Security and History Management Improvements to PBSWeb (Draft version of HPCS 2001 Paper)

George Ma, Victor Salamon, and Paul Lu

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

July 17, 2001

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. By using a Web browser and server software infrastructure, PBSWeb supports both local and remote users, maintains a history 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.

Since our first description of the PBSWeb project [7], we have completely re-implemented the system to address three important deficiencies:

1. Instead of using the filesystem to manage job histories, we are now using the PostgreSQL relational DBMS to improve history queries and reliability.
2. Instead of clear text socket connections, we are using the Apache Web server with Secure Socket Layer to improve the security of communications between browser and server.
3. Instead of using Perl and raw HTML documents, we are using the PHP server-side scripting language to access the PostgreSQL database, to manage user sessions, and to create dynamic Web pages.

We describe the security and history management benefits of the new PBSWeb.

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

1 Introduction

The PBSWeb front-end to the Portable Batch System (PBS) [10] is designed to simplify the common tasks of submitting to, and monitoring jobs on, batch scheduled high-performance computing (HPC) systems [7]. Following the principle of "simplify the common cases", PBSWeb provides a Web-based point-and-click graphical user interface (GUI) to the powerful, but complicated, command-line interface of PBS.

More fundamentally, there is the need for a more unified and friendly approach to how computational scientists interact with HPC resource providers (RP). Currently, the process of submitting, monitoring, and re-submitting jobs to an RP requires knowledge of many different pieces of technology. For example, in the context of the Multimedia Advanced Computational Infrastructure (MACI) project [8], using our HPC systems requires at least some knowledge of Unix commands, a text editor (for scripts), either `ftp` or secure copy (for data transfers), and batch scheduler commands. Although the power user or code developer needs expertise in these areas, what about the computational scientist who just uses existing applications to run a series of experiments on the HPC system?

PBSWeb strives to provide a unified environment in which all of the scientist's interactions with the HPC system can be done via a single Web-based system. As described in our previous paper, PBSWeb can also can be accessed from any computer and operating system with a browser (i.e., practically any system). The user does not have to log onto the system (in the Unix sense), edit raw scripts, move data files via the command line, or read copious `man` pages. Most of the basic functionality of PBSWeb was implemented during the Summer of 2000 in a prototype system. However, there were (and are) a number of deficiencies in PBSWeb that needed to be addressed.

Recently, PBSWeb has been re-implemented using new Web technologies to address the issues of:

1. More convenient management of experiments.
2. Secure access to a remote resource provider.
3. Dynamic Web pages to provide more functionality.

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.

PBSWeb now uses a relational database management system (DBMS) to store and retrieve a history of job scripts. The system provides a job history database so that previous job scripts can be re-used as-is or with small changes. In the earlier version of PBSWeb, the so-called history database was a collection of files in a directory of the filesystem. The new approach with the DBMS makes it easier for several people to actively share the same PBSWeb account without risking data corruption. In the future, it will be possible to support more sophisticated queries on the job history.

The computational resources of MACI are designed to be shared by users at different geographical locations. In practice, most researchers use the resources at their home institution because of the extra difficulties of using remote resources. 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. One important step towards those larger goals is to ensure secure access to the PBSWeb servers and the HPC systems. A proper production system requires security. Consequently, our Apache Web server for PBSWeb now uses the Secure Socket Layer (SSL) to avoid sending any sensitive data or passwords in clear text. We use standard Apache and SSL packages without any changes.

Lastly, we wanted the Web pages of PBSWeb to be more dynamic than in the previous prototype. For example, when a previous job script is selected from the history database, the rest of the page should immediately and dynamically show the parameters of the job script. Also, the system should track the user's session to provide more stateful and context-sensitive responses.

1.1 Portable Batch System

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) [10] and Load Sharing Facility (LSF) [9] automate CPU and resource scheduling, the many command-line options, scriptable parameters, and different tools present a steep learning curve for the typical user.

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 MACI project [8] uses PBS to manage three SGI Origin 2000/3000 multiprocessors (with a total of 176 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 and secure remote access.

1.2 Overview

We begin with a review 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 the 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.

Several sections of this paper have been adapted from our previous paper [7] since many of the issues and technologies remain the same.

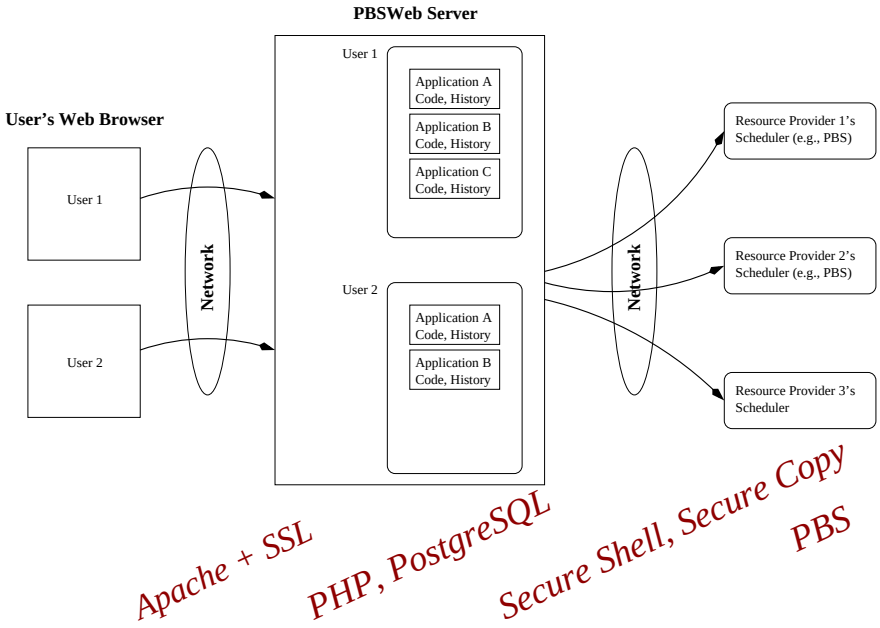


Figure 1: PBSWeb: Architecture

2 PBSWeb

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 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.

The users, the PBSWeb server, and the resource providers do not have to be at the same location (Figure 1). Users from across campus or across the country can access the resources. Furthermore, the same PBSWeb server can serve as a front-end to different RPs. Our current PBSWeb server can submit jobs to three different computers based on the user's choice. A future goal for PBSWeb is to have the system automatically select the most appropriate or least loaded system on which to run the job.

To begin, an authorized user connects to the Web 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.

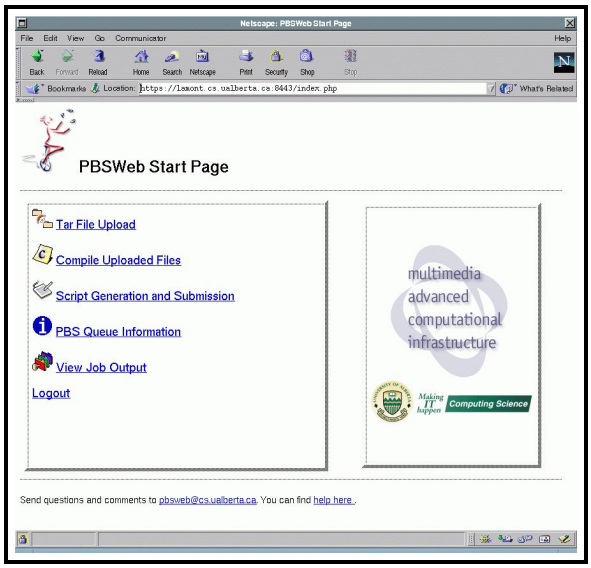


Figure 2: PBSWeb: Main Page

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).

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 (Figure 3). 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,

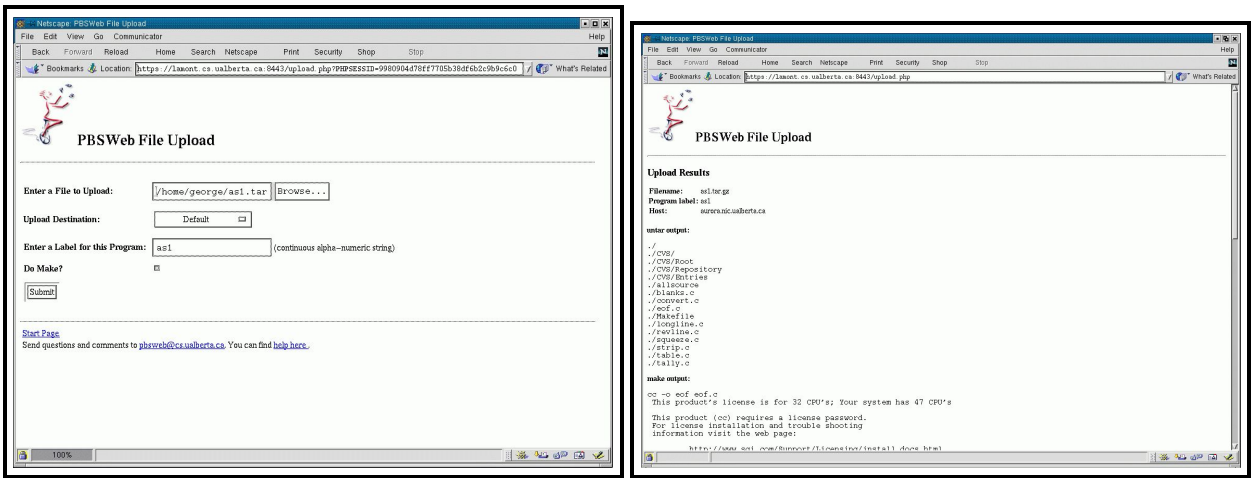


Figure 3: PBSWeb: Upload Tar File and Make

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. If the application has already been executed using PBS, then the previous job scripts can also be selected via the list of `Job names`. The remaining fields of the Web page dynamically change to reflect the parameters of the selected job (Figure 4).

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 `Default`, `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 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.

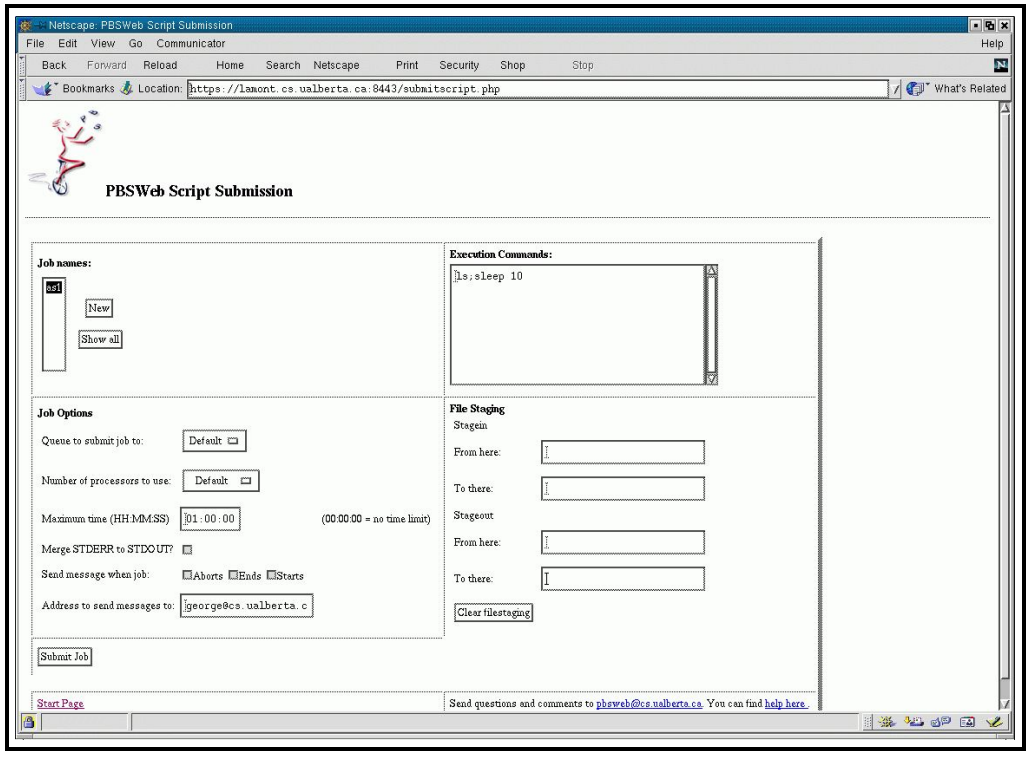


Figure 4: PBSWeb: PBS Job Submission and Script

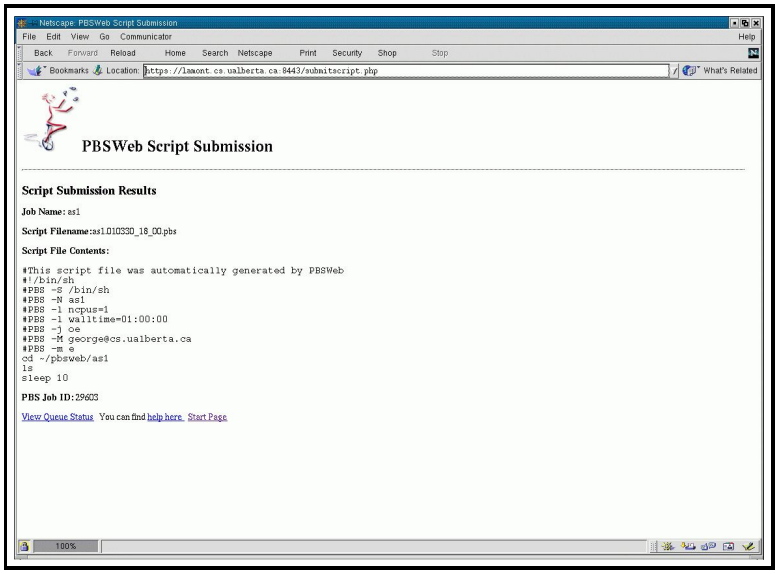
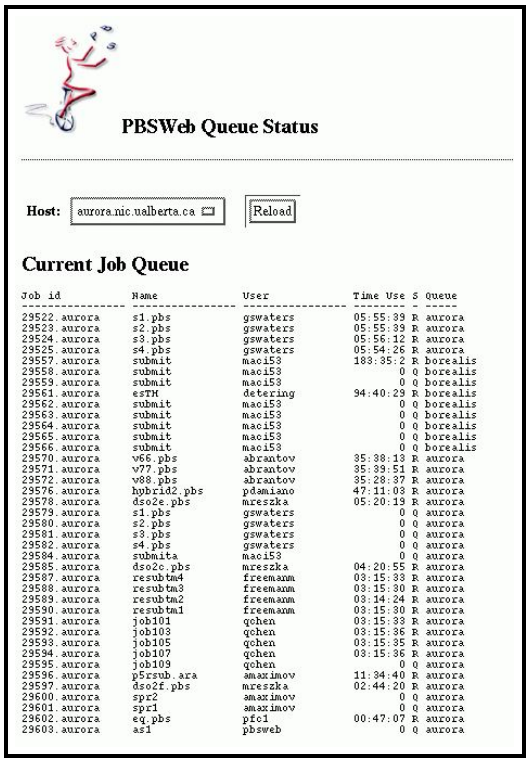


Figure 5: PBSWeb: Job Submitted



PBSWeb Queue Status

Host: aurora.nic.ualberta.ca Reload

Current Job Queue

Job id	Name	User	Time Use	S	Queue
29522.aurora	s1.pbs	gswaters	05:55:39	R	aurora
29523.aurora	s2.pbs	gswaters	05:55:39	R	aurora
29524.aurora	s3.pbs	gswaters	05:56:12	R	aurora
29525.aurora	s4.pbs	gswaters	05:54:26	R	aurora
29557.aurora	submit	maci53	183:35:2	R	borealis
29558.aurora	submit	maci53	0	Q	borealis
29559.aurora	submit	maci53	0	Q	borealis
29561.aurora	esTM	detering	94:40:29	R	borealis
29562.aurora	submit	maci53	0	Q	borealis
29563.aurora	submit	maci53	0	Q	borealis
29564.aurora	submit	maci53	0	Q	borealis
29565.aurora	submit	maci53	0	Q	borealis
29566.aurora	submit	maci53	0	Q	borealis
29570.aurora	v66.pbs	abrantov	35:38:13	R	aurora
29571.aurora	v77.pbs	abrantov	35:39:51	R	aurora
29572.aurora	v88.pbs	abrantov	35:28:37	R	aurora
29576.aurora	hybrid2.pbs	pdamiano	47:11:03	R	aurora
29578.aurora	dso2e.pbs	mreszka	05:20:13	R	aurora
29579.aurora	s1.pbs	gswaters	0	Q	aurora
29580.aurora	s2.pbs	gswaters	0	Q	aurora
29581.aurora	s3.pbs	gswaters	0	Q	aurora
29582.aurora	s4.pbs	gswaters	0	Q	aurora
29584.aurora	submita	maci53	0	Q	aurora
29585.aurora	dso2c.pbs	mreszka	04:20:55	R	aurora
29587.aurora	resubtm4	freemanm	03:15:33	R	aurora
29588.aurora	resubtm3	freemanm	03:15:30	R	aurora
29589.aurora	resubtm2	freemanm	03:14:24	R	aurora
29590.aurora	resubtm1	freemanm	03:15:30	R	aurora
29591.aurora	job101	qchen	03:15:33	R	aurora
29592.aurora	job103	qchen	03:15:36	R	aurora
29593.aurora	job105	qchen	03:15:35	R	aurora
29594.aurora	job107	qchen	03:15:36	R	aurora
29595.aurora	job109	qchen	0	Q	aurora
29596.aurora	p5rsub.ara	amaximov	11:34:40	R	aurora
29597.aurora	dso2f.pbs	mreszka	02:44:20	R	aurora
29600.aurora	spr2	amaximov	0	Q	aurora
29601.aurora	spr1	amaximov	0	Q	aurora
29602.aurora	eq.pbs	pfc1	00:47:07	R	aurora
29603.aurora	as1	pbsweb	0	Q	aurora

Figure 6: PBSWeb: Queue Status

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 Origins 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. The first implementation of PBSWeb [7] was a proof-of-concept prototype. We quickly recognized significant limitations in PBSWeb and re-implemented the entire system to address issues in security and job history management.

3.1 Components of PBSWeb

PBSWeb is composed of several components: a secure Web server (i.e., Apache with Secure Socket Layer (SSL)), a server-side scripting system (i.e., PHP), a database system (i.e., PostgreSQL), PHP scripts, and PBS servers. Also, secure shells (`ssh`) and secure copies (`scp`) are used to execute commands and transfer data on behalf of the user. These components have been integrated such that they work together as a system.

Web server: The role of the Web server is to ensure ubiquitous access to the system. The Web-server is PHP-enabled through a module in Apache. At the server, the Web pages are stored as PHP scripts. Before being passed on to the user's browser, the PHP is processed on the server into a HTML document with embedded JavaScript. Within PHP scripts, the job history is accessed and other dynamic content can be generated. PHP is a better choice than CGI (our previous technology) because it is more flexible and powerful, especially when combined with a DBMS.

Database: The database system records all the actions performed by the user. In particular, all user accesses, file uploads, job submissions and user preferences are saved in the database.

In addition to history information, the database holds information about the users, such as username and password. Also, the database contains PBS host (i.e., resource provider) configuration information. This allows maximum flexibility for administration purposes. Adding a new user to the system is equivalent to adding some new entries in the database tables. Adding new machines that provide PBS service is just as easy. Since the dynamic HTML pages are generated based on the database, the PHP scripts do not change as users and RPs are added to the system.

Database access is done in a persistent manner; connections are kept open across a session until explicitly closed. This feature decreases overall response time by not having to repeatedly open and close connections to the database.

Using a real DBMS is a significant improvement over the previous version which used the filesystem to store its

data. Issues such as synchronization, file structures, and validation are now automatically taken care of by the DBMS. In addition, it is now possible to keep the information in one place and access it consistently.

PHP Scripts: The choice of PHP as our server-side scripting language is based on its versatility for Web programming. PHP can do what was previously accomplished using Perl. In addition, PHP offers seamless access to Web-state information. Also, PHP code is embedded into the HTML pages (as a server-side include (SSI)) which helps organize and modularize PHP programs and related HTML documents.

The files are organized into categories. Some files are responsible for HTML page generation, others represent libraries of functions. For example, all database access is modularized in one file. Also, there is a category of files that hold configuration data for the whole system. System paths constitute an example of what kind of configuration information is stored in these files.

PBS Server: The Web server and the PBS server are separate; the Web server does not need to run on the same machine as the PBS server. When PBSWeb needs to invoke a PBS executable to, for example, submit a job (i.e., `qsub`) or check a queue (i.e., `qstat`), the PBSWeb server starts a `ssh` secure shell on the PBS host under the user's account name.

PBSWeb is layered on top of the existing PBS installation and does not change it in any way. Consequently, we claim that PBSWeb could be easily ported to other batch scheduler systems.

3.2 Handling User Data

As stated above, all PBSWeb configuration information resides in PHP configuration files and in the database. One particular configuration parameter is the directory name used for uploading files, which defaults to `pbsweb`. Although PBSWeb keeps a history of the user's job scripts, the actual executables, source code, job output, and input data are stored in the user's own account under the `pbsweb` directory. This directory will be present in the user's home directory in each host the user wants to access through PBSWeb.

Login and Authentication: At login time, the username/password combination is checked against the values stored in the PBSWeb database. This username and password is different from any username/password used to access the HPC systems themselves, which is a small but notable security advantage.[1] If they match, a unique session key is

[1] Specifically, PBSWeb does not have to know any username/password information for the computer accounts.

generated from a combination of the username and current time. This ensures that different simultaneous accesses of the same username, such as two scientists using the same account, are handled properly.

The value of the session key is saved in the database. The session key is the value that will bind together all subsequent file and job submissions; it is possible to trace every user action to the original session in which the action took place.

Besides the login script, other PHP scripts are derived from a common template that handles authentication. The authentication is based on the session key; its value is passed from page to page. If the value of the session key is found in the database and marked as unexpired, then it is a valid session key and the page is authenticated. Through the use of this mechanism we ensure that the scripts on the PBSWeb site can only be accessed if properly authenticated.

User Account Access: Access to the PBSWeb user's account and home directory at the RP is achieved using secure shell (`ssh`). Before a user creates a PBSWeb account, the user must add PBSWeb's `ssh` public identity key to his or her `authorized_keys` file in order to give PBSWeb home directory access. The public key is supplied on PBSWeb's account creation page. PBSWeb is thus given complete access to the user's account. Similarly, PBSWeb is also able to use secure copy (`scp`) to transfer data and source code to the user's account (i.e., into directory `pbsweb`).

Admittedly, giving PBSWeb complete access to one's account is not ideal. However, we feel that the `ssh`-based solution is better than simply providing PBSWeb with the user's actual password. First, not storing the user's HPC system's password within PBSWeb eliminates one possible security attack. Second, it is easier for the user to revoke PBSWeb's ability to access the account. The user can simply comment out the relevant identity in the `authorized_keys` file instead of changing the password or trying to remove it from PBSWeb's database. Third, PBSWeb inherits all of the encryption and authentication benefits of the secure shell without reinventing the wheel or requiring superuser access at different RPs.

In the future, we hope to develop a system that is analogous to `sudo` to give PBSWeb constrained access to the user's account.

Security Overview: As already discussed, there are three important components to PBSWeb's approach to security. First, there is password protection to access PBSWeb itself. The key is difficult to session forge and it is never sent across the network in unencrypted form. Second, SSL is used between the browser and the Apache server to protect sensitive information and data, such as passwords and the session key. Third, secure shell and secure copy protect data

transfers and information between the PBSWeb server and the HPC system.

4 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 run-time 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 workflow 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.

5 Concluding Remarks

In practice, effective high-performance computing and metacomputing requires a proper software infrastructure. 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 are developing an interface to PBS called PBSWeb. It is an on-going project and there remains a great deal of work to be done; we have been improving the system since the first prototype was implemented. To address issues with security and managing the user's job history, we completely re-implemented PBSWeb to (1) use secure and encrypted channels between the browser, PBSWeb, and the RP (i.e., SSL, secure copy, and secure shell), (2) use a relational DBMS to store the user's history, and (3) to manage session keys and provide dynamic content based on the user's session context.

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.

Availability

PBSWeb is currently undergoing testing with MACI users. We plan on making the entire system available in the near future. Please contact `paullu@cs.ualberta.ca` for further information at this time.

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] George Ma and Paul Lu. PBSWeb: A Web-based Interface to the Portable Batch System. In *Proc. 12th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS)*, pages 24–30, Las Vegas, Nevada, U.S.A., November 6–9 2000. Available at http://www.cs.ualberta.ca/~paullu/.

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

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

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