PBSWEB: A WEB-BASED INTERFACE TO THE PORTABLE BATCH SYSTEM

GEORGE MA PAUL LU

Department of Computing Science
University of Alberta
Edmonton, Alberta, T6G 2E8
Canada
{george|paullu}@cs.ualberta.ca

Abstract

The resource managers (e.g., batch queue schedulers) used at many parallel and distributed computing centers can be complicated systems for the average user. A large number of command-line options, environment variables, and site-specific configuration parameters can be overwhelming. Therefore, we have developed a simple Web-based interface, called PBSWeb, to the Portable Batch System (PBS), which is our local resource manager system.

We describe the design and implementation of PBSWeb. By using a Web browser and server software infrastructure, PBSWeb supports both local and remote users, maintains a simple history database of past job parameters, and hides much of the complexities of the underlying scheduler. The architecture and implementation techniques used in PBSWeb can be applied to other resource managers.

Keywords: *job management, batch scheduler, Portable Batch System (PBS), Web-based GUI, cluster and parallel computing*

1 Introduction

Parallel and distributed computing offers the promise of high performance through the aggregation of individual computers into clusters and metacomputers (e.g., wide-area networks of computers). Fast nation-wide networks also make it possible for users to remotely access high-performance computing (HPC) resource providers (RP). However, that potential cannot be fulfilled without the proper software infrastructure to access the RPs in a convenient and efficient manner.

One of the most basic and common tasks for a user of an RP is submitting a job to the local resource manager. Although systems such as the Portable Batch System (PBS) [9] and Load Sharing Facility (LSF) [8] automate CPU and resource scheduling, the many command-line options, scriptable parameters, and different tools present a steep learning curve for the typical user.

Although tools and systems to support application *development* (rightly) receive a lot of research attention, the problem of application *execution* and management is often overlooked. In a typical workgroup, a small number of developers may write the code (or install downloaded code), but everyone must execute the applications, whether it is for testing or production runs. In other words, running applications is the common case. Also, the same application is often run with many different parameters, and possibly by different members of the same workgroup. Customizing, sharing, and revision control of the job control scripts can quickly become unwieldy.

To simplify the common tasks of submitting jobs to a resource manager, re-running jobs, and monitoring jobs in queues, we have developed a Web-based interface to PBS called PBSWeb. The system provides a job history database so that previous job scripts can be re-used as-is or with small changes. We feel that the combination of functionality in PBSWeb fulfills a need in the user community.

More importantly, the architecture of PBSWeb is designed to evolve and support the emerging model of wide-area metacomputing with remote users, transparent job placement across many different RPs, and end-to-end workflow management. Each of these enhancements involves the solution of outstanding research problems. PBSWeb will be the development and evaluation framework for our research in these areas.

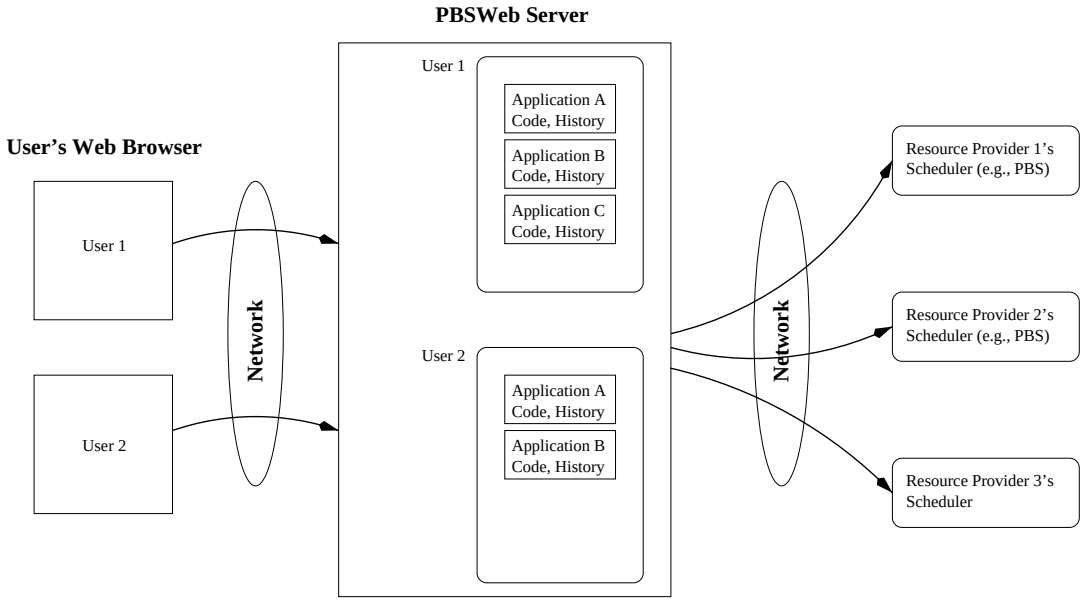

Figure 1. PBSWeb: Architecture

1.1 Portable Batch System

The Portable Batch System (PBS) is one of a number of different batch queue schedulers for processor resources. Resource managers and schedulers are software systems that allocate resources to different user requests (i.e., jobs) while attempting to maximize resource utilization and minimize interference between different jobs. Individual users submit jobs to PBS, the scheduler enqueues the jobs of the various users, and jobs are executed as the requested resources become available and according to the job's queue priority.

In part, PBS is a popular system because it is powerful, customizable, and it can be downloaded free of charge. For example, the Multimedia Advanced Computational Infrastructure (MACI) project [7] uses PBS to manage two SGI Origin 2000 multiprocessors (with a total of 112 processors) at the University of Alberta, and a cluster of Alpha-based workstations (with 130 processors) at the University of Calgary.

Although PBS has many technical merits, it can be a difficult system to learn and use. The system consists of a number of different programs (e.g., `qsub`, `qstat`, `qdel`, `qhold`, `qalter`), each with many different command-line options and configuration parameters. Furthermore, jobs (or runs) are submitted to the system in the form of a job control script containing per-job parameters. Each run requires its own job control script, which leads to the problem of how the various scripts can be easily modified and shared between different users with revision control.

`xpbs` and `xpbsmon` are two graphical user interfaces (GUI) distributed with PBS. Although these GUI tools are easier to use than command-line programs, they do not address the problem of managing job control scripts.

1.2 Overview

We begin with a description of the PBSWeb architecture and how it is designed to support remote access, to provide user authentication, and to maintain job control script histories. A detailed description of the PBSWeb interface and functionality is followed by an overview of how PBSWeb is implemented. Since PBSWeb is part of a larger C3.ca project to improve the sharing of high-performance computing resources [4], we then discuss the longer-term goals of the project and put it in context with related work from other groups.

2 PBSWeb

We have built a prototype front-end interface to PBS. Beta testing began in August 2000, with a wider deployment among MACI users to follow. By layering on top of PBS, we hope to make resource management and resource sharing more convenient in the common case.

PBSWeb is intended to simplify the task of *submitting* jobs to RPs that are controlled by a scheduler. It is assumed that the application source code itself is already developed and now the user would like to run jobs. If the user is only running existing application code, then he does not need to

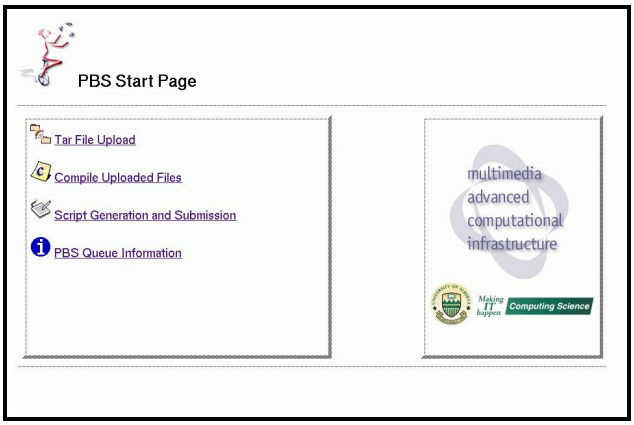

Figure 2. PBSWeb: Main Page

know anything about interactive shells or how to develop code. PBSWeb makes it easy for remote users to access an RP without having to share filesystems and without having to log onto the system just to submit a job. PBSWeb also maintains a per-user and per-application history (and repository) of source code files, jobs submitted to PBS, and past preferences for various user-selectable PBS parameters (e.g., time limits, email addresses) (Figure 1).

To begin, an authorized user connects to the relevant Web server. The server does not have to be on the same system as where the jobs will eventually run. One PBSWeb server can be the front-end to several different RPs or each RP can run its own instance of the PBSWeb server. After supplying a valid username and password, the main page is presented (Figure 2). Since the interface is based on Web pages, there is no need to understand site-specific operating system configurations.

To run a new application, the user must first upload the source code through PBSWeb. To submit a job using an existing application, the user selects the previously uploaded source code from a menu and submits the job from a Web page. PBSWeb simplifies the building (i.e., compiling) of the executable code on the target machine and the selection of PBS parameters either based on the user's history (comparable to command histories in Unix shells such as `tcsh`) or reasonable system defaults.

Four basic functions are *currently* supported by PBSWeb:

1. Upload source code in a tar file.
2. Compile source code already uploaded to PBSWeb.
3. Submit a job to PBS (with PBSWeb assisting in writing the job script).
4. Check on jobs in the PBS queue(s).

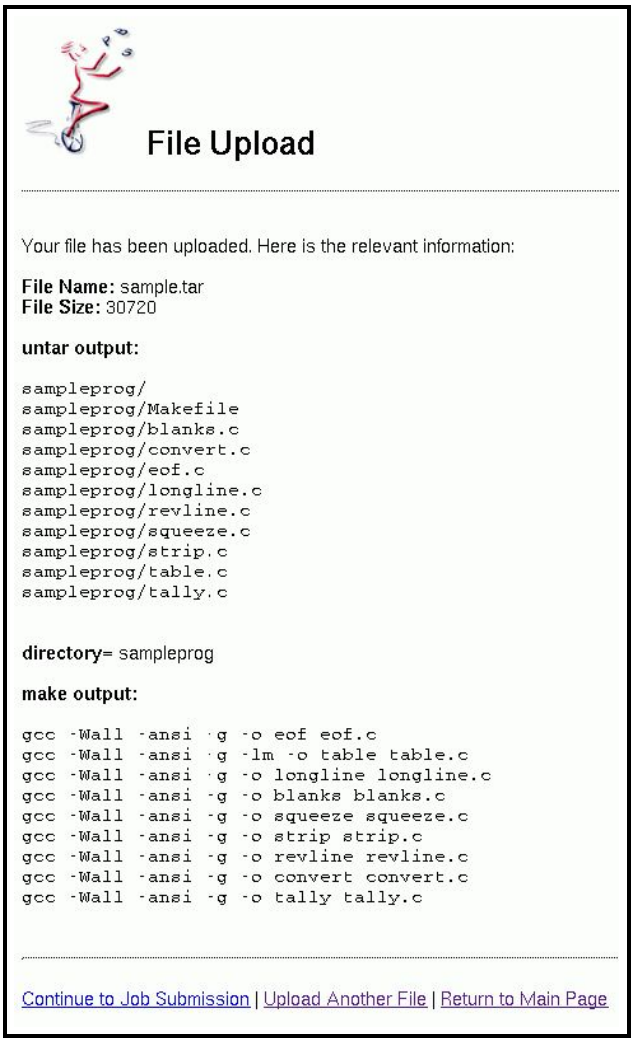

Figure 3. PBSWeb: Upload Tar File and Make

We expect that functions (3) and (4) will be the most frequently used.

To upload a tar file, a standard Web browser file selector is used to specify the file (not shown). By default, once the tar file is sent from the browser to the PBSWeb server, it is untared and a `make` command is issued. The output is reported back to the user (Figure 3). At this point, the user can proceed with submitting a job, upload another tar file, or return to the main page.

PBS (and similar systems) allow the user to control the job using a job script model. Therefore, the user may enter the commands for the script in the `Execution Commands` text field (Figure 4). For each application (i.e., tar file) that has been uploaded, PBSWeb remembers the recently executed commands so that the user does not always have to reproduce and debug complicated sequences

of operations. Instead, the user may select the commands using the `Command History` pop-up menu. If the application has already been executed using PBS, then the previous job scripts can also be selected using a pop-up menu (not shown).

Whether composing a new job script or modifying a previous job script, the user is free to select among any of the valid parameters to PBS. For parameters with only a few valid options, radio buttons are used, as shown for the job queue options of `dque`, `Sequential`, and `Parallel` (Figure 4). Therefore, the user does not have to memorize all of the valid options for, say, the job queue.

More complicated parameters that cannot be selected from a menu, such as the name of the job and the email address to notify upon job completion, are entered using text fields. Whenever possible, the text fields are initially filled in with reasonable default values or values provided by the user in the past.

Parameters with many possible options, such as the `Number of processors to use` for the job, are selectable using a pop-up menu. The goal is to reduce the need to memorize what parameter values are valid and what values are good choices for defaults at a given RP.

When the job script parameters have been finalized, the user clicks the `Submit Job` button and PBSWeb calls the proper command (i.e., `qsub`) with the proper command-line arguments and job script (Figure 5).

Again, excluding situations where compiles fail or jobs terminate under abnormal conditions, it is possible for an authorized user to upload, configure, and submit jobs to PBS without ever logging onto the RP's system. Also, the user can check the status of jobs in the PBS queues using the PBSWeb interface (Figure 6).

Furthermore, with a properly designed makefile (which is, admittedly, non-trivial), it is possible for a user to utilize the same PBSWeb interface and tar file to run jobs on, say, the MACI SGI Origin 2000s in Edmonton or the MACI Alpha Cluster in Calgary. In the future, the PBSWeb interface will be able to monitor the machine loads in Edmonton and Calgary and automatically (with the user's permission) run a message-passing job on either the SGI or cluster, depending on whichever will give the fastest turnaround time.

3 Implementation of PBSWeb

A Web-based approach was taken in designing the PBSWeb interface because Web browsers are pervasive and are a consistent and platform-independent way of interfacing with PBS. HTML documents, an Apache server, and Common Gateway Interface (CGI) scripts written in Perl, are the basic elements of the implementation of PBSWeb.

Again, there are four main operations in PBSWeb and each is handled by a different set of CGI scripts:

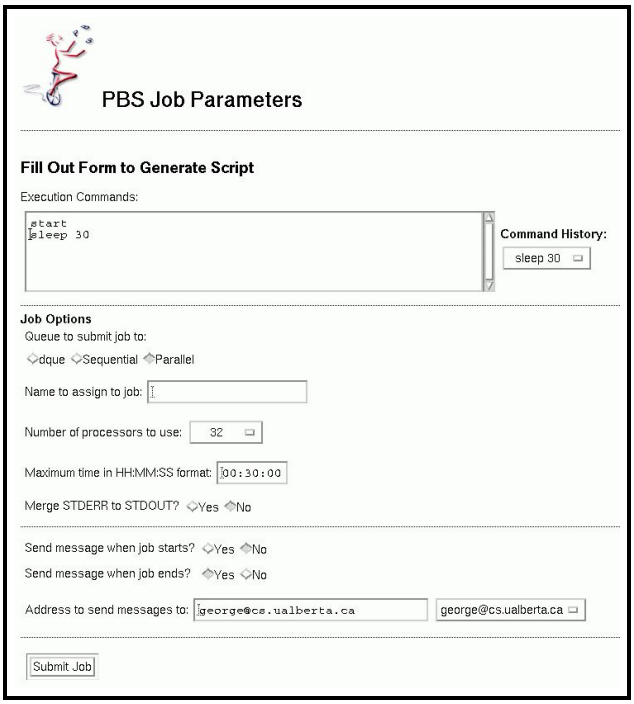

Figure 4. PBSWeb: PBS Job Submission and Script

1. Upload files
2. Compile programs
3. Generate scripts and submit jobs
4. View the queue status of a host

First, the file upload operation is intended, typically, to upload tar files of program source code. Uploaded files are placed in a subdirectory (`pbswebdir`) of the user's home directory, which is reserved for PBSWeb's use. The tar file is then extracted to a separate subdirectory of `pbswebdir`. Therefore, the user still requires an account and home directory at the RP to access secondary storage.

Second, program compilation is handled with a simple `make` command. Thus, successful program compilation is dependent upon the user supplying a suitable makefile. If the user does not choose to compile the source code directly after file upload and extraction, compilation can be performed at a later time by choosing the compile programs operation.

Third, the script generation operation is implemented using a HTML-based form. Here the user can specify various job options, such as the number of processors to use and the queue to which the job is to be submitted, simply and directly, without having to remember the complicated syntax of a PBS script. Also, the form gives sensible

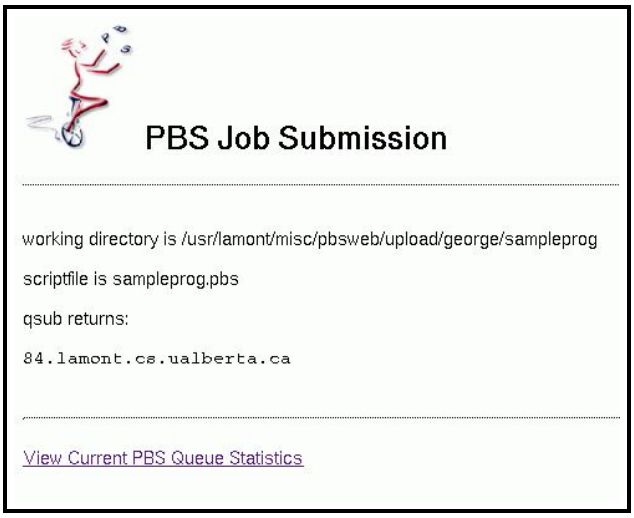

Figure 5. PBSWeb: Job Submitted

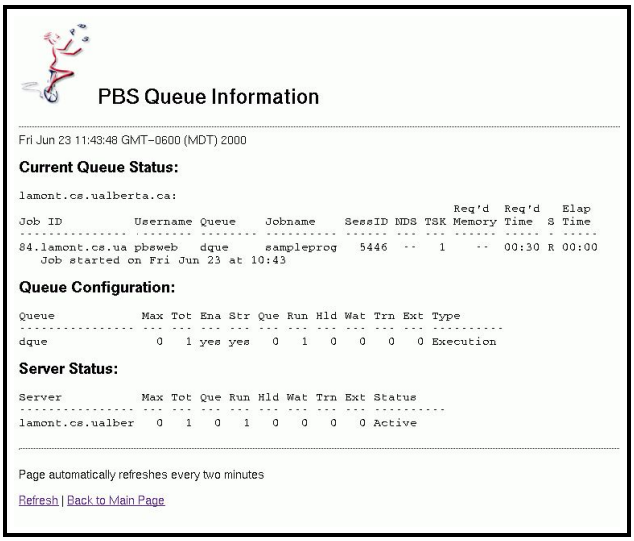

Figure 6. PBSWeb: Queue Status

default values for the most commonly used PBS job options. After the PBSWeb form has been suitably filled out, the data is sent to a CGI script, which generates a valid PBS script. PBSWeb then submits the generated script on the user's behalf to the specified PBS queue, or the default queue if a queue is not specified.

PBSWeb does more than simply provide a template form to generate PBS job scripts; the system also keeps a history of the job option parameters and executed commands. Thus, each PBSWeb user has a custom form which contains a history of previous jobs. The user can recall previously-generated script files and use them as a base for the next job submission. Of course, if a more experienced PBS user wishes, he may upload a custom script file and use it to submit a job or edit the uploaded script and then submit it.

The multi-user functionality of PBSWeb is accomplished by using the Secure Shell (`ssh`). When a user first creates a PBSWeb account, the user must copy PBSWeb's secure shell public identity key into the user's personal `authorized_keys` file. This allows PBSWeb to assume the identity of the PBSWeb user. When a new account is being created, PBSWeb does a secure shell login into the user's system account and creates the `pbswebdir` subdirectory for PBSWeb to work in. All of the user's uploaded files are stored in this directory, along with history files and information about the directory structure.

Each program is given its own subdirectory under the PBSWeb working directory. All of the script files generated by PBSWeb are stored in these program subdirectories. Thus, each one of the user's programs has its own unique history. When PBSWeb generates a custom job script form, it opens a secure shell session to the user's account, goes to the directory of the program that is to be executed, looks for the most recently used script, and loads the parameters of that script into the form as starting values. Also, the user can choose to load any of the previous scripts. When submitting a job script, PBSWeb opens a secure shell session to the user's system account and automatically issues a `qsub` command with the script file generated by the form script.

To decrease the execution time of the CGI scripts, some temporary files are created in the PBSWeb's home directory; fewer secure shell sessions to the user's account have to be opened and many of the intermediate steps are done within PBSWeb's own directory. For example, when a user uploads a file or runs a PBS script generation CGI, the files are first saved in PBSWeb's directory and then transfered to the user's `pbswebdir` by using secure copy (`scp`). After, the files are copied, PBSWeb deletes the related files in its own directory.

Although the secure shell provides a reasonable and safe mechanism to allow the PBSWeb server to compile, submit, and monitor jobs on behalf of a user, we are still investigating other options. In principle, the ability of PBSWeb to run *any* command as another user is too omnipotent. Ideally, a system that combines the authentication functionality of the secure shell and the per-executable permission control of, say, `sudo`, would be ideal.

4 Future Work

A number of desirable features are currently missing from PBSWeb. After beta testing in the MACI environment, it is our goal to make PBSWeb into an open-source development project.

On a metacomputing level, there is a need for com-

plete end-to-end workflow management so that multiple jobs can be co-ordinated to solve the overall problem. For example, a computation may involve pre-processing input data, a simulation, and post-processing of the results as three separate jobs. Each job is independently submitted to a batch scheduler, possibly at different RPs, but the jobs are linked by their inter-dependence.

In theory, the different HPC resources across a country can be harnessed for a single computational task. This concept of metacomputing [1] has attracted a lot of attention. Unfortunately, in practice, the lack of an appropriate infrastructure makes it difficult to transparently perform one phase of a multi-phase computation on, say, a cluster and another phase on a shared-memory computer. Thus, the painful reality is that sharing resources generally means that a user is allowed to log onto the different machines, manually transfer any needed data files between machines, manually start up the jobs, check for job completion and the integrity of the output, and repeat for each platform. The many manual and error-prone (both human and technological) steps required to do this usually means that a user always stays on one platform and "makes do." This defeats the purpose of sharing resources and can lead to some overloaded and other under-utilized platforms.

A long-term research goal is the design, implementation, and evaluation of a workflow manager for parallel computations. Consider the scenario where a computation is organized as a pipeline or a sequence of individual jobs. Ideally, if the pre-processing of input data is near-embarrassingly parallel, the cluster would be the best platform for the computation. Then, the communication-intensive phase of the computation can be performed on the parallel computer. From the user's point of view, a job is submitted to a workflow manager, which would then automatically initiate the first phase of the computation, transfer the intermediate output to the second hardware platform, and then initiate the second phase of the computation. The user is only interested in receiving the final output or an error report.

The reliability of the end-to-end workflow (i.e., start of entire computation to end of computation) is an important issue. Data validation between phases increases the reliability of the results by detecting corrupted data before it affects subsequent phases. For example, in our experience, although TCP/IP guarantees correct network transfers, lost connections and full file systems can easily (and silently) truncate files and corrupt the workflow. Simple validation checks at the application-level are needed. Abnormal process termination must be detected and subsequent jobs must be suspended until the situation is corrected. Intermediate data files must be logged and archived so that jobs may be re-started from the middle of the pipeline (i.e., automatic checkpoint and restart).

At the highest level of abstraction there should be a graphical Web-based interface that allows a user to communicate with the workflow manager. The user configures the workflow, specifies the input data, the computational constraints, and the phases using the user interface. At the lowest levels, the workflow manager co-ordinates the site-specific local schedulers and file systems at each computing site. The workflow manager receives a computational task in a manner similar to a batch scheduler. But, the manager is aware of the different phases of the computational task and it is aware of the distributed nature of the computing sites. The manager is configured to check for the local availability and integrity of the specified input data, it runs optional user-specified scripts to verify the pre- and post-condition sanity of the data, it submits the appropriate job to the local scheduler, it checks the integrity of the output, and then co-ordinates with the local scheduler at the next computing site to perform the next phase of the computation.

5 Related Work

There are a number of metacomputing research projects and systems, including Legion [6], Globus [5], Albatross [2], and Nimrod/G [3]. Baker and Fox survey some of the projects and issues [1].

Our project targets a higher-level of abstraction than most of the current projects. For example, we are not addressing the issue of how to use heterogeneous and distributed resources *within* a single parallel job. These runtime system issues are low-level problems. Instead, we focus on the issues of transparent data sharing and co-ordination *between* the jobs and sites. We feel that a high-level approach focuses the scope of the technical challenges and more directly addresses end-to-end issues of reliability and transparency. It is easier to adapt to missing data, network outages, and overloaded computers if a workflow of jobs spans multiple sites, instead if a monolithic parallel job spans multiple sites. Notably, Nimrod/G [3] shares these high-level design goals and supports a computational economy approach to resource sharing.

We also feel that a high-level approach is better able to exploit existing technologies and infrastructure and produce a usable system. Although some components of a workflow infrastructure already exist in the form of distributed file systems (e.g., AFS) and batch schedulers (e.g., PBS, LSF), they are not well integrated nor transparent. Setting up a distributed file system across many sites leads to a variety of problems with configuration and administration, thus it may not always be possible. Political and human factors in co-ordinating multiple sites must also be addressed with flexible and customizable policies in the workflow manager. However, if some sites share an AFS file system, that can be exploited by having the file system perform that data transfers under the control of the work-

flow manager. Similarly, batch schedulers have made great progress in scheduling individual jobs, but they are not designed to handle computational tasks requiring a sequence of jobs on different computing platforms and at different sites. There are research issues with respect to efficient global scheduling of tasks across a number of distributed computing sites, which would be an extension of existing work in optimized batch scheduling.

6 Concluding Remarks

In practice, effective high-performance computing and metacomputing requires a proper software infrastructure to support the workflow management of parallel computation. Currently, too many manual and error-prone steps are required to use HPC computing resources, whether within one RP or across multiple RPs.

Something as simple as submitting a job to a batch scheduler requires the user to write a job control script, define several environment variables, and call the right program with the right command-line parameters. Resource schedulers, such as the Portable Batch System, are powerful and necessary systems, but we feel that the learning curve is too high. We also feel that the system should automatically manage job control scripts so that it is easy to modify previous control scripts for new runs.

Towards these goals, we have developed an interface to PBS called PBSWeb. By exploiting Web-based technologies, PBSWeb makes it particularly easy to support both local and remote users using different platforms. PBSWeb also supports per-user and per-application job control histories and code repositories to make it easier to manage large sequences of production runs. In the future, the basic architecture and model of PBSWeb will be extended to support the automatic selection of computing resources and the co-ordination of computations that span multiple computing centers.

Acknowledgements

Thank you to José Nelson Amaral, Jonathan Schaeffer, and Duane Szafron for their valuable comments on this paper.

Thank you to the Natural Sciences and Engineering Research Council of Canada (NSERC), the Multimedia Advanced Computational Infrastructure (MACI) project, C3 (through a Pioneer Project), the University of Alberta, and the Canada Foundation for Innovation (CFI) for their financial support of this research.

References

[1] M. Baker and G. Fox. Metacomputing: Harnessing informal supercomputers. In Rajkumar Buyya, editor, *High Performance Cluster Computing: Architectures and Systems, Volume 1*, pages 154–185. Prentice Hall PTR, Upper Saddle River, New Jersey, 1999.

[2] H.E. Bal, A. Plaat, T. Kielmann, J. Maassen, R. van Nieuwpoort, and R. Veldema. Parallel computing on wide-area clusters: the Albatross project. In *Extreme Linux Workshop*, pages 20–24, Monterey, CA, June 1999.

[3] R. Buyya, D. Abramson, and J. Giddy. Nimrod/G: An architecture for a resource management and scheduling system in a global computational grid. In *Proceedings of the 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000)*, Beijing, China, 2000.

[4] C3.ca. `http://www.c3.ca/`.

[5] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[6] A.S. Grimshaw and W.A. Wulf. The Legion vision of a worldwide virtual computer. *Communications of the ACM*, 40(1):39–45, January 1997.

[7] MACI. Multimedia advanced computational infrastructure. `http://www.maci.ca/`.

[8] Platform Computing, Inc. Load sharing facility. `http://www.platform.com/`.

[9] Veridian Systems. Portable batch system. `http://www.openpbs.org/`.