A lightweight Web interface to Grid scheduling systems

Christopher Parker
University of Cape Town
Rondebosch
South Africa
cparker@cs.uct.ac.za

Hussein Suleman
University of Cape Town
Rondebosch
South Africa
hussein@cs.uct.ac.za

ABSTRACT

Grid computing is often out of reach for the very scientists who need these resources because of the complexity of popular middleware suites. Some effort has gone into abstracting away these complexities using graphical user interfaces, some of which have been Web-based. This paper presents a lightweight and portable interface for Grid management, that is made possible using recent advances in dynamic technologies for Web applications. Case studies are presented to demonstrate that this interface is both usable and useful. An analysis of usage then highlights some positive and negative aspects of this approach.

Categories and Subject Descriptors

H.1.2 [**Models and Principles**]: User/Machine Systems; D.2.2 [**Software**]: Interoperability

General Terms

Grid Computing Usability

Keywords

Grid Computing, Scheduling, AJAX, Usability

1. INTRODUCTION

Grid computing, although widespread, is still a relatively young field in Computer Science as it was only formalised by Foster and Kesselman [14] in the early 2000s. As with any new field, however, Grid computing is not without its shortcomings. Since Grids are inherently distributed systems with a strong focus to date on security, reliability and functionality, little attention has been paid to usability [10]. Users of such environments are left with the burden of a steep learning curve and are forced to use low-level tools to accomplish the simplest of tasks. The success of Grid technology is therefore largely dependent on the effective usability of such systems. High-level tools and interfaces that abstract low-level complexity from users are vital to the long-term uptake of Grid technology.

In order to alleviate some of the problems plaguing Grid systems, this paper describes a high-level Web interface to a Grid environment built by taking a lightweight design approach. By assuming such an approach, the problems plaguing traditional Web applications, such as responsiveness, bandwidth efficiency, their level of dynamic interaction and scalability, are minimized. Due to the amount of computation, data transfer and interactivity expected from an interface operating in a Grid environment, a high degree of scalability and bandwidth efficiency is important. Furthermore, a lightweight approach differs from traditional Web development techniques in that it provides users with the feel expected from Web 2.0 applications. Social networking sites such as Facebook [3] and applications such as Gmail [5] and Flickr [4] have increased user awareness of Web 2.0. Finally, a lightweight approach provides a structured, flow-oriented and uninterrupted experience to users. Traditional Web applications have assumed a task-oriented design where users are forced to focus on one task at a time. A lightweight dynamic approach assumes a resource- or object-oriented design that more closely maps to Grid computing.

The rest of this paper is outlined as follows: Section 2 provides a short overview of Grid computing followed by a discussion on existing Grid interfaces in Section 3 and lightweight Web technologies in Section 4. These sections are then drawn together in Section 5 with an overview of the architecture and functionality of the Web interface designed as part of this research. Finally an overview of some case studies is presented in Section 6. Section 7 provides some concluding remarks.

2. GRID COMPUTING MIDDLEWARE SCHEDULERS

The most common Grid middleware, also considered to be the de-facto standard for Grid middleware, is the Globus Toolkit [15]. Globus provides a set of tools that enable access to a Grid but does not provide any mechanism for job scheduling. For this research, a computational campus Grid was built using Grid-aware scheduling middleware.

Many Grid systems and toolkits, such as the Globus Toolkit [15], are merely underlying fabrics for connecting various parts of the Grid together, ensuring secure communications between these parts and providing abstractions of underlying complexities inherent in the core Grid architecture. Many Grid tools rely on third party solutions for scheduling at the execute node or cluster level. This architectural characteristic allows for a wide range of schedulers to be employed based on the needs of a particular organisation. For this research, and to test the capabilities of the tools developed, the Condor [24] and IBM Loadleveler [20] scheduling systems were used.

2.1 Condor

The Condor batch scheduling system, developed at the University of Wisconsin at Madison [24], is an opportunistic system that makes use of spare compute cycles on networked Condor-enabled nodes to complete work submitted by a user (on condition that certain user specified resource requirements are met). The Condor system supports a wide variety of job types but only vanilla (binary) jobs were utilised.

Once the Condor system has found a suitable candidate machine on which to execute a job, the job is packaged, sent across to this machine, sandboxed and then begins execution. Upon completion, result files are copied back to the submission host. All these operations are taken care of transparently by the underlying Condor system and no user intervention is required once the job is submitted. For the purposes of Grid computing, the Condor flocking system [13] was configured for use in our test Grid. Flocking allows a user to submit a job to his/her own compute nodes and have them run on (or flock to) compute nodes owned by others in a different administrative domain. Furthermore, Condor transparently handles job or machine failures inherent in large distributed systems such as Grids by restarting and/or migrating failed jobs.

2.2 IBM Tivoli Workload Scheduler Loadleveler

Loadleveler [20], like Condor, is a batch scheduling system that takes advantage of idle Loadleveler-enabled machines on a network. Both Loadleveler and Condor are very similar in their architecture and provide similar services. Loadleveler, like Condor, has a flocking-type capability termed the Loadleveler multi-cluster.

Condor and Loadleveler are by no means the only batch schedulers available. Other scheduling systems such as LSF [27], PBS [19], Torque [29] and Sun N1 Grid Engine [16] are equally capable. PBS is no longer free and its open derivative, OpenPBS, is no longer under development and therefore was not taken into consideration. It should be noted, however, that many researchers still make use of this system. Torque is an open-source and actively developed version of PBS, but this research focused only on Condor and Loadleveler but could be extended to include Torque as well. LSF, on the other hand, is proprietary software and requires licensing to run, and was therefore also not considered.

3. EXISTING GRID INTERFACES

Numerous Grid portal systems and development toolkits are available to enable high-level access to Grid computing environments. Toolkits such as Gridport [11] and containers such as Gridsphere [26] allow developers to quickly build and deploy Grid-based tools. The aim of these systems is to allow for a highly customizable, standards-compliant design by allowing developers to write small portlets that plug into a container.

The Gridport toolkit [11] 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.

4. LIGHTWEIGHT USER INTERFACES

With the increased popularity of Web2.0 over the past few years, a steady increase in sites utilising Web2.0 techniques has been observed [21]. The concept of the Web page, in Web2.0 terms, has been replaced with that of the Web application or “webapp”. A Web application differs from a Web page in terms of interactivity, its dynamic properties and most importantly, the fact that it is an application that is deployed within a Web browser.

Since the backend business logic of a typical Grid interface requires mediation between the Grid software and a user’s browser, the overhead imposed by an interface should be kept to a minimum. By requesting only the minimal amount of data required for a particular action, and relying heavily on the client-side interface engine, this goal can be achieved. The use of a combination of lightweight technologies therefore helps to create a more responsive, bandwidth efficient, dynamic and usable interface that is especially suited for use in a Grid environment.

Finally, due to the complexity of a Grid-based job, the use of a dynamic interface helps users to not lose sight of the task at hand by having to wait for subsequent screens to load after each step in the job specification process. This also reduces traffic between the browser and server once the application has been transferred, thereby providing a more responsive user experience.

This section will give an overview of technologies that aid in making Web applications lightweight and more efficient. Finally, a more in-depth discussion on AJAX, used in this research, will be given.

4.1 DHTML

Since Web2.0 applications require a high degree of interactivity and usability, traditional Web development has given way to a more dynamic approach [31] with technologies such as DHTML [28]. DHTML is a development paradigm and not a standalone technology as it incorporates existing technologies such as Javascript, CSS and HTML. The ability of DHTML to manipulate the Document Object Model (DOM) within the browser allows for a page to be updated dynamically and therefore does away with the need to refresh the entire page. This approach has significant benefits in terms of bandwidth efficiency between the browser and the Web server as DHTML is a client-side technology. Data is only transferred when a user makes a request in the form of an update in the browser.

4.2 AJAX

AJAX, or Asynchronous Javascript and XML, is a term first coined by Jesse-James Garrett in 2005 [9]. AJAX provides a development paradigm for building lightweight Web applications. A typical AJAX application makes use of technologies including: DHTML for event-driven components; HTML/XHTML, CSS, HTTP, server-side scripting, Javascript, XML/Document Object Model (DOM), and most importantly, XMLHttpRequest (XHR) [25].

AJAX's XHR calls, along with its ability to modify the DOM in the browser by way of DHTML, allow for page elements to be updated dynamically and hence do away with a full page refresh required by the traditional Web development paradigm. AJAX applications, once loaded, appear to be more responsive due to the minimal data transfer between the browser and the server and the ability to update only the necessary screen elements.

Technologies such as the Wireless Application Protocol (WAP) have illustrated how lightweight communication can be achieved by allowing asynchronous transaction requests [30] and thereby shortening total communication time and increasing responsiveness. In the Web environment, the WAP equivalent of lightweight communication is the XMLHttpRequest (XHR) [25] API. This API allows Web applications to make synchronous or asynchronous calls, the latter being more common, in order to transfer data between a client and Web server on demand. This behaviour decreases application response time and allows for dynamic data transfer.

Although AJAX is a relatively new concept, many popular websites and services currently make use of AJAX-based interfaces. One of the first such interfaces was Google Suggest [17]. This interface makes search engine query suggestions appear in a drop-down list as one begins to type the search term. Google has been successful in making AJAX techniques popular by incorporating this design approach into services such as Gmail. Many of the elements in the GMail mailbox are dynamically updated depending on the event a user triggers. Another well known service that makes extensive use of AJAX techniques is Google Maps [23]. In this application, data is dynamically transferred to the browser as a user exposes portions of off-screen maps.

AJAX has had application beyond classic webapps. Gozali et al. [18] outline an AJAX-based user interface to Open Public Access Catalogs (OPACs) for better usability and task support. Interactive network visualization tools created by Douitsis et al. [12] illustrate the use of AJAX in conjunction with SVG to create a more responsive and dynamic online tool.

This research is considering the application of lightweight user interfaces to the Grid domain.

5. DESIGNING A LIGHTWEIGHT GRID INTERFACE

5.1 Grid Environment

The test Grid built for this research is composed of three separate clusters, a handful of independent laboratory computers and a Grid server. This server runs the Web interface and also contains local installations of the two scheduling systems, discussed in Section 2. Jobs are pushed from the Grid server to the Grid-enabled clusters via the flocking and multi-cluster capabilities of Condor and Loadleveler respectively. In total, the test Grid was comprised of approximately 60 cores belonging to approximately 25 computers.

One of the chief concerns when designing this system was catering for multiple local schedulers in one interface. Since scheduling systems can become deprecated or simply be overtaken by better systems, it is important to ensure that a Web interface is able to accommodate any new scheduler by creating a language—and scheduler-free base architecture. Multi-scheduler support therefore ensures that local Grid environments consisting of many different scheduling systems, perhaps in different administrative domains, are controlled from one central location by providing a simple mechanism by which to select the scheduler of choice when submitting a job. By giving users the power to alter the base scheduling system, one allows for the submission of jobs to parts of the Grid not under direct control of one of the scheduling systems for example. If there are resources on one part of such a divided Grid that an application needs in order to execute, then being able to change the base scheduler becomes important. Since current scheduling systems are not interoperable, it is not possible to simply transfer jobs from one scheduling system to another. Another benefit of the multi-scheduler approach is that users only need to learn one procedure for submitting and monitoring jobs on a Grid.

5.2 Grid Interface Design Details

After careful consideration of various Web interface technologies, it was decided that the interface designed for this research project would be built by taking an AJAX approach. Since the aim of the research was to build not only a lightweight system, but also a system that has a high degree of interactivity and usability, an AJAX approach provides the framework necessary to realise these goals. The ZK (Zero-Kode) [8] AJAX-based toolkit was used for the development of the system presented in this paper. Reasons for choosing this toolkit include the large set of predefined widgets and adequate documentation.

Interviews were held with four senior scientists from different scientific domains, such as Chemistry and Physics, that actively make use of (High Performance Computing (HPC) resources. Data on application types, software tools, scheduling systems, dataset sizes, operating systems, architectures, run times and many other such metrics were gathered during these interviews. From this data, it was clear that parameter-sweep applications would be most useful to the scientists. Parameter-sweep applications are Single Instruction Multiple Data (SIMD) type applications where the same program or binary is used to process slightly different input data on each "sweep" of the application. An example of such an application would be a High Energy Physics application looking for patterns in data passed to it from a detector. Such data is always in the same format.

The final design considerations that were decided upon were the components that the interface was to support. These include Grid status, job creation, job submission, job query,

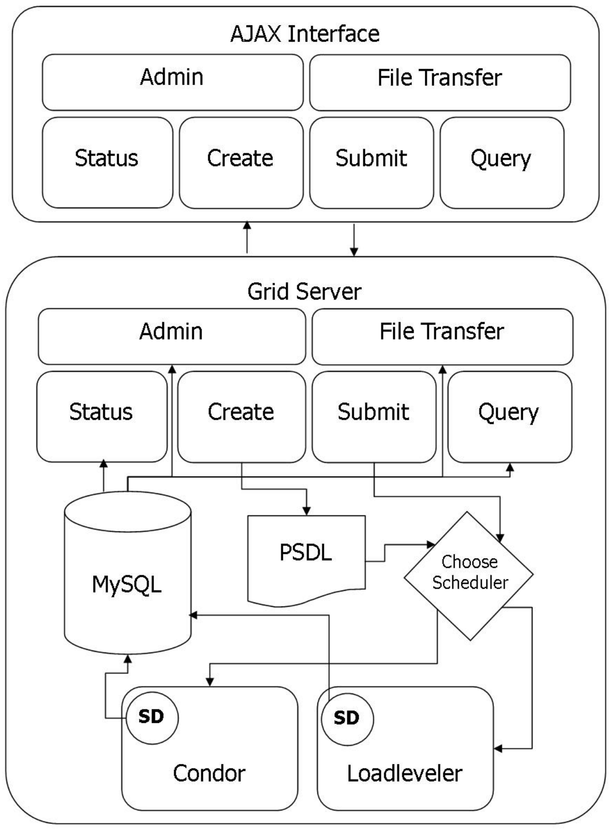


Figure 1: High-level system design

admin and file transfer components (see Figure 1). As can be seen from this figure, each of the components are split into client- and server-side components. The client-side is responsible for making requests to the server for new information as well as displaying the output of these requests while the server-side is responsible for receiving requests and generating the data to update the Web interface. An overview of each of these components is presented in the following sections.

5.3 System Status

The status component is responsible for providing up-to-the-minute information on the status of the computational Grid environment. As can be seen from Figure 1, each scheduler has a custom-built Status Daemon (SD) wrapper script which is executed at five minute intervals. These scripts wrap around the low-level command-line status utilities bundled with each scheduler. The purpose of these scripts is to gather information from all Grid machines on which the particular scheduler is installed, extract the relevant information from the wealth of status statistics and then populate a MySQL database with this information. The status component of the Web interface is then periodically updated by an XHR call to reflect the latest status of the Grid. It is important to note that only the status component is updated and a full page refresh is not required.

The status component contains a summary bar for each pool where an overview of the overall pool resource information is provided graphically (load, memory, diskspace and compute power). Furthermore, an expanded view provides details about which pool machines are available as well as their operating systems and architectures.

5.4 Job Submission

The job submission component allows a user to submit a job to the Grid in a two-step process, namely job creation followed by job launching.

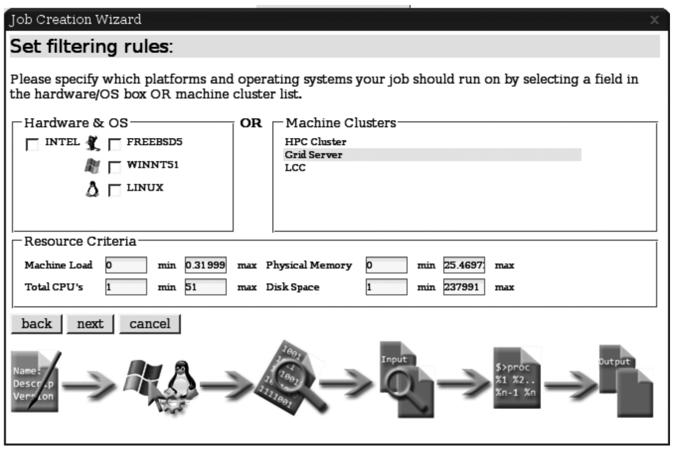
The first step in the job submission process is the job creation step. During this step a user specifies a job by following steps outlined in a wizard, or alternatively, loads an existing job template from file—the origins of which will be discussed shortly. The interface provides a seven step wizard that prompts users for various pieces of information as listed below:

- Job specific information (job name, description, etc.)
- Application specific information (similar to job information)
- Resource filtering (architectures, OSes, etc.)
- Executable selection
- Input file selection
- Input argument enumeration
- Output-specific settings

The wizard was designed to look and feel like wizards found in most desktop applications. By making use of the DHTML functionality present in AJAX, parts of this wizard can be hidden as new parts are revealed, thereby providing the illusion of the familiar guided wizard. Furthermore, parts of the wizard not required by the user are not loaded prior to its use, thereby reducing overall communication between the browser and the server and also reducing latency. This principle holds for many of the interface components. The job creation wizard, for example, is loaded for the first time only when a user selects this option from the main interface window.

There are two parts of the wizard that deserve special mention. The first of these parts is the filtering step. As can be seen from Figure 2, a user is able to select the platform and operating system their Grid application is to be executed upon by selecting from the set of available architectures and operating systems, or by selecting a set of clusters or pools from the groupbox to the right. Furthermore, the selected machines can be further filtered by applying a set of resource constraints. These values are automatically generated by the Grid interface and represent the lower and upper bounds for these values depending on the current Grid status.

The next part of the job creation process is the input argument enumeration component. Since it was decided that the interface would support parameter sweep type applications, an argument enumeration system was developed and


Job Creation Wizard

Set filtering rules:

Please specify which platforms and operating systems your job should run on by selecting a field in the hardware/OS box OR machine cluster list.

Hardware & OS

- INTEL
- FREEBSD5
- WINNT51
- LINUX

OR

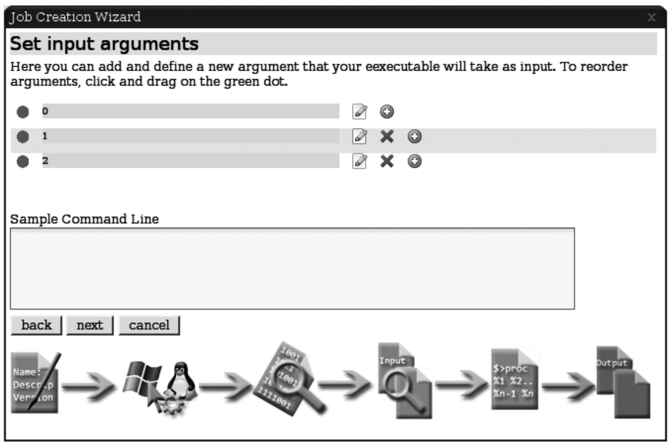
Machine Clusters

- HPC Cluster
- Grid Server
- LCC

Resource Criteria

Machine Load	0	min	0.31999	max	Physical Memory	0	min	25.4697	max
Total CPU's	1	min	51	max	Disk Space	1	min	237991	max

Figure 2: Job Creation : Filtering


Job Creation Wizard

Set input arguments

Here you can add and define a new argument that your eexecutable will take as input. To reorder arguments, click and drag on the green dot.

- 0
- 1
- 2

Sample Command Line

Figure 3: Job Creation : Argument Enumeration

incorporated into the wizard (see Figure 3). This system allows a user to specify or enumerate arguments that would traditionally appear only on the command-line, by using the Web interface. Each enumeration, which can be thought of as a row in a table, is then passed to the application as one "sweep".A user is able to specify arguments by creating a field and then selecting from one of four different types, namely, flag, number, multiple-file and single-file. A flag is simply a textual entry that is repeated for the number of arguments a user specifies and is repeated verbatim for each run of the application. The number type is a dynamically allocated argument for each run. A user specifies a minimum value, a maximum value and the incremental value at which the argument will advance. In order to handle input files, two file types are provided. The multiple-file type allows a directory of input files specified as part of the job creation process to be distributed over the entire application run. Finally, the single-file argument operates much like the flag argument; however, it specifies a single file to be included as an argument to each run. It is important to note that with n runs of an application, the user specifies an argument only once. If the application happens to require multiple arguments per run, the user simply adds a new field corresponding to each argument and the interface will populate the runs with the appropriate values. In other words, if some application requires 100 sub-jobs, where each sub-job requires 4 arguments, the user only creates four arguments and sets how the interface should distribute the arguments to each run by making use of argument types just mentioned. Futhermore, arguments can be reordered by use of a drag and drop interface built into the wizard. Furthermore, a sample command-line window is provided that allows users to see the final output of their argument selection in real-time. This window is dynamically updated each time a change to an argument is made.

Once the job to run has been specified and the job creation process is complete, the interface creates a job template file in the user's home directory. This template, specified in XML, is based on the Job Submission Description Language (JSDL) [6] which provides a standard, language-independent way to specify Grid-based jobs. JSDL, however, was deemed unsuitable for parameter sweep applications due to inadequate support for multiple argument specifications. In order to compensate for this shortcoming, a modified version of JSDL, termed Parameter Sweep Description Language (PSDL) was created.

Once a job has successfully been created, it can be launched on the Grid. Since a user can load a job template from file, the job creation and submission steps have been separated. In order to launch a job, the interface provides visual cues in a job notification panel to alert a user that the job is ready to be launched. Once the alert, in the form of a flashing icon, is clicked, the job submission process begins. The process starts by selecting the scheduler that the user has specified as the current default scheduler. A submission script associated with the chosen scheduler is loaded and executed with the PSDL template as input (see Figure 1).

The time taken to set up a job depends on the number of runs. If, for example, a job consisting of 100 individual runs of 50mb data each is submitted, the submission process will naturally take longer than, say, 100 runs of 10mb data each. Therefore, submissions containing gigabytes of input data take tens of minutes to submit since all input files are copied to a spool directory on the submit machine.

5.5 Job Querying

The job query component allows a user to browse all jobs previously submitted as well as monitor the status of jobs currently running or still in the run queue, delete jobs, and retrieve the output generated from completed jobs. A user can at any time choose to update the information displayed in this component. Once again, only the status information is retrieved from the server and all other interface components remain unaffected.

5.6 Other Interface Components

The four components described so far account for the main functionality present in the interface; however, mechanisms for file transfer as well as administration were deemed important. The administration system allows for the addition of users, the changing of user passwords as well as the ability to view the details of and delete all jobs submitted using the Web interface. Furthermore, the administration system allows for the addition of user quotas to the system. This is important in controlling who has access to the Grid, for how long and when such users are allowed access, thereby ensuring a fair and equitable share of Grid resources.

The final component of importance is the file transfer component, based on Webdav [22], which relies on functionality built into the Apache Tomcat container. The Webdav subsystem provides users with a quick and efficient way to bulk-upload files to their home directory on the submission system, by creating a network share on desktop operating system. Users can then transfer files from their desktop to the submission server using simple copy and paste. Since the interface deals with parameter sweep applications, which could typically consist of many hundreds of files, a Web-based upload system was not considered to be feasible for large numbers of files. Nevertheless, the core interface also has file upload capabilities built into an AJAX file browser for cases when users wish to upload or overwrite single files.

6. CASE STUDIES

In order to test both the performance and the ability of the Web interface to cater for different parameter sweep scenarios, a number of case studies were conducted. A set of realistic problems were chosen from well-defined application domains that are known to make use of high performance computing resources.

The whetstone benchmark, a classic computational performance benchmark, formed the first case study to be launched using the Web interface. The aim of this case study was simply to run the benchmark on all the execute nodes present on the Grid to ensure the correct operation of all the appropriate systems. For the second case study, a simple text indexer was run. The aim of the indexer is to build an inverted file for each word in a set of text files—an operation fundamental to search engines. The third case study involved the conversion of audio from `.wma` format to `.mp3` format. This study represents those applications requiring data conversion, an operation that is usually computationally intensive. The final case study involved the rendering of animations across the Grid. By making use of Blender [1], an open source rendering package, as well as animations from the open movie Elephant's Dream [2], this case study served to illustrate the diverse set of problems for which a parameter sweep-specific interface can be used.

6.1 Analysis - Issues

For both the audio conversion and distributed rendering case studies, an alternate submission mechanism was needed. The audio conversion required a series of sub-programs to be run in sequence in order to convert from `.wma` to `.wave` formats (requiring one program) and then from `.wave` to `.mp3` (requiring another program). Furthermore, both these case studies required shared object files to be packaged along with the binaries in order to run on platforms where certain software, such as Python (required by Blender), was not installed.

Since the interface takes only one executable per job, it was necessary to write a bash script which would be able to execute the jobs described so far. In the case of the distributed rendering case study, instead of specifying the Blender binary as the executable, the bash script was specified as the executable and the Blender binary, along with all other required files, was packaged into a tar archive and passed to the script as a single-file argument in the interface. The script, shown below, therefore serves as a wrapper for the job.

```
#!/bin/bash
tar xvfz inputFiles.tar.gz
export LD_LIBRARY_PATH=.
./blender -b input.blend -x 1 -o \
output -F MOVIE -s $1 -e $2 -a
rm blender lib* inputFiles.tar.* \
input.blend
```

In this script, $1 and $2 represent numerical frame range values passed to the script from the interface. For each run of a job, a different set of frame ranges are passed to the script with the same input file. This input file contains all the object files (referred to by `LD_LIBRARY_PATH`) as well as the animation to be rendered and the Blender binary.

6.2 Analysis - Performance

For each case study conducted a range of timing information was gathered to gain insight into interface and scheduler performance. Summarised in Table 1, these results include the time taken to render the PSDL XML document after the job creation step; the cumulative spool file size for all subjobs; the number of runs of the application; the time taken to submit the job to the scheduler (including copying of input files and executables to the spool file); the approximate real compute time (excluding scheduler negotiation); the total run time of the job (including scheduler negotiation) and the time taken for the job to execute on a single CPU. For all case studies, all machines on the test Grid (approximately 50 Linux-based cores) were utilised. Since the time taken for certain scheduling operations is variable, approximate timing values are indicated with a tilde.

The results show that the time to generate the PSDL document is negligible for both small and large jobs. Jobs such as the audio converter, complex in terms of the number of runs as well as input arguments, take only three seconds to generate. Since the PSDL file for a job this large contains more than 8000 lines, this time is acceptable due to the speed at which Xerces [7] can extract and generate XML data. The time to submit such a large job, however, is noticeable. In Table 1, submit times are shown for all four case studies. It was found that the time to submit a job is, unsurprisingly, directly proportional to the total data size of the job. This is due to the scheduling system creating a spool directory for each subjob into which it places the executable as well as any input files required by the job. For a large job such as the audio converter the submission process took approximately 17 minutes due to this copying procedure. It was observed that no jobs are scheduled until the entire job has been spooled. This observation is a scheduler-specific limitation and futher investigation is needed into determining its cause.

The process of submitting a job, unfortunately, renders the Web interface unusable until after the job has been submit-

Case Study	PSDL Gen.	Data Size	Runs	Submit	Grid Real	Grid Total	Speedup
Whetstone Benchmark	0.258	550 Kb	50	1.487	˜1867	˜2248	˜20.35
Text Indexer	1.855	89.78 Mb	500	25.2	˜1649	˜1024	˜6.67
Audio Converter	3.000	4750.13 Mb	500	1036	˜1718	˜2438	˜21.81
Distributed Rendering	0.726	783.21 Mb	89	106.736	˜1320	˜1680	˜8.29

Table 1: Case study performance data (time reported in seconds)—Speedup = Serial Time / Grid Total Time

ted, due to an interface instance running as a single thread. For small jobs this is not a problem, but for large jobs such as the one just mentioned, this is a severe limitation. It is possible to overcome this limitation but this has not currently been implemented. By having the interface delegate the job submission process to a separate process on the server, for example, and by removing the submission logic from the interface in its entirety, this problem can be overcome. Futhermore, by making use of AJAX, the interface can be set to display a message as soon as the job has been submitted successfully.

The results also show the speedup from utilising the Grid as opposed to a single machine. The theoretical speedup for the test Grid should be of the order of 50 times more than a single machine due to 50 cores being utilised. The results, however, show that the maximum speedup attained was at most 20. There are many reasons for this, the first having to do with scheduler negotiation time. Since it can take several minutes to identify a machine on which a job is able to run (see Table 1), there is an idle period in which no work is being done. The main reason for this in the Grid used for this research, is that Condor flocking was enabled. The flocking process has to obtain information on many clusters as well as schedule new jobs to these clusters. It was observed that the scheduling process took much longer on the flocking-enabled Grid as opposed to a single cluster installed with the Condor scheduler. It was also noted that jobs begin to execute on some flocked clusters before others, indicating that there may be a delay in receiving information from these latter clusters. This delay is a possible reason for the lengthy idle times observed, although further investigation is required. The interface is therefore in no way responsible for these performance issues.

The time taken to transfer large volumes of data, both to and from remote hosts, affects the speedup. The results verify the theoretical size/time Grid tradeoff in that the longer a job runs, the less overhead in terms of negotiation and file transfer will be incurred relative to the total running time of the job itself. Although the use of a Grid has benefits for jobs that run for short periods of time, a significant speedup will only be noticed if thousands of such jobs are to be run as pre-startup negotiation tends to take longer than post-startup negotiation.

6.3 Analysis - Feasibility of AJAX

An AJAX-based approach to Web application development promises to enhance many facets of Web applications from system efficiency to usability. The cost at which such improvements are achieved must, however, be considered.

Firstly, the development time for complex AJAX-based Web applications can be shortened by using toolkits such as ZK that deal with browser details and present the programmer with a simple and familiar desktop- or Web-metaphor API.

The speed of AJAX-based Web applications is often inversely-related to the functionality provided by the toolkit. Various optimisations are possible to deal with this, including the selective and on-demand inclusion of libraries and interface elements. Such techniques were used in this study. A production system would typically include hand-optimised AJAX code.

The download size of the application was larger than a typical Web page, but this is compensated for over time because further pages are not loaded from the server—only raw data updates are transferred over the network. Due to the large amount of information that Grid jobs produce, an AJAX-based interface effectively reduces the amount of bandwidth required to update and display this information, thereby reducing traffic in networked environments.

Furthermore, due to the high frequency of updates to the status of Grid jobs and Grid resources, the dynamic properties of interfaces built using the AJAX approach ensure that data displayed in the interface is always fresh.

While there are potential pitfalls, most of these can be addressed adequately. AJAX has thus been demonstrated to be a feasible approach, in this study, for the development of a lightweight Grid scheduling interface.

7. CONCLUSION

Grids have traditionally been controlled and used through relatively low-level interfaces. This study attempted to create a higher-level interface in the context of recent advances in Web technology.

A proof-of-concept lightweight Grid scheduler interface was developed using the AJAX bundle of technologies. The experience of the case studies confirms the usefulness of this type of abstraction for real problems, notwithstanding some complexity in current AJAX toolkits. However, as demonstrated, the interface presented to the end-user can be similar to that of desktop applications. Such an approach should thus contribute to the widespread adoption of Grid computing tools.

The tools developed during this research will be made publically available once all changes suggested during the user evaluations as well as some improvements have been made.

Ongoing work is focused on controlled usability studies and additional performance measurements, to provide further insights and evidence in support of the applicability of AJAX in such applications.

8. ACKNOWLEDGMENTS

This work is based upon research supported by the National Research Foundation.

9. REFERENCES

[1] Blender, free open source 3d content creation suite. WWW Page, April 2008. http://www.blender.org/.

[2] Elephant's dream the open movie. WWW Page, April 2008. http://www.elephantsdream.org/.

[3] Facebook. WWW Page, April 2008. http://www.facebook.com/.

[4] Flickr photo sharing. WWW Page, April 2008. http://www.flickr.com/.

[5] Google mail. WWW Page, April 2008. http://www.gmail.com/.

[6] Job submission description language (jsdl). WWW Page, Aptil 2008. http://www.ggf.org/documents/GFD.56.pdf.

[7] Xerces java parser. WWW Page, April 2008. http://xerces.apache.org/xerces-j/.

[8] The zk ajax toolkit. WWW Page, April 2008. http://zkoss.org/.

[9] Apadtive Path Corporation. Ajax: A new approach to web applications. WWW Page, April 2008. http://www.adaptivepath.com/publications/essays/archives/000385.php.

[10] B. Beckles, V. Welch, and J. Basney. Mechanisms for increasing the usability of grid security. *International Journal of Man-Machine Studies*, 63(1-2):74–101, 2005.

[11] M. Dahan, M. Thomas, E. Roberts, A. Seth, T. Urban, D. Walling, and J. R. Boisseau. Grid portal toolkit 3.0 (gridport). In *Proc. 13th International Symposium on High-Performance Distributed Computing (13th HPDC'04)*, pages 272–273, Honolulu, Hawaii, USA, June 2004. IEEE Computer Society.

[12] A. Douitsis and D. Kalogeras. Interactive network management visualization with SVG and AJAX. In *LISA*, pages 233–245. USENIX, 2006.

[13] X. Evers, J. F. C. M. de Jongh, R. Boontje, D. H. J. Epema, and R. van Dantzig. Condor flocking: Load sharing between pools of workstations. In *NLUUG Voorjaarsconferentie*, pages 111–126, May 1994.

[14] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organization. *The International Journal of High Performance Computing Applications*, 15(3):200–222, Fall 2001.

[15] I. T. Foster. Globus toolkit version 4: Software for service-oriented systems. *J. Comput. Sci. Technol*, 21(4):513–520, 2006.

[16] W. Gentzsch. Sun grid engine: Towards creating a compute power grid. In *CCGRID*, pages 35–39. IEEE Computer Society, 2001.

[17] Google Corporation. Google suggest. WWW Page, April 2008. http://www.google.com/webhp?complete=1.

[18] J. P. Gozali and M.-Y. Kan. A rich OPAC user interface with AJAX. In E. M. Rasmussen, R. R. Larson, E. Toms, and S. Sugimoto, editors, *JCDL*, pages 329–330. ACM, 2007.

[19] R. L. Henderson. Job scheduling under the portable batch system. *Lecture Notes in Computer Science*, 949:279–??, 1995.

[20] IBM Corporation. Ibm tivoli workload scheduler loadleveler. WWW Page, April 2008. http://www-03.ibm.com/systems/clusters/software/loadleveler/index.html.

[21] M. Jazayeri. Some trends in web application development. In *FOSE '07: 2007 Future of Software Engineering*, pages 199–213, Washington, DC, USA, 2007. IEEE Computer Society.

[22] E. J. W. Jr. and M. Wiggins. WEBDAV: IETF standard for collaborative authoring on the web. *IEEE Internet Computing*, 2(5):34–40, 1998.

[23] R. M. Lerner. At the forge: Google maps. *Linux Journal*, 2006(146), June 2006.

[24] M. J. Litzkow, M. Livny, and M. W. Mutka. Condor - A hunter of idle workstations. In *ICDCS*, pages 104–111, 1988.

[25] M. Mahemoff. *Ajax Design Patterns*. O'Relly Media, 2006.

[26] J. Novotny, M. Russell, and O. Wehrens. Gridsphere: a portal framework for building collaborations. *Concurrency and Computation: Practice and Experience*, 16(5):503–513, 2004.

[27] Platform Computing Corporation. LSF product information. WWW Page, 1996. http://www.platform.com/.

[28] J. A. Ramalho. *Learn advanced HTML 4.0 with DHTML*. Wordware Publishing, pub-WORDWARE:adr, 1999.

[29] G. Staples. TORQUE - TORQUE resource manager. In *SC*, page 8. ACM Press, 2006.

[30] H. Ueno, N. Ishikawa, H. Suzuki, H. Sumino, and O. Takahashi. Performance evaluation of WAP and internet protocols over W-CDMA networks. *Cluster Computing*, 8(1):27–34, 2005.

[31] H. Weinreich, H. Obendorf, E. Herder, and M. Mayer. Not quite the average: An empirical study of web use. *TWEB*, 2(1), 2008.