Design Issues in Building Web-Based Parallel Programming Environments

Kivanc Dincer and Geoffrey C. Fox
Northeast Parallel Architectures Center and
Department of Electrical Engineering and Computer Science
CST Mail-Stop 3-217
Syracuse University
Syracuse, NY 13244-4100
{dincer, gcf}@npac.syr.edu

Abstract

We exploited the recent advances in Internet connectivity and Web technologies for building Web-based parallel programming environments (WPPEs) that facilitate the development and execution of parallel programs on remote high-performance computers. A Web browser running on the user's machine provides a user-friendly interface to server-site user accounts and allows the use of parallel computing platforms and software in a convenient manner. The user may create, edit, and execute files through this Web browser interface. This new Web-based client-server architecture has the potential of being used as a future front-end to high-performance computer systems. We discuss the design and implementation of several prototype WPPEs that are currently in use at the Northeast Parallel Architectures Center and the Cornell Theory Center. These initial prototypes support high-level parallel programming with Fortran 90 and High Performance Fortran (HPF), as well as explicit low-level programming with Message Passing Interface (MPI). We detail the lessons learned during the development process and outline the tradeoffs of various design choices in the realization of the design. We especially concentrate on providing server-site user accounts, mechanisms to access those accounts through the Web, and the Web-related system security issues.

Keywords: *Parallel programming, programming environments, web security, HPF, MPI, Fortran 90, WWW.*

1 Introduction

The World-Wide Web (WWW) has emerged as an exciting and innovative front-end to the Internet. It provides users with a uniform and convenient means of accessing the wide variety of information resources (pictures, text, data, sound, video) on the Internet. On the other hand, Web browsers make the Internet a more user-friendly environment by integrating mechanisms to access all those resources within a single tool. Although the primary reason for connecting a remote Web server by using a Web browser is to gain access to information, it is not the only resource available. The WWW infrastructure also has the potential of being used as a convenient alternative way of accessing remote high-performance hardware and resources.

We expect that in the future the Web will be the standard user interface in many organizations for accessing computational resources. X-windows on Unix platforms, or Windows environments on personal computers will be replaced by Web browser-based user interfaces. Based on this intuition, we have designed and implemented *Web-based parallel programming environments* (*WPPE*s) that go beyond providing a set of features commonly found in Web servers. In such an integrated environment, the Web browser provides a user-friendly interface to server-site user accounts and allows the use of high-performance parallel computing platforms and software in a convenient manner. Users do not need to log into a Unix account or type Unix commands. Once they supply the required username and password, they are logged into their accounts and can use the computational facilities through the Web interface. Even anonymous users may be given a limited set of privileges to access the environment.

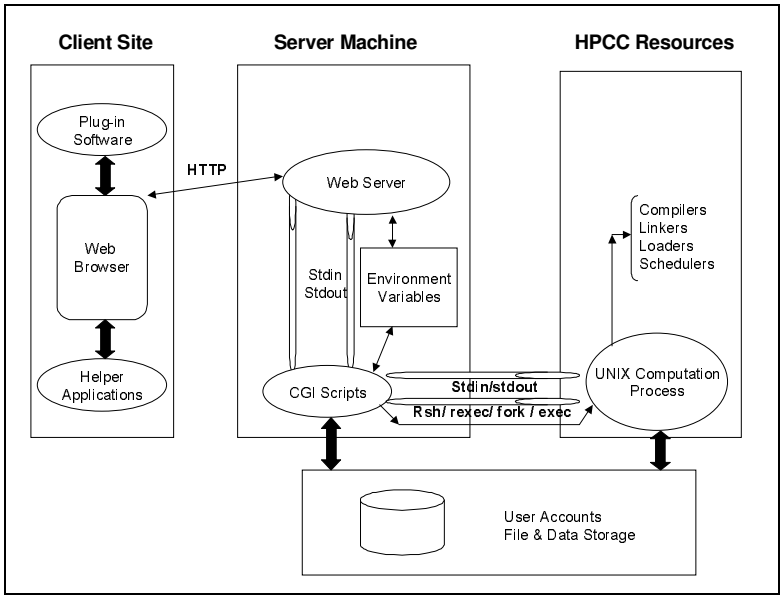

Figure 1. Illustration of the WPPE architecture.

The WPPEs presented here are prototypes that we developed using standard Web technologies. The interactive Web, high-performance computation back-ends controlled by CGI-extended Web servers, and HTTP-based communication represent other critical enabling technologies in this framework. HTML is used mostly for static components, while the use of Java and JavaScript ensures interactivity and visual animation at the client-site.

While we target this infrastructure to work inside large organizations with Intranets consisting of a large number of high-performance computers, it is also usable on a larger scale across the entire Internet.

Web-based delivery makes it possible to access posted resources from any platform. Using a central computation server makes it easier for its maintainer to make changes to the posted software, to indicate certain resources as sharable, and to change access restrictions and management policies in a quick and convenient manner without needing super-user privileges. In addition, quick implementations are possible since there is no need to modify the server and browser codes.

In this paper we describe our experiences in building proof-of-concept WPPEs. These prototype systems were primarily used for educational purposes and for software demonstrations at Syracuse and Cornell Universities. We discuss several technical challenges associated with building such WPPEs, depending on the underlying system and targeted use of the software. Among these can be counted providing virtual or real UNIX accounts to system users, ensuring the security of the system, and deciding on the system facilities to be provided to users. Section 2 summarizes the distinctive features of the prototype systems and describes their system architecture. Section 3 lays out several design alternatives and discusses the tradeoffs associated with each of them. Section 4 presents the system security measures that are crucial in order to avoid comprising the security of the whole system

2 The Architectural Properties of WPPEs

The implementation of WPPE's is based on standard Web technologies. Built-in server-site extensibility mechanisms such as CGI and client-site support with Java, JavaScript, and helper applications open the way to high-performance computation servers accessed through the Internet using Web browsers.

The WPPE architecture is built upon a three-tier client-server paradigm that separates the presentation of services and results to the user (i.e., user interface) from its internal command processing, data storage, and computing. The three layers of this architecture are the Web browser user interface, the Web server, and the back-end high-performance computing resources, as shown in Figure 1. The client demands access to remote parallel high-performance computing resources, and a central Web server coordinates the accesses to the local computational resources and helps to maintain the user accounts on the server-site. The server site carries the heavy load of all the computation and file storage.

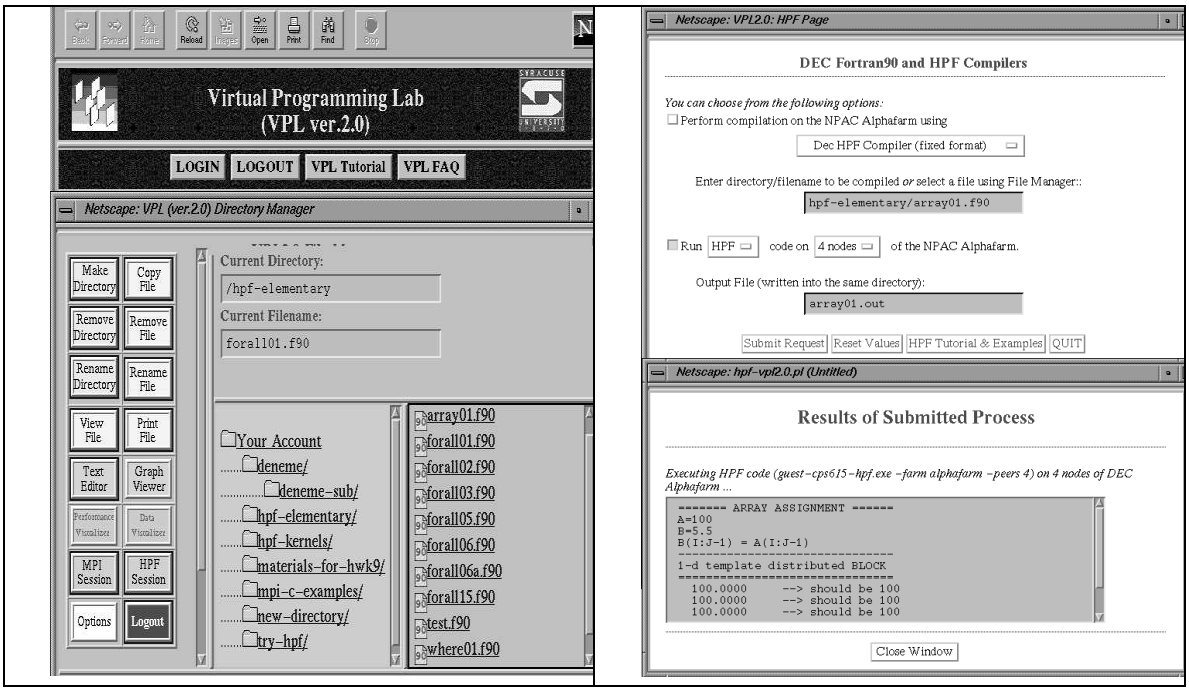

Figure 2. Snapshots of VPL the user interface. On the right, HPF compilation screen is shown.

2.1 The WPPE User Interface.

Client site requirements for using WPPEs are minimal, only a JavaScript- and Java-enabled Web browser needs to be available on the client machine. The Web browser software is responsible for displaying HTML documents and Java applets that are loaded from the server site. The Java language helps to provide an interactive user-interface and well-developed graphical utilities for visualizing the results and behavior of programs.

2.2 The WPPE Web Server

The CGI-extended Web server is the core of the system. An HTTP server accepts requests from the browser and starts the CGI scripts responsible for requested operations. The requests may be as simple as fetching a file from a disk or as complicated as compiling a code and showing the results graphically using some kind of visualization tool. Stored CGI scripts in the Web server layer provide the most common method of intervening between user interface and high-performance computing resources and the user accounts in the third layer.

CGI scripts activate or fork UNIX processes on the HPCC machines in order to perform requested services, and collect results or error messages from those forked processes. Some of these processes conduct file and directory operations, while others activate compilers, linkers, loaders, and other services on the back-end parallel machines. Since CGI scripts are written in Perl, A Perl interpreter should also be installed on the server machine.

All the related HTML files with JavaScript extensions, Java class libraries, and temporary configuration files are stored on the server site.

2.3 Back-End, High Performance Computing Resources

The back-end, high-performance computing platforms own huge computing resources (CPU time, memory, bandwidth, disk space). Upon a user's request, the server maps the computation (possibly involving parallel or distributed subtasks) onto the available computing resources.

In most cases, WPPEs use the compilers of sequential and parallel languages installed on these machines. In some systems, a front-end machine of the same type as the nodes of the parallel machine may be able to compile the programs separately. Web/HPF provides access to IBM's and PGI's HPF compilers on the IBM SP-2 platform, while VPL allows compilation of Fortran 77, C, Fortran 90, and HPF codes on a DEC Alphafarm workstation cluster. MPI and other runtime support or message-passing libraries can be used similarly.

2.4 Portability of WPPE Software

The WPPEs discussed here are highly portable from one UNIX system to another. Implementation of WPPE architecture is built upon platform-independent and portable components. *Platform-independent components* are readily supported in other systems without requiring any further processing. *Portable components* can easily be transported to other platforms, and this process requires at most a recompilation. Actual binaries may differ from system to system, but they all behave similarly, regardless of the platform they run on.

A WPPE configuration file is provided to ease the porting process. Locations of user directories and system utilities, URLs of Web servers are tuned into the new system by modifying the WPPE configuration file. All the building components of the WPPE software are platform-independent or portable:

- The user interface is implemented using HTML documents with JavaScript functions and Java classes that are all platform-independent. The Web server sends them to the client browser as ASCII or byte-code files, and the browser interprets them in order to shape the user interface to perform necessary actions.
- The CGI scripts that control the actions of the server in response to user requests from the client browser are written in Perl and kept as ASCII files. The Perl interpreters process them in the same way. The only required change may be the pathname for the Perl interpreter in the new system. Items forming the back-end computation engine of the server (such as the compilers of various languages, and their loaders and linkers) and runtime support libraries may have a different command format and arguments. These are not part of the WPPE software but of the underlying system. Depending on the UNIX shell[1] installed on the new system, the *system* and *exec* commands may need to be modified.
- The required Web server software may be acquired from different sources. It is also possible to find free Web servers for any platform, many of which are compatible with their commercial counterparts.
- Similarly, Web browsers are available for all UNIX machines as well as personal computers. In order to use WPPEs, the user's browser should support JavaScript 1.1 and Java. We recommend using Netscape browser, version 3.0 or later in order to use our prototype systems. Microsoft's Internet Explorer only supports a subset of JavaScript 1.1 and therefore cannot handle all the features required for the implementation.

[1] There are different shells or the UNIX command interpreters available. The most common ones are sh, csh, jsh, ksh, bash, and tcsh.

2.5 Use of WPPEs in a Research and Educational Institution

Our original motivation for developing WPPEs was to use them in a research and education setting for various purposes. We intended to build a generic, Web-based front-end that could be used for supervised and unsupervised demonstrations of developed software, as well as on-site and virtual training workshops with little or no modification. Table 1 summarizes some of the features and constraints of WPPEs targeted for different areas. The requirements and constraints affect our design choices. Below, we summarize the main features of each of these four areas of usage:

- **Supervised demonstrations of developed software.** Typical examples include demonstrations of software products that are outcomes of an institution's research projects. The system should allow a bit of programming for the sake of demonstrating the flexibility of the software.
- **Unsupervised demonstrations of developed software.** Examples of software tools developed at a research center are posted on the WWW for trial and evaluation by interested parties from all over the world. Anonymous users are bound to running a predefined set of demonstration programs.
- **On-site training workshops.** The purpose of these workshops is to teach advance concepts in programming or the use of parallel machines and tools. Users can access the computational facilities as well as educational material from the same interface, the Web browser. The lifetime of the user accounts is limited to the duration of the training program, which is typically just a few days. Users mostly view the example codes and practice the newly taught concepts by executing slightly modified versions of those codes. Setting up a full UNIX account for such a short duration is cumbersome and requires super-user privileges.
- **Virtual training workshops or semester-long parallel programming labs.** These usually take several months to complete, and some or all of the registered attendees may be on remote locations. They usually require more involved programming on different HPCC systems.

2.6 Completed Prototypes

We have completed the implementation of three different WPPEs. Each of the following systems represents a different design alternative, as will be explained in the upcoming sections. The first one is targeted towards supervised and unsupervised software demonstrations, while the second and third are mostly

used in on-site and virtual workshops and in semester-long parallel programming labs.

2.6.1 HPF and Parallel C++ on the Web

A Web-based demo prototype, called "PCRC Web Demo" [1], was used to display the status of the PCRC [2] project at the Supercomputing'95 conference. NPAC's F90D/HPF compiler and Cooperating Systems Corporation's parallel C++ compiler, which share the same common runtime system, were demonstrated through a Web interface. The user interface for this system was implemented using standard HTML and Web form interface. This single-user system was later used for other supervised demos.

2.6.2 The Virtual Programming Laboratory (VPL)

VPL [3] is an integrated parallel programming environment. It consists of a visual file manager for manipulating files and directories in a user's account, laboratory modules for compiling and executing message-passing MPI [4] programs (written in Fortran, C, and Java) and data-parallel programs (written in Fortran 90 and HPF [5]), performance analysis and visualization subsystem to depict executed programs' performance behavior as animated or static displays, and a graphic plotting component to materialize output data as two-dimensional plots (Figure 2).

VPL targets the teaching of parallel programming using high-level parallel languages and low-level message-passing libraries. VPL was first used in a graduate level computational science course at Syracuse University during the Fall 1996 semester. Students used VPL to do their Fortran 90, HPF, and MPI programming assignments and accessed the relevant resources on a DEC Alphafarm cluster via standard Web browsers.

As its name indicates, the programming environment provided by VPL is virtual, and users have virtual accounts that are constructed by partitioning the file space provided for a dedicated UNIX account into subdirectories for each user. VPL supplies users with a restricted set of UNIX shell utilities. The accounts are password-protected, and we used regular Web server (NCSA) and basic Web authentication to provide access to the accounts.

2.6.3 The Web/HPF

With C. Hecht, K. Barbieri, and A. Trefethen, the WPPE used for the PCRC Web demo was tailored for the Cornell Theory Center (CTC) environment, and named as the Web/HPF module [6]. The Web/HPF has been being used as a front-end to an IBM SP-2 parallel machine on the Andrew file system in CTC's Virtual Workshops since the beginning of 1997. It provides access to users' UNIX accounts and allows users to compile and execute HPF codes on the SP-2 by means of an HTML, Web forms, and JavaScript-based Web browser interface. Users also have access to the traditional UNIX text editors such as vi, pico, or emacs, and to other graphical tools on the server machine.

The Web/HPF module takes care of Web and Kerberos authentication, token access, and the attachment of users' requests to their account id. The Web user-name/password pairs are used as keys to access a password-protected *root*-accessible database keeping the real Unix usernames and passwords.

Usage	Amount of Programming	Users	Location of Users
Supervised Demos	A little bit	Known & trusted	On site or not
Unsupervised Demos	None	Not known	Anywhere
Onsite Training	Considerable	Known	On site
Virtual Training	Considerable	Known	On site or not

Table 1. Potential WPPEs and their associated constraints.

Usage	User has account?	Server Account	Process ID
Supervised Demos	Yes (virtual)	Web	Web
	Yes (real)	Root	User
	No	Web	Web
Unsupervised Demos	Yes (virtual)	Web	Web
	No	Web	Nobody
On-site and Virtual Workshops	Yes (virtual)	Web	Web
	Yes (real)	Root	User

Table 2. Appropriate server account and user id choices for various WPPEs.

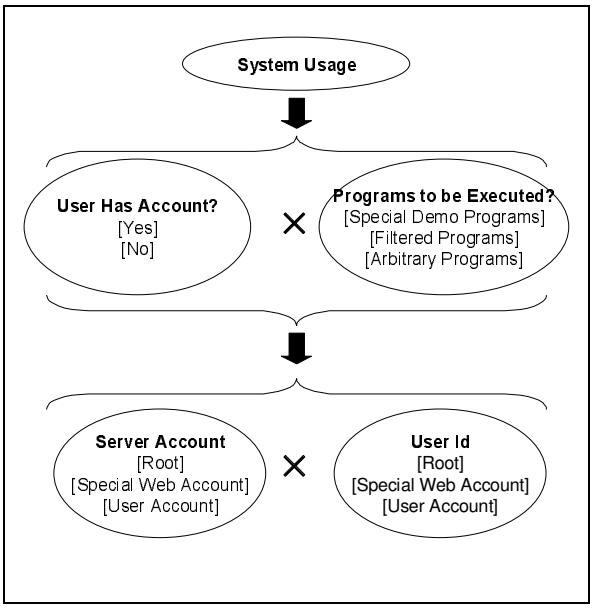

Figure 3. Relationship between intended usage and configuration of server account and user ids.

3 Implementation Issues

Depending on the intended usage and the underlying system properties, the requirements and implementation strategies for WPPEs vary significantly. In this section we will discuss some of the important issues and implications of several design choices. The following issues play important roles in the design:

- Providing user accounts and associated server configuration
- Ability to use server-site utilities
- Using client-site software
- Other issues such as logging user activity and accounting of system resource usage

3.1 Providing User Accounts and Associated Server Configuration

The intended use of the system determines the type of programs to be executed by system users and whether the users of the system will have accounts on the system. In turn, these two factors determine the *server account* and the *user id* options in the server configuration, as shown in Figure 3. *Server account* is the account where the original server daemon process is started and run. *User id* is the id that the server will use to answer requests. This may be the same as the server account user id, or if the server is run as root, it may be the user id of the children processes. CGI scripts and computational processes are activated using this user id. The users may or may not have an account on the system, and if they do, then the account reserved for them may be a real UNIX account or a virtual account.

3.1.1 Accessing Real UNIX Accounts.

A WPPE can be used as a front-end to users' UNIX accounts and it also provides a convenient GUI for accessing high-performance computational resources. Since the underlying operating system utilities readily provide most of the protection, file access, file space, and security mechanisms, the implementation of WPPE mainly concentrates on providing a convenient user interface, and on ensuring security in accessing user accounts through a Web interface.

As shown in Table 2, the server should be running under the *root* account, and will promptly switch to the user's UNIX account id by executing a *setuid* command as soon as the authorization is completed. A few lines of CGI code take care of this process. This is different than the way in which Web servers switch to other user ids that have been specified during their configuration.

The server configuration user id switch option is static and the server should be restarted to change the configuration settings. We may only benefit from that option if we choose to start a separate server having a different access port for each user of the system, which is not very scalable. Unless we start those servers via the UNIX *inetd*, all of them will stay sleeping in the background and consume system resources.[2] Even then, their startup is much slower, since the binaries should be loaded on the machine first and remain sleeping in the background once they have been started.

3.1.2 Providing Virtual Accounts.

Providing virtual accounts for WPPE users is a novel way of providing service to multiple users through a Web interface. The physical account's file space is partitioned among the WPPE users. Setting up the user accounts is easy and does not require superuser privileges. The CGI scripts and JavaScript functions handle coordination and partition of disk space among users. In some ways, providing such a virtual account environment is similar to providing a multi-user operating system on top of a single-user operating system; file protection and security mechanisms need to be implemented.

[2] Probably not the processing time, but file allocation table entries and other system resources are wasted.

We activated the server daemon from a dedicated account and kept it running with the same user id while serving further WPPE user requests to provide virtual accounts. Running the server daemon as a specially created, non-privileged user is important, because an intruder who finds a vulnerability in the server will have access privileges for only this unprivileged user. We used these dedicated accounts to run supervised demo programs.

3.1.3 Anonymous Users.

Anonymous users can access the WPPEs that we classify as unsupervised demo systems without requiring a user id or password in the system. Read-only and execute-only file access permissions given to those users should be adequate for running demo programs.

In order to prevent anonymous users from overwriting each other's files, each anonymous user is assigned a user id. This user id is a small tag derived from the connecting host id of the anonymous user and is added to the temporary files created as a result of the compilation and execution operations.

In this case, the server is run from a dedicated account and as soon as a connection is made the user id is switched to *nobody*, which has minimum privileges in the system.

3.1.4 Single-User Mode.

For supervised software demonstrations, a quick way of providing a WPPE is to design a single-user system. The requirements are minimal. The amount of disk space for setting up a Web server is usually small. For example, the NCSA server requires less than 2-MB space.

3.2 Ability to Use Server Site Utilities

A further consideration during the design stage of the WPPEs is whether the WPPE will allow the use of X-windows-based, standard system utilities located on the server site. As one of our design goals is to make WPPEs from personal computers as well as from UNIX workstations, we gave major importance to providing platform-independent versions of every kind of software that will be required in providing an interaction with the user. For example, we provided a platform-independent text editor, that was implemented by using Web forms, JavaScript, and CGI. We will soon supply a Java-based editor with compatible functions to traditional text editors.

Depending on the underlying system implementation and target platforms, it may sometimes be advantageous to exploit some server-site utilities. For example, when the client machine is a UNIX workstation, users may be given the option of using the X-windows-based system utilities installed on the server machine. The users of the Web/HPF module may choose to use a traditional X-windows-based editor such as emacs, vi, or pico, that directly open an external window on the user's display.

We may compare the advantages and disadvantages of custom-built components versus Unix system utilities as follows:

- Server-site utilities actually run on the server machine and have a tendency to overload the server machine artificially, since every key stroke is sent to and processed on the server machine. On the other hand, client-site utilities such as the ones implemented using form/CGI or Java runs on the client's machine. The interaction with the server is required only for certain operations.
- Using server-site utilities has the advantage of bringing the users a full-featured, well-tested software package that most of them are familiar with. However, Web-aware software products are not very widespread yet and therefore we had to build custom-made utilities with restricted capabilities.
- Many X-windows-based utilities provide a way to access the user-account directory structure, especially when loading or saving a file. We chose not to give the user the option of using those standard utilities in a WPPE with virtual accounts, since the user would be able to access other users` accounts using this mechanism.

A question that comes to mind is whether we should really make all of the client-site components platform-independent by rewriting them from scratch in Java. We believe that only the utilities that will be used by the client site and that need to be transmitted on the Internet are candidates for being made platform-independent. If a component will be used solely on the server (e.g., CGI modules), it does not need to be written in a platform-independent language like Perl. For the sake of faster execution, it can be written in a compiled language and its binary executable can be used in the server. Similarly, programs that need to be executed may times on the server can be written in standard portable languages and can be recompiled conveniently if the WPPE needs to be ported onto another platform.

3.3 Using Client-Site Software

In order to keep the transition from Unix to Web + Unix, or Windows to Windows + Web, the existing software should be reused while adopting the new Web technology. Until recently WPPE architectures were bound to using only server-site utilities for all processing.

JavaScript and Java[3], and file upload-support in recent browsers have brought in the possibility of being able to do some processing and computation on the client site or upload a file to the server site only when requested. Recent advances in browser technology have made it possible to extend the functionality of browsers by attaching helper applications and plug-ins to them. A Web browser can pass data sent from the server to a helper application that can interpret the incoming data. In this way, standard utilities that many people are accustomed to using on personal computers can be exploited with little modification. We expect a technology trend in this direction. For example, many text editors can be customized with little effort to access files across the network at URL addresses and to interact with Web browsers. This may also allow part or all of the user's file space to be kept on the client-machine. We expect that this approach will be more in line with the current trend toward extending Web browsers with helper applications.

3.4 Other Issues

3.4.1 Accounting

Parallel computer users are usually assigned a certain amount of computation quota and are charged for their process time usage. If a WPPE is being used to log into a user's real account, calculating the charges is trivial. Otherwise, the Web system must cooperate with the accounting system in order to reflect the usage of the resources to the user's actual account.

3.4.2 Logging of user activities

Automatic server logs do not write all this information to the log files; we may need to record various additional information with standard logs. We set the access permissions to the directory in which we keep the logs securely so that only the owner of the physical account can access to this directory.

Furthermore, there is a potential threat to running the server with the same id as the user. Users of the system should be prevented from being able to modify the log. This is similar to the filtering of arbitrary user programs that is explained in the next section.

The Web server is never run as *nobody*, since in that case any user would be able to modify logs that keep the accesses to the system.

3.4.3 Location of Servers

If the Web server runs in a confined environment and is not permitted to use *rsh* or *rexec* to execute parallel programs on the computational platform, then we have to run the computational Web server on a node of the parallel machine. This has the disadvantage of consuming the resources on the parallel machine. Furthermore, if users have their own servers running, the accounting process is affected. At NPAC we could run the servers from a dedicated workstation and send jobs to the parallel machines using remote execution mechanisms.

3.4.4 Automatic server startup

Parallel machines have a shorter mean time between failures. If the server is running on a node of the parallel machine, as in the Web/HPF, then we need to automate the process of starting the server during the reboot of the machine in order to keep the system functional. Servers may need to be started automatically by using *crontab* files after every crash of the machine on which they run.

A subject related to that is the detection of broken links to the high-performance machines. Since the Web server acts as a gateway to the backend computational resources, it should check the status of the systems at regular intervals and disable links to crashed or unavailable machines.

4 System Security

4.1 Server Setup

The first potential security threat may come from the HTTP server software itself. It is crucial to use one of the servers that is known to have no security bugs [7]. Once this is ensured the server has to be configured in a very careful manner in order not to hinder the security of the entire system. Many of the threats that occur may be the result of a careless configuration of the HTTP server and the poor organization of the server document directories [8].

When configuring the Web servers used mainly to serve static documents, it is recommended that extra protection be given to sensitive directories by turning off the *server-side includes* (*ssi*) option. However, in Web-based programming environments like the ones we have built, it was almost always necessary to use the *ssi* option to create dynamic documents on the fly.

The notable negative aspects of using ssi option include performance and security. Performance is an issue, since the server is not just sending some HTML to a client but parsing every HTML file for ssi directives before sending it to the client and building that HTML on the fly, or, actually executing commands on the server. We enabled the ssi option per-file basis in order to prevent the parsing of all files for ssi directives. The *ssi* can be abused by hostile users who prey on scripts that directly output things that have been sent to them.

[3] Current browsers supporting Java still do not allow applets to access the local file system or to use network communication freely on the client site

4.2 CGI Scripts

These CGI scripts compute information to be returned to users and are often driven by input from the remote user, who may be hostile. If these programs are not carefully constructed, remote users may be able to subvert them to execute arbitrary commands on the server system. Almost all vulnerabilities arise from these issues. It is a good idea to provide CGIs as statically linked binaries rather than as interpreted scripts. This will remove the need for a command inside the *chroot*ed environment. Moreover, the contents, permissions, and ownership of files in the CGI script directory should be carefully set.

4.3 Required Authorization Checks and Execution Rights

Running the server or clients as *root* or *superuser* is always dangerous; therefore, the script interpreted with the *root* privileges should not be longer than a few statements. If the authorization system breaks after authentication but before switching to the user's ID, the intruder may gain access to the *root* account.[4]

On the other hand, the WPPEs giving access to anonymous users do not need password authentication. Anonymous users have no write or modify access rights on the system. They are able to run the special demo programs, but are not allowed to upload their own programs As a precaution, the server gets the IP address of the client machine and put limits on the stack size and execution time of the anonymous user processes.

4.4 Authentication by Web Passwords

Access to WPPEs requires users to authenticate themselves by using a user id and password to the system. We used a Web authentication mechanism based on Web passwords and IP addresses to restrict accesses to the system. A user who does not supply a matching username/password pair is not allowed to access the system directories. The Web authentication mechanism allows certain directories or files under a server be specified as protected. A password file containing the usernames and associated passwords of all the valid users of the system is generated.

Another concern might be the security of transferring these passwords over the Internet. In spite of common myth, the passwords in recent browsers such as Netscape 3.0, are not transferred in "clear text" form, but in encrypted form just as in Telnet or FTP. Therefore, Netscape 3.0 is as secure as using other popular network-based, remote-access utilities.

[4] But even in this case, it should not be able to reach the shell interpreter if the X-window utilities are disabled.

4.5 Filtering User Input for Metacharacters

In WPPEs some input fields typed by users may be used as input to command interpreters such as Perl, AWK, Unix shells, or programs that allow commands to be embedded in outgoing messages such as /usr/ucb/mail. Any characters that have special meaning to the underlying shell in a query string may cause a CGI script to misinterpret those characters. A mischievous client may use special characters to confuse your script and gain unauthorized access. Languages like Perl and the Bourne shell provide an *eval* command that permits the execution of a string constructed by another statement on the fly. If the server script is going to use any data from the user to construct a command line for a call to *popen* or *system*, before calling the function, backslashes should be put before any characters that have special meaning to the shell. The input should also be checked against all kinds of special Unix meta-characters such as "*\n\r(.,/;~!)>|^& amp;$`<*" [9] and rejected if determined to be potentially dangerous, then using JavaScript functions at the client site, and CGI functions at the server site, the WPPE filters out all these metacharacters. This additional filtering at the server-site may seem to be redundant, yet we determined that it is critical in order to prevent mischievous attackers from interpreting manually-supplied command strings.

4.6 Protection Against Other Users

In a WPPE with virtual user accounts, another concern is possible attacks by other legitimate users of the system. To prevent this we set up a special account for the system and ran the server from this account. This brought us the flexibility of using Unix file protection mechanisms to protect our directories from other users of the same system. Furthermore, as much as possible, the diagnostics and error messages never indicates the actual location of the Web account and directory pathnames to the users.

We took several measures to prevent users of virtual Web accounts from being able to access other users' account. There is no way for the users of the system to go out of their directories by using any of the utilities provided by the system. VPL is the most capable system in this respect and approximates the Unix shell's functionality. In VPL, the commands such as delete/make/rename directory or delete/remove/rename/ copy file always manipulate items in the user's home account. Using JavaScript for validity checks at the client site, and more advanced CGI script-based checks on the server site, guarantees this result. Other WPPEs that we built needed only a subset of these precautions to ensure privacy of accounts.

4.7 Filtering Arbitrary User Programs

It was a challenge to provide access to more powerful, higher-level computing services without compromising security. When the server is running with users' UNIX account IDs, users are able to run any arbitrary programs, as they are executing in their own accounts. However, when virtual Web accounts are used, we should bring further restrictions to the programs to be executed by the system. Programs supplied by the user for execution may contain possible system calls. We filter the user programs for "system calls," and reject the execution if one is found. The user may also gain some access to other directories of the same account using input/output statements.

Confining untrusted programs is technically simple when the code is being interpreted. Since every single instruction is executed under the control of the interpreter, the interpreter can check the instruction arguments to ensure that all accesses are legal before executing the instruction. For example, Netscape [**10**] prohibits all local file accesses and allows network connections only to the server from which the applet was downloaded. We could have done something similar, but we chose to restrict the input/output syntax/semantics a little and allow user input/output. The input/output statements are filtered against any files the users open outside their own directories. We do not allow the use of variables as filenames in open statements, since this would elaborate the filtering process and might force us to adopt Java's way of thinking. The user must always put the filename in quotes in the open statement so that the system can validate it.

On one occasion following the Supercomputing'95 demonstration we evaluated the possibility of allowing anonymous users to run arbitrary programs in a WPPE without hindering the security of the system. One obvious security concern was the inevitability of giving Web clients the ability to compile and execute arbitrary programs on local hardware with whatever privileges the Web server might have. We could preclude some mischief by refusing programs that contained system calls, but users could easily trick the server into giving out, for example, password data using only standard I/O. Therefore we decided to take the safest choice of completely prohibiting the execution of arbitrary programs by anonymous users.

5 Conclusion

The implementation of Web-based programming environments poses challenging technical problems that need to be addressed. In this paper we have described the design and implementation of several WPPEs that we built to facilitate the development and execution of parallel message-passing and data parallel programs. The prototype systems allow browsing, program development, directory management, performance, and scientific data visualization within the framework of a single package. They have the unique property of using the Web as the standard interface for accessing computational resources from any platform (Unix, PC, or Mac) using a standard Java- and JavaScript-enabled browser.

Acknowledgements

We would like to thank Caroline Hecht, Kathy Barbieri, Susan Mehringer and Anne Trefethen from Cornell Theory Center for their collaboration in tailoring the HPF/Web module. Caroline should be acknowledged as the major implementor of the HPF/Web module. We corresponded mainly with Caroline and shared our experience, ideas, and some critical codes that made the implementation possible. We would also like to thank E. Weinman for proofreading this manuscript.

References

[1] Cowie, J., Dincer, K., and Li, X., "Towards a Web-Based PCRC Programming Environment," SCCS Technical Report, NPAC, Dec. 95.

[2] Parallel Compiler Runtime Consortium Project, at `http://www.npac.syr.edu/projects/ pcrc/.`

[3] Dincer, K. and Fox, G. C., "Using Java and JavaScript in the Virtual Programming Laboratory: A Web-Based Parallel Programming Environment," Concurrency: Practice and Experience, June 1997.

[4] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard," International Journal of Supercomputer Applications, vol. 8, no. 3 & 4, pp. 157--416, 1994.

[5] High Performance Fortran Forum, "High Performance Fortran Language Specification: Version 1.0," Scientific Programming, vol. 2, no. 1 & 2, 1993.

[6] Mehringer, S. And Barbieri, B., "A Workshop at Your Workstation," Cornell Theory Center Forefronts Magazine, pp. 10--11.

[7] WWW Security FAQ at http://www genome.wi.mit.edu /WWW/faqs/www-security-faq.html.

[8] ftp://ftp.auscert.org.au/pub/auscert/papers/unix_security_checklist, ftp://ftp.auscert.org.au/pub/auscert/papers/unix_ security _checklist 2.19 World Wide Web (WWW) – httpd.

[9] CERT Advisory CA-95:04 C.8.

[10] Netscape Communications, Inc. At URL http://home. netscape.com.