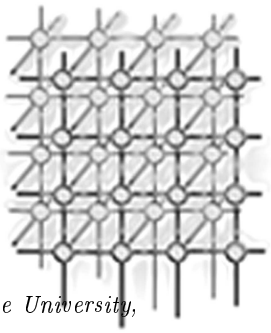
CONCURRENCY—PRACTICE AND EXPERIENCE

Concurrency: Pract. Exper. 2000; **00**:1–7 *Prepared using* cpeauth.cls *[Version: 2001/03/05 v2.01]*

The Gateway Computational Web Portal



Marlon. E. Pierce [*,†], Choonhan Youn [‡], Geoffrey C. Fox[§]

School of Computational Science and Information Technology, Florida State University, Tallahassee, FL 32306-4120; and

Indiana Pervasive Computing Research Initiative, Department of Computer Science, Indiana University, Bloomington, IN 47405-7104.

SUMMARY

In this paper we describe the basic services and architecture of Gateway, a commodity-based web portal that provides secure remote access to unclassified Department of Defense computational resources. The portal consists of a dynamically generated, browser-based user interface supplemented by client applications and a distributed middle tier, WebFlow. WebFlow provides a coarse-grained approach to accessing both stand-alone and grid-enabled back end computing resources. We describe in detail the implementation of basic portal features such as job submission, file transfer, and job monitoring and discuss how the portal addresses security requirements of the

*Correspondence to: 2435 Fifth Street, WPAFB, OH 45433-7802

†E-mail: pierceme@asc.hpc.mil

‡E-mail: cyoun@csit.fsu.edu

§E-mail: fox@csit.fsu.edu

Received

Revised

Copyright © 2000 John Wiley & Sons, Ltd.

deployment centers. Finally, we outline future plans, including integration of Gateway with Department of Defense testbed grids.

KEY WORDS: computational portals, computing environments, computational grids

1. INTRODUCTION

Computational grid technologies hold the promise of providing global scale distributed computing for scientific applications. The goal of projects such as Globus[1], Legion[2], Condor [3], and others is to provide some portion of the infrastructure needed to support ubiquitous, geographically distributed computing [4, 5]. These metacomputing tools provide such services as high-throughput computing, metaqueuing jobs across multiple machines, high-performance file transfer, information management, and security.

However, there are many services that these grid infrastructure technologies have not completely addressed. First and foremost, grid services can be daunting to use, especially to users not familiar with high performance computing (HPC) operating systems, queuing systems, and so forth. Second, from the application user's point of view, using HPC resources on a grid may be just one part of the task he or she has to accomplish. The use of grid resources needs to be seamlessly integrated with, for instance, commercial applications for pre- and post-processing data. Third, while technological challenges to integrating different grid technologies may be overcome, the sociological and institutional problems may be intractable, or at least solvable only over a much longer time scale. Thus it seems very likely, for example, that a single grid tying together the U. S. Department of Defense, Department of Energy, National

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

Science Foundation, and other institutions will be difficult to achieve, although within each of these institutions there might be one or more computational grids. Federating these grids with European and Asian grids introduces even greater hurdles. However, from a user's point of view there may not be the same problem, as a single user may acquire accounts from different institutions. Thus, our hypothetical user would benefit from a single environment that ties together his or her personal list of resources, even if there is no formal relationship between the institutions. Fourth and last, even within a particular institution there will be both HPC resources on a grid and resources kept isolated from the grid. Again, a user of these resources is likely to need both, so from his or her perspective there is a need to manage these resources from a single environment.

Grid computing environments address the problems of the preceding paragraph. One common type of such an environment is a web portal, which takes advantage of the technologies and standards developed for Internet computing such as HTTP[6], HTML[7], XML[8], CGI, CORBA[9, 10], and Java[11], and uses them to provide browser-based access to HPC systems (both on the grid and off). These portals may offer additional services as well, such as collaborative capabilities.

We have developed a computational web portal system, Gateway, for the Aeronautical Systems Center and Army Research Laboratory Major Shared Resource Centers. The goal of the Gateway project is to provide an intuitive browser interface that hides the difficulties of accessing and using HPC resources, and to do so in a secure manner.

The intent of Gateway is to provide web interfaces for a number of different technology areas, such as chemistry and material science, structural mechanics, and fluid dynamics. Interfaces

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

for these disparate areas can be extended from a set of generic services that provide the portal's core functionality. During the current phase of Gateway's development, we identified and implemented the following core services:

(i) Security: allow access only to authenticated users, give them access only to authorized areas, and keep all communications private.

(ii) User session state management: archive all of the user's interactions with the system, and allow the user to recover and edit old sessions.

(iii) Information resources: inform the user what codes and machines are available.

(iv) Queue script generation: based on the user's choice of code and host, create a script to run the job for the appropriate queuing system.

(v) Job submission: through a proxy process, submit the job with the selected resources for the user.

(vi) Job monitoring: inform the user of the status of his submitted jobs, and more generally provide events that allow loosely coupled applications to be staged.

(vii) File transfer: allow the user to transfer files between his desktop computer and a remote system, and to transfer files between remote systems.

These constitute the basic set of services. Other services that we are developing include collaborative job control, shared visualization, and task composition to create meta-jobs out of atomic tasks.

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

2. GATEWAY PORTAL ARCHITECTURE

The Gateway portal is implemented using the classic three-tiered architecture. These tiers separate functionality and responsibility, allowing us to develop pieces independently. Figure 1 illustrates a sample configuration; the actual configuration for a specific site can be varied as described below to agree with the security requirements of that site. The user interacts with the portal through either a web browser, a client application, or both. This is illustrated in Figures 2 and 3. The Tool Bar contains Java applications such as a file browser (shown) and a job monitor. Users can also submit pending jobs to the middle tier directly through the Tool Bar. Users also interact with the portal through any standard browser. The over-the-wire connection protocols between the client and the remote server include HTTP(S), IIOP[10], and SECIOP[12]. These latter two are the CORBA standard and secure protocols, respectively. The middle tier consists of two basic sections: a web server running a servlet engine (a Java Virtual Machine, or JVM) and a distributed CORBA-based middle tier (WebFlow). The web server typically runs a single JVM on a single server host that contains JavaBean components. These components may implement specific local services, or they may act as proxies for WebFlow distributed components running in different JVMs on a nest of host computers. WebFlow servers consist of a top level master server and any number of child servers. The master server acts as a gatekeeper and manages the life cycle of the children. These child servers can in turn provide access to remote back end services such as HPCs running PBS or LSF queuing systems, a Condor flock, a Globus grid, and data storage devices. Direct communication between the back end services is also possible.

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

2.1. User Interface

The central idea of the Gateway user interface is to allow users to organize their work into problem contexts, which are then subdivided into session contexts. Problems are identified by a descriptive name handle provided by the user, with sessions automatically created and time-stamped to give them unique names. Within a particular session, the user chooses applications to run and selects resources such as the number of nodes for parallel jobs, memory requirements, and wall-clock time. This interface organization is mapped to components in the WebFlow middleware described below. The data gathered from a particular session is represented internally as a linked set of hash tables. This structure is mapped into a directory structure on the server and stored persistently. We refer to the entries in the hash tables as context data.

Within the user interface we provide three tracks: code selection, problem archive, and administration, as shown in Figure 3. The code selection track allows the user to start a new problem, make an initial request for resources, and submit the job request to the selected host's queuing system. The problem archive allows the user to revisit and edit old problem sessions so that she can submit her job to a different machine, use a different input file, and so forth. Changes to a particular session are stored in a newly generated session name. The administration track allows privileged users to add applications and host computers to the portal, modify the properties of these entities, and verify their installation. This information is stored in an XML data record, described below.

The user interface is developed using JavaServer Pages (JSP), which allow us to dynamically generate web content and interface easily with our Java-based middleware. For example, Figure

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

5 shows a list of available applications and hosts. This is generated from an XML data record. A portal administrator can edit the data record to, for example, add a new host, so the Code Selection page will be automatically updated the next time any user visits it.

We remove flow control from the pages themselves, implementing this through an administrative Java servlet by using the "Command" design pattern [13, 14], illustrated in Figure 4. This allows us to easily insert new pages or otherwise alter the flow between pages, and to encapsulate additional functionality between certain pages. For instance, the "Submit" button for running a job invokes the control servlet, which looks up the appropriate command object from a hash table. This command results in a call to the middleware tools for job submission and a display of the next page in the user interface. We implemented this design in order to provide a user interface that can be easily improved and expanded by web developers with only a minimal knowledge of Java, high-performance computing, and grid technologies. At the current stage of development we are more interested in providing the framework tools for building user interfaces than in the user interfaces themselves.

2.2. Component-Based Middleware

The middle tier acts as the service broker that links the user's requests to the appropriate back-end resources. Because of the designed linkage between JavaBeans and JSPs, we partially implement the middle tier as JavaBean components. However, JavaBeans all run on the same host server, which does not allow us to deploy a distributed middle tier on multiple, perhaps geographically distributed, hosts. To do this we have a more powerful, CORBA-based implementation of the JavaBeans specification called WebFlow. In principal this would allow

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

us to handle load balancing by distributing the server workload among various machines, but the primary goal is to provide a network of proxies on machines that may have different file systems or that may even be operated by different organizations.

The WebFlow middleware has been described elsewhere [15, 16, 17, 18], and we will only summarize it here. Also, for a general overview of the role of commodity technologies in computational grids, please see Ref. [19]. WebFlow allows a hierarchical arrangement of CORBA servers to be created. A single server acts as a parent and gatekeeper to any number of child servers. Child servers themselves can contain child servers of their own, and so on. This is depicted in Figure 6. The parent server maintains proxy images of all its children and acts as a single entry point for contacting the children. Within a particular child, we can add containers that map to the user's base context, problem contexts, and session contexts of the user interface. The servers can also hold *modules*, CORBA implementation files that provide specific services.

Clients contact and interact directly with the parent (root) server only, which acts as a naming service to forward requests on to the appropriate child. We use a standard directory-style naming convention for child servers, so for example a grandchild server named "gchild" would be accessed through the root by the name "parent/child/gchild." A particular child component is added to a parent by a mechanism analogous to the container organization used to create graphical user interfaces. Parents can contain any number of children, but children have a single parent, resulting in a tree structure. A server within the hierarchy tree can contain both child servers and modules, and a particular module can be added to any number of servers. Thus, for example, two child servers at two different sites can each contain a module

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

for accessing the remote file system. A client contacting the parent server could then look up each of these children and thus have transparent access to the two file systems through a single entry point.

3. DATA DESCRIPTORS

In our portal design, we identified several groups of portal data that need to be described using XML [8]. We refer to these objects as *descriptors* and have so far defined descriptions of host machines, applications, and portal services.

XML descriptors are used to describe data records that should remain long lived, or static. As an example, an application descriptor contains the information needed to run a particular code: the number of input and output files that must be specified on the command line, the method the application takes input and output, the machines that the code is installed on, etc. Machine, or host, descriptors describe specific computing resources, including the queuing systems used, the locations of application executables, and the location of the host's workspace. Taken together, these descriptors provide a general framework for building requests for specific resources that can be used to generate batch queue scripts.

To implement application and host descriptors, we adopted the eXtensible Scientific Interchange Language (XSIL) [20]. XSIL was developed as a description language for scientific data. XSIL defines a relatively small set of generic data markup tags. The power of XSIL comes from the ability to associate markup tags with Java code that extracts information from the tags. XSIL documents for a specific application are associated with Java code that extends the functionality of the base XSIL Java classes. Thus, XSIL can be appropriated for

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

general data descriptions. By adopting XSIL, we sacrifice some expressiveness and specificity when describing our portal data, but gain the advantage of a rapidly deployable system that can be easily adapted to our needs. XSIL's generality furthermore makes it easy to develop descriptors that can be extended to include additional information without introducing new tag definitions. Finally, we gain the advantage of a single markup language for describing both portal and scientific data.

In addition to application and host descriptors, we have implemented a third descriptor type to describe services. Service descriptors are used to describe the interfaces to particular WebFlow modules, rather than data records. Here XML is used since it represents a language-independent means of describing an interface that may be implemented in any particular programming language. In our current implementation, we use Java bindings for dynamically loaded CORBA objects. These objects are stored in a CORBA Interface Repository, so the appropriate translation of the service descriptor is to CORBA's Interface Definition Language (IDL).

4. SERVICE IMPLEMENTATIONS

In the previous section we described the overall architecture of the system and the information descriptors that we use. We now describe how these components are used to implement the basic services. Security is described in a separate section.

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

4.1. File Transfer

One of the important services we identified in planning the current phase of development was a file transfer mechanism that would allow entire directories to be transfered between the desktop and the remote servers using a simple user interface. This is illustrated in Fig. 2. This is intended to supplement file uploading and downloading through the browser. Although multiple file uploads are allowed in the HTTP specification, most browsers do not support this feature. We have implemented this as a client-side application that communicates directly from the user's desktop to a WebFlow server running a specially written module that can read to and write from the remote file system. Java input/output classes are used to access files on both the local and remote file systems. Files are then translated into binary arrays and passed over the wire using either IIOP or SECIOP. We can use this to send entire directory structures between machines, but in practice we limit transfers to include only the files in the chosen directory, not any subdirectories.

4.2. Batch Script Generation

The Gateway portal is designed in part to aid users who are not familiar with HPC systems and so need tools that will assist them in creating job scripts to work with a particular queuing system. From our viewpoint as developers, it is also important to design an infrastructure that will allow us to support many different queuing systems at different sites. It was our experience that most queuing systems were quite similar (to first approximation) and could be broken down into two parts: a queuing system-specific set of header lines, followed by a block of script instructions that were queue independent. We decided that a script generating service would

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

best be implemented with a simple "Factory" design pattern [13], with queue script generators for specific queues extending a common parent. This is illustrated in Figure 7. This allows us to choose the appropriate generator at runtime. New generators can be added by extending a common parent. Currently we support PBS, LSF, and GRD queuing systems.

Queue scripts are generated on the server, based on the user's choice of machine, application, memory requirements, etc. This script is then moved (if necessary) to the selected remote host using a WebFlow module. The information needed to generate the queue script is stored as context data, described below. This allows the user to return to old sessions and revise portions of the resource requests. If this modified request is to be run on a different queuing system, then a new script can be generated using the slightly modified context (session) data.

4.3. Job Submission

WebFlow provides a module for executing submission requests either as local commands or as remote procedures through the rsh and ssh commands. These command are called as external processes to the JVM running the module. Because WebFlow servers can run on distributed hosts, the "local" command execution can be performed by a child running directly on the desired HPC system.

It is possible to provide a closer coupling between WebFlow and the application if the code or at least the interface is available. Subroutines and functions can be wrapped in IDL and called through a custom-developed module. In practice, we have not implemented this service for any codes because we primarily support commercial applications.

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

4.4. Job Monitoring

We provide a rudimentary method for monitoring the status of the user's job. We do this through a simple event generating mechanism on the remote HPC, rather than through server-side polling. Possible event states are "pending", "queued", "running", and "completed". The pending state applies to a job that has been described but not yet submitted to a queue. Support for this state is useful at centers that impose limits on the number of jobs that can be queued at one time. The queued job has been submitted to a particular host and is waiting to be executed. The notifications that a job has started to run and has finished (possibly with an error) are generated by a simple call-back program embedded in the the queue script that contacts the server and passes the event status. Because we typically support "black box" commercial codes, we have not implemented support for any internal checkpoint events while the code is running. All the events associated with a specific user are stored as a serialized hash table in the user's descriptor directory.

There are several advantages to using job-generated events in place of server polling. It scales well, since a single server does not have to periodically ping many running applications. Instead, the server is notified only when something interesting happens, and this notification is relatively simultaneous, with no built in lag time inherit to polling. Also, in our implementation the call-back is decoupled from the servers, so the servers can crash or be shut down and restarted without affecting the event monitoring. The disadvantage of the scheme is that it assumes that the HPC system itself and the network connecting it to the servers will remain up. In order to make monitoring more robust, we need to couple job event monitoring with host and network monitoring.

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

4.5. Session State Management

We have implemented a service for archiving interactions with the portal, allowing users to recover old state information and to recover from a server crash within a user session. This service, called ContextManager, has been implemented as both a WebFlow module and a web server JavaBean. There are trade-offs between the two versions: the WebFlow module can be distributed over multiple machines, but the local-only version has superior performance. The ContextManager service manages user, problem, and session context hash tables, which are mapped to a tree of directories on the server(s). The base of each tree is the user context, with subdirectories for problems and sub-subdirectories for sessions. We refer to this data as context data. Context data minimally describes the parent and children of a particular node, but can be extended to include any arbitrary name-value pairs. This is useful for storing, for instance, HTTP GET or POST requests. In practice we store all information gained from the user's interaction with the browser forms. For instance, the user's request to run a particular code on a certain host with a specified amount of memory, nodes, and wall time is represented as a series of name-value pairs that is stored in a directory that is associated with the problem and session contexts in which the request was created.

5. SECURITY REQUIREMENTS

Security is of utmost importance when grid resources are made available through the Internet. Security solutions of commercial web sites are typically inadequate for computational grids: customers and retailers are protected by third parties such as credit card companies and banks,

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

the company sponsoring the site profits from the sale of goods rather than services, and the site has no need to allow users direct access to its computational resources. None of these conditions apply to centers running computational portals.

In this section we discuss specific implementation issues for the Gateway portal, as well as some general security issues faced by portals.

5.1. Security in Multilayered Architectures

Three-tiered web portals require at least two tiers of security: client-server and server-back end connections must be authenticated, authorized, and made private. In addition, Gateway's distributed middle tier introduces the need for security between parent and child WebFlow servers.

Portals must be compliant with the existing security requirements of the centers where they are run. Gateway is being deployed in Department of Defense (DoD) computing centers that require Kerberos[21] for authentication, data transmission integrity, and privacy [22]. Centers also are required to use one-time passwords, created in effect by combining a static Kerberos password and a six-digit random passcode generated by a token card. Once a user is authenticated, he or she has access to all host machines at a center and can access other centers using cross-realm authentication.

We built the Gateway portal on top of the centers' preexisting security infrastructure. The key piece to making this possible is the CORBA security service. This is a general service that supports both Generic Security Service mechanisms (such as Kerberos) and SSL[23]. The goal of CORBA security is to provide a common application interface specification to a

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

number of different security mechanisms. Different CORBA vendors then choose the security mechanisms that they wish to support. Our first task then is to find a vendor supporting Kerberos and modify the WebFlow server to implement the additional security service classes. We use the secure ORB developed by Adiron Software, LLC [24]. This secure ORB must then be configured to support the particular requirements and policies of the deployment site.

The client-server connection is the first layer of the portal that must be secured. One implementation issue that we must face is that static passwords are not allowed and there is no native support in the security service for accepting the random token number. This precludes simple deployment based on Kerberos passwords. However, we decided that this was a solved problem: Kerberos clients are available for most platforms, and the secure ORBs can be configured to authenticate using a preexisting Kerberos ticket-granting ticket (TGT). Client applications can thus prove authentication to the kerberized WebFlow master server, assuming the client has obtained a TGT through a previous, normal Kerberos login.

The distributed WebFlow servers represent the second layer that must be secured. One challenge here is that the WebFlow servers typically do not run as privileged processes, whereas Kerberos makes the assumption that kerberized services are run as system level processes. Practically, this requires the service to have access to a system's keytab file, which has restricted access. In implementing kerberized WebFlow servers, we have obtained special keytab files from the Kerberos administrator that are owned by the application account and which are tied to a specific host machine. The use of keytab files can be avoided by user-to-user authentication. This would allow a client and server to both authenticate with TGTs, instead of a TGT

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

and keytab. Although part of the specification, user-to-user authentication is not typically supported by Kerberos implementations.

WebFlow presents an additional problem to the kerberized ORB. The parent and child servers act as both client and server at different times, so the WebFlow server must possess access to a valid Kerberos TGT as well as the keytab. The internal representation of the Kerberos credential object in the WebFlow server is thus a composition of the two credentials.

WebFlow servers may be run as a regular user account. This allows us to deploy a single parent server that runs as a gatekeeper and multiple child servers, one for each user. This scheme presents a simple solution to authorization: the user's personal server has all the normal permissions and restrictions as any other process he or she runs. Users may then make requests to their associated child servers, but the requests are actually intercepted and invoked by the master server. That is, the parent server acts as the user's proxy client and actually invokes the specific request. This requires two security features: mutual authentication and delegation. Mutual authentication is needed because the WebFlow child server, when it contacts the master to be dynamically added to the child list, must prove its identity to the parent. Thus it must authenticate as any other client. Likewise, the parent, in its role as gatekeeper, will act as client to the child when making invocations. Fortunately, the secure ORB can be easily configured to use mutual authentication as a policy. Delegation is required because the initial client request is processed by the master, and so the client must delegate authority to the master to invoke a command on its behalf. It is possible to implement an additional safety feature in the gatekeeper that verifies the client has requested access to a

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

child that it is authorized to use. This can be done by inspecting the credential received from the client and comparing it to the name associated with the requested child server.

Finally, secure connections to the remote back end machines (such as HPCs or mass storage) must also be made. To accomplish this, we create the user's personal server in the middle tier with a forwardable Kerberos ticket. The server can then contact the remote back end by simply invoking an external mechanism such as a kerbererized remote shell invocation. This allows commands to be run on the remote host through the pre-existing, approved remote invocation method.

5.2. Kerberos Security for Web Applications

One of the difficulties of using Kerberos as an authentication service for web applications is that it is not compatible with existing browsers. Early implementations of Gateway attempted to solve this problem by using a Java applet to establish the secure connection and manage the client requests. This introduced new problems, however, as this required installing native libraries on the client. We were thus required to override the native browser's Java Runtime Environment with a plug-in, but we discovered in turn that this broke the applet-browser communication capabilities.

As a solution to this problem, we developed the Charon security module. Charon consists of a client-side proxy application and a WebFlow module that intercept HTTP requests and responses between the user's browser and the Apache server. This is illustrated in Figure 8. A user wishing to connect to the Gateway web site must first get a TGT and then launch the client Charon application. He can then point his browser to a specified port on his local

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

host. All traffic through this port is forwarded to the Charon server (a module running in the WebFlow master server), which forwards the request to the local web server. The response is collected by the Charon server and sent back to the Charon client, which sends it to the local port. All web traffic uses DES encryption and MD5 hashing to preserve message integrity. These are standard features of SECIOP.

A drawback to the previous scheme is that it requires some installation on the user's desktop, and the dynamically linked native libraries require privileged access before they can be installed on some machines. As an alternative solution, we provide a browser interface that will create a Kerberos TGT on the server for the user. The user is authenticated in the usual way, with password and passcode. After the ticket is created on the server, we use a message digesting algorithm to create a unique cookie for the browser. This cookie and the client browser's IP address are used for future identification. Before the browser can view secured pages, these two identification pieces are verified against the session values stored on the server. All wire communications go over 128-bit encrypted SSL connections. Client certificates can be used as a third means of identification. The user can delete his server-side session tickets through the browser, and they will also be automatically deleted when the JSP session context expires.

6. INTERFACING WITH GRID SERVICES

The current version of the Gateway system was developed independently of typical grid services such as Globus, Legion, and Condor, although previous versions incorporated an interface to Globus job submission tools as an external process. We see an important benefit to providing

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

a bridge between systems that use standard grid technologies and systems that do not; i.e. stand-alone HPCs, clusters, private mass storage systems, and databases.

In the next phase of development, we will incorporate grid services available through the DoD's testbed implementations of Globus and Condor. We plan to initially develop interfaces to Condor's scheduler, and to the information and resource management services provided by the Globus toolkit. Globus's Resource Specification Language (RSL) provides a bridge to different queuing systems that could supplement our batch script generating tools. Likewise, we would benefit from grid information services such as the Globus MDS. Currently we maintain application and host information in an XML data record on the web server. The Grid Commodity Toolkit [25] should make this integration straightforward.

However, it is still important to support resources that do not use Globus tools. For this we believe that it is vitally important that the Global Grid Forum develop standards for resource description (based surely on RSL) and specifications for transforming these into batch scripts for the appropriate queuing systems. Likewise, we see the need for universal descriptions of information services to be developed, so that the machines running Globus information services can be integrated with other types of information services running on HPC resources.

7. SUMMARY AND FUTURE WORK

The Gateway portal provides a coarse-grained approach to accessing high performance computing resources. We have the capability to provide bridges to different grid infrastructure services, and where required implement these services ourselves.

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

The primary weakness in our current implementation is the lack of use for standard grid technologies. These provide much more sophisticated and robust services for job submission, scheduling, and information management than we have the resources to develop. We also currently provide no direct way of monitoring the availability of remote resources. An inherit assumption in our data structure is that remote host machines are always available. Solutions to these problems are available through other projects, and we plan to take advantage of these in the near future when the DoD grid testbed becomes available.

Apart from traditional grid services, we plan to integrate support for pre- and post-processing tools into our portal. An important example of this is remote visualization, which would give researchers tools for doing preliminary analysis remotely before beginning the time-consuming task of transferring large data sets to local machines for more intensive analysis. More broadly, we will investigate problems in grid workflow, which will provide the logic and mechanisms for linking multiple applications together into a single meta-job. These jobs can be distributed across many machines and will require data transfer and the ability to handle success/failure conditions, convergence conditions for applications, and scheduling for remote resources.

It is also crucial that Gateway stays abreast of commercial developments of standards in web services. Relevant standards include SOAP[26], WSDL[27], and UDDI[28]. If these protocol and message-centric commercial models prove viable, they will become important alternatives to interface-centric technologies such as CORBA.

ACKNOWLEDGEMENTS

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

This project was funded by the Department of Defense High Performance Computing Modernization Program through the Programming Environments and Training (PET) Programs at the Aeronautical Systems Center and Army Research Laboratory Major Shared Resource Centers. Ken Flurchick and Jan Labanowski of the Ohio Supercomputer Center and Tomasz Haupt of Mississippi State University were lead developers in earlier versions of this project. We wish to thank Paul Sotirelis and Kelly Kirk of the National Center for Supercomputing Applications for their assistance in developing remote Kerberos login.

REFERENCES

1. The Globus Project.
   http://www.globus.org/ [20 July 2001].
2. Legion: A Worldwide Virtual Computer.
   http://www.cs.virginia.edu/ legion [20 July 2001].
3. Condor: High Throughput Computing.
   http://www.cs.wisc.edu/condor [20 July 2001].
4. Foster I, Kesselman C (Eds.). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.
5. Global Grid Forum.
   http://www.gridforum.org [20 July 2001].
6. HTTP: Hypertext Transfer Protocol
   http://www.w3c.org/Protocols [20 July 2001].
7. Hypertext Markup Language Home Page
   http://www.w3c.org/MarkUp
8. Extensible Markup Language
   http://www.w3.org/XML/ [20 July 2001].
9. Orfali R, Harkey D. *Client/Server Programming with Java and CORBA*. John Wiley and Sons, 1998.

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

10. CORBA/IIOP Specification
http://www.omg.org/technology/documents/formal/corba_iiop.htm [20 July 2001].
11. Java 2 SDK, Standard Edition Documentation, Version 1.3.1
http://www.java.sun.com/j2se/1.3/docs/index.html
12. CORBA Security Service
http://www.omg.org/technology/documents/formal/security_service.htm [20 July 2001].
13. Gamma E, Helm R, Johnson R, Vlissides J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley 1995.
14. Fields D, Kolb M. *Web Development with JavaServer Pages*. Manning 2000.
15. Haupt T, Akarsu E, Fox G, Youn C. The Gateway System: Uniform Web Based Access to Remote Resources. *Concurrency: Practice and Experience*, 2000; **12**(8);629-642.
16. Bhatia D, Burzevski V, Camuseva M, Fox G, Furmanski W, Premchandran G. WebFlow–A Visual Programming Paradigm for Web/Java Based Coarse Grain Distributed Computing." *Concurrency: Practice and Experience*, 1997; **9**(6);555-577.
17. Akarsu E. *Integrated Three-Tier Architecture for High-Performance Commodity Metacomputing*. Ph. D. Dissertation, Syracuse University, 1999.
18. For additional references and presentation materials on Gateway, please see http://www.gatewayportal.org/Presentations.html.
19. Fox G, Furmanski W. High performance commodity computing. In *The Grid: Blueprint for a New Computing Infrastructure*. Foster I, Kesselman C (eds.). Morgan Kauffmann, 1999.
20. XSIL: extensible scientific interchange language.
http://www.cacr.caltech.edu/SDA/xsil/ [20 July 2001].
21. Neuman C, Tso T. Kerberos: an Authentication Service for Computer Networks. *IEEE Communications*, 1994 **32**(9):33-38.
22. Department of Defense High Performance Computing Modernization Program Security Issues.
http://www.hpcmo.hpc.mil/Htdocs/Security.
23. SSL 3.0 Specification
http://home.netscape.com/eng/ssl3.

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

24. Adiron: Secure System Design

http://www.adiron.com.

25. von Laszewski G, Foster I, Gawor J, Lane P. A Java Commodity Grid Kit. *Concurrency and Computation: Practice and Experience* 2001; **13**,645-662.

26. SOAP Version 1.2

http://www.w3c.org/TR/soap12 [20 July 2001]

27. Web Services Description Language (WSDL) 1.1

http://www.w3c.org/TR/wsdl [20 July 2001]

28. Universal Description, Discovery, and Integration of Business for the Web.

http://www.uddi.org [20 July 2001]

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

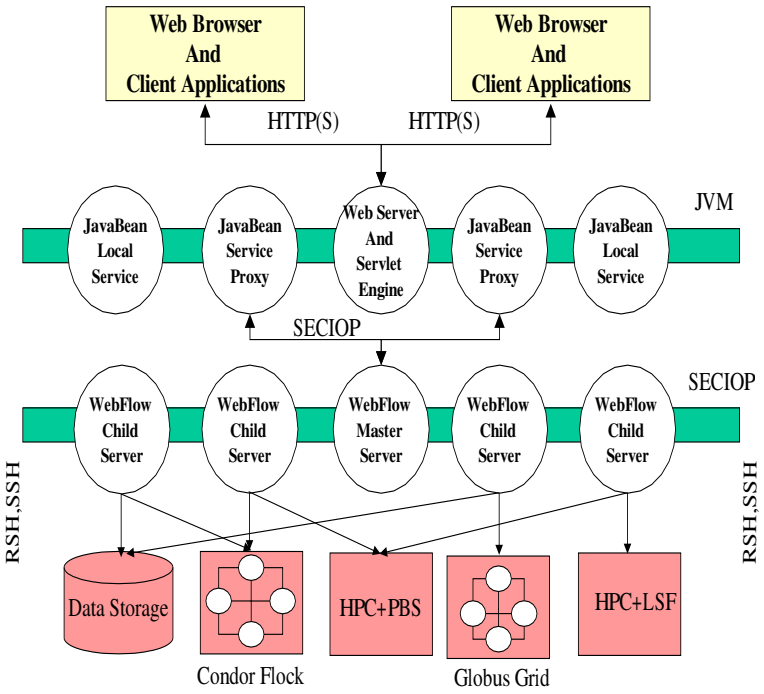


Figure 1. The Gateway computational portal is implemented in a multi-tiered architecture.

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

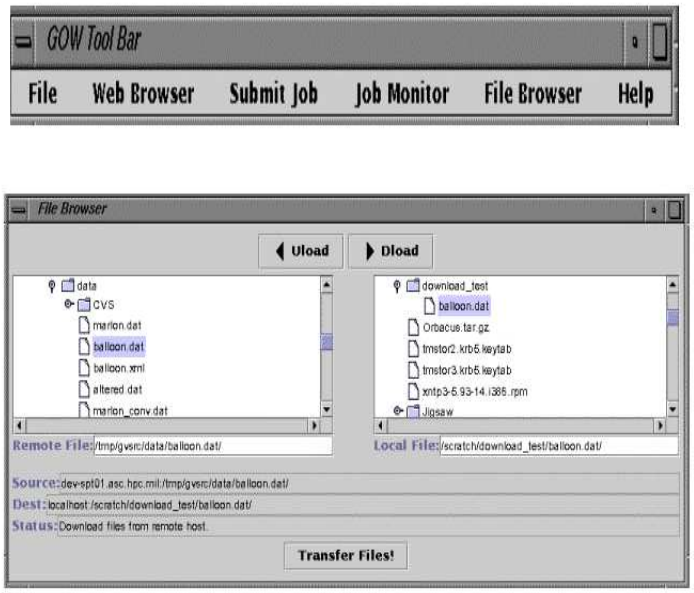


Figure 2. Gateway client applications are contained in the client tool bar. Tools include a file browser (shown below) and job monitorer. Users also submit jobs directly to the middle tier through the tool bar.

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

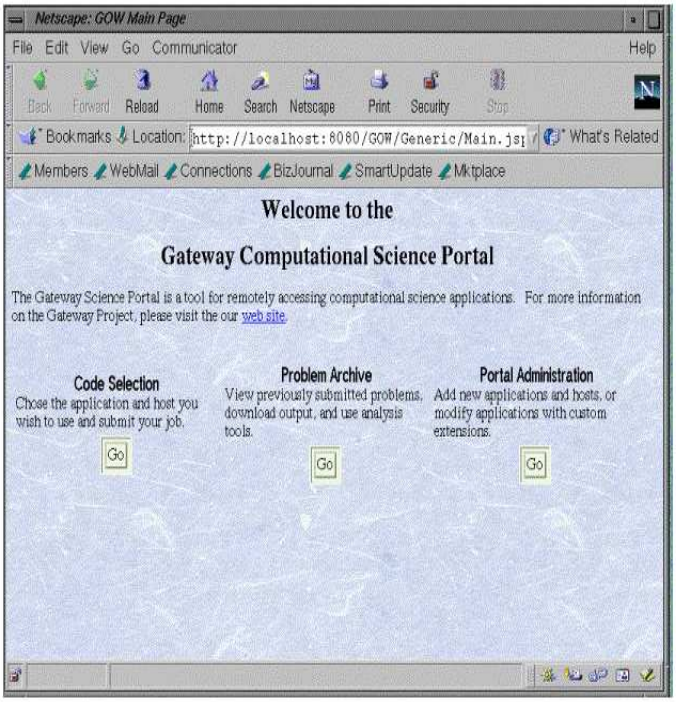


Figure 3. Gateway's "Welcome" page offers users three basic tracks: code selection for beginning new problems, problem archive access, and portal administration.

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

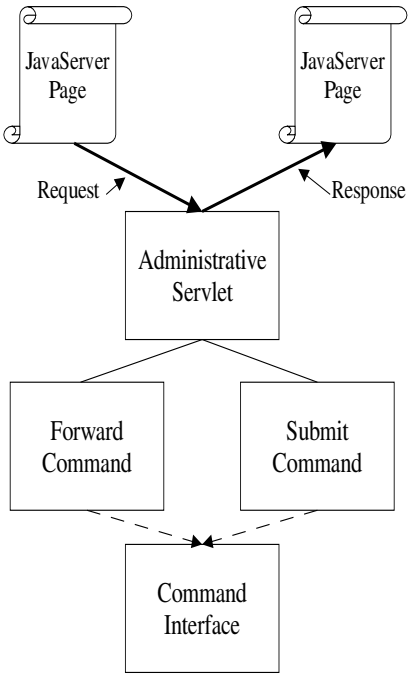


Figure 4. An administrative servlet controls flow between pages. The JavaServer Page sends its request to the servlet and specifies the command to execute. This command could be a simple redirect to the next page (Forward Command) or could encapsulate additional actions (such as the Submit Command). All command classes implement a common interface.

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

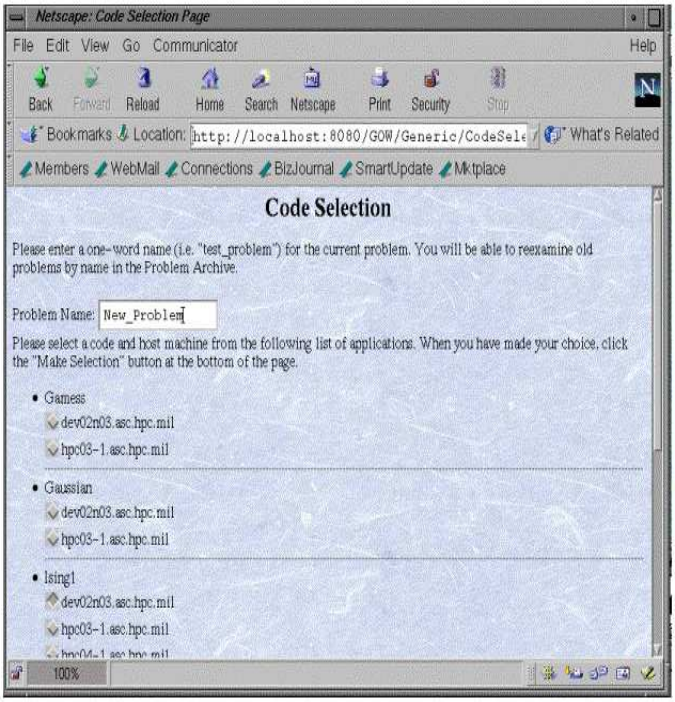


Figure 5. Gateway's "Code Selection" page allows users to choose from different applications available from different hosts. The page is dynamically generated from XML data descriptors.

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

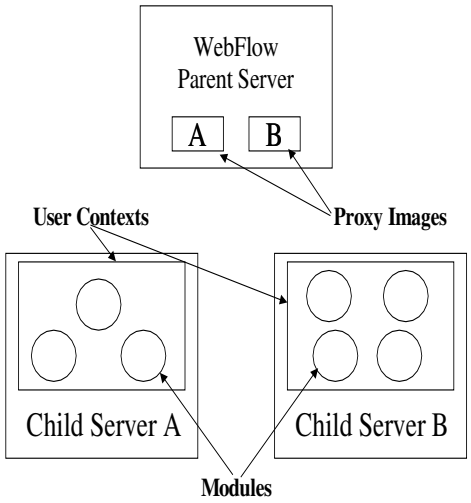


Figure 6. WebFlow servers are organized in a container hierarchy. A parent server maintains proxy images of its two child servers, A and B. The child servers each contain a user context (container) that holds several modules each.

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

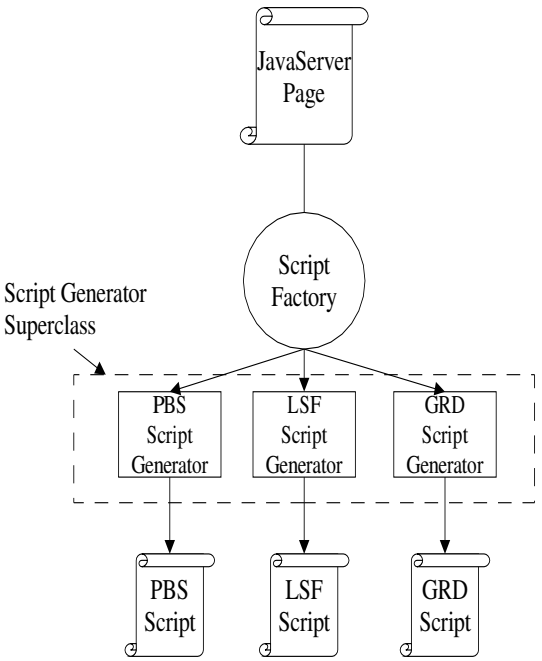


Figure 7. Scripts for particular queuing systems are generated on request from a calling JavaServer Page. A factory JavaBean creates an instance of the appropriate script generator. All script generators extend a common parent.

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

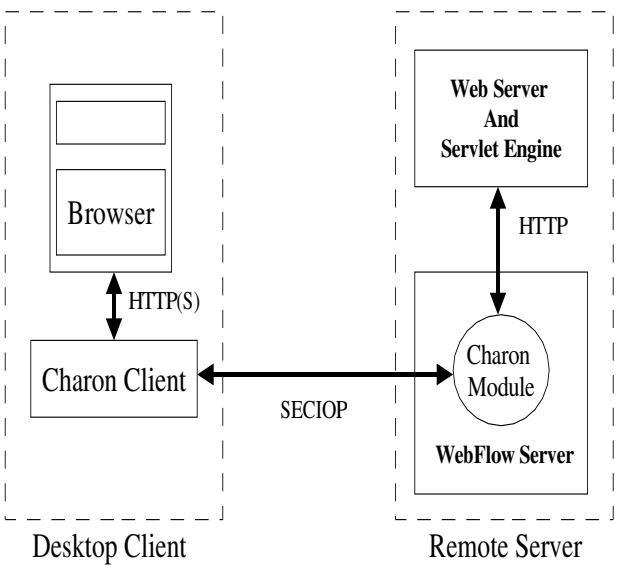


Figure 8. The Charon security module is used to tunnel HTTP requests over a secure CORBA wire protocol.

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls