Grid Forum, 2005.

[7] C. Parker, A lightweight interface to local grid scheduling systems, msc thesis, http://pubs.cs.uct.ac.za/archive/00000464/01/Final.pdf Accessed 28 March 2011.

[8] M. Dahan, M. Thomas, E. Roberts, A. Seth, T. Urban, D. Walling, J. Boisseau, Grid portal toolkit 3.0 (gridport), in: Proceedings. 13th IEEE International Symposium on High performance Distributed Computing, 2004, pp. 272–273.

[9] J. Novotny, M. Russell, O. Wehrens, Gridsphere: a portal framework for building collaborations: Research articles, Concurr. Comput. : Pract. Exper. 16 (2004) 503–513.

[10] G. He, Z. Xu, Design and implementation of a web-based computational grid portal, in: Web Intelligence, 2003., pp. 478 – 481.

[11] G. Singh, E. Deelman, G. Mehta, K. Vahi, M.-H. Su, G. B. Berriman, J. Good, J. C. Jacob, D. S. Katz, A. Lazzarini, K. Blackburn, S. Koranda, The pegasus portal: web based grid computing, in: Proceedings of the 2005 ACM symposium on Applied computing, 2005, SAC '05, ACM, New York, NY, USA, 2005, pp. 680–686.

[12] M. Thomas, S. Mock, J. Boisseau, Development of web toolkits for computational science portals: The NPACI hotpage, in: Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing, 2000, HPDC '00, IEEE Computer Society, Washington, DC, USA, 2000.