Web based metacomputing

Tomasz Haupt, Erol Akarsu, Geoffrey Fox, Wojtek Furmanski
Northeast Parallel Architecture Center at Syracuse University

Abstract
Programming tools that are simultaneously sustainable, highly functional, robust and easy to use have been hard to come by in the HPCC arena. This is partially due to the difficulty in developing sophisticated customized systems for what is a relatively small part of the worldwide computing enterprise. Thus we have developed a new strategy - termed HPcc High Performance Commodity Computing[1] - which builds HPCC programming tools on top of the remarkable new software infrastructure being built for the commercial web and distributed object areas. We add high performance to commodity systems using multi tier architecture with GLOBUS metacomputing toolkit as the backend of a middle tier of commodity web and object servers. We have demonstrated the fully functional prototype of WebFlow during Alliance'98 meeting.

Introduction

Programming tools that are simultaneously sustainable, highly functional, robust and easy to use have been hard to come by in the HPCC arena. This is partially due to the difficulty in developing sophisticated customized systems for what is a relatively small part of the worldwide computing enterprise. Thus we have developed a new strategy - termed HPcc High Performance Commodity Computing[1] - which builds HPCC programming tools on top of the remarkable new software infrastructure being built for the commercial web and distributed object areas.

This leverage of a huge industry investment naturally delivers tools with the desired properties with the one (albeit critical) exception that high performance is not guaranteed. Our approach automatically gives the user access to the full range of commercial capabilities (e.g. databases and compute servers), pervasive access from all platforms and natural incremental enhancement as the industry software juggernaut continues to deliver software systems of rapidly increasing power.

We add high performance to commodity systems using a multi tier architecture with traditional HPCC technologies such as MPI and HPF supported as the backend of a middle tier of commodity web and object servers.

Figure 1. Top view of the WebFlow design.

Our research addresses needs for high level programming environments and tools to support distance computing on heterogeneous distributed commodity platforms and high-speed networks, spanning across labs and facilities. More specifically, we are developing WebFlow - a scalable, high level, commodity standards based HPDC system that integrates (c.f. Fig. 1):

- High-level front-ends for visual programming, steering, run-time data analysis and visualization, and collaboration built on top of the Web and OO commodity standards (Tier 1).
- Distributed object-based, scalable, and reusable Web server and Object broker Middleware (Tier 2)
- High Performance Backend implemented using the metacomputing toolkit of GLOBUS (Tier 3)

Note this can be applied to either parallel or metacomputing applications, and provides a uniform cross platform high level computing environment. We have demonstrated the fully functional prototype of WebFlow during Alliance'98 meeting

WebFlow Overview

The visual HPDC framework introduced by this project offers an intuitive Web browser based interface and a uniform point of interactive control for a variety of computational modules and applications, running at various labs on different platforms and networks. New applications can be composed dynamically from reusable components just by clicking on visual module icons, dragging them into the active WebFlow editor area, and linking by drawing the required connection lines. The modules are executed using Globus[2] optimized components combined with the pervasive commodity services where native high performance versions are not available. For instance today one links Globus controlled MPI programs to WebFlow (Java connected) Windows NT and database executables. When Globus gets extended to full PC support, the default WebFlow implementation is replaced by the high performance code.

Individual modules are typically represented by visualization, control, steering or collaboration applets, and the system also offers visual monitoring, debugging and administration of the whole distributed applications and the underlying metacomputing resources. In the future, WebFlow will offer tools for easy conversion of existing (sequential, parallel or distributed) applications to visual modules via suitable CORBA[3], COM[4] or JavaBeans[5] based wrapper/proxy techniques.

New applications created within the WebFlow framework follows a natural modular design in which one accumulates a comprehensive, problem domain specific library of modules in the first phase of a project. Then one would explore the computational challenges of the project in a visual interactive mode, trying to compose the optimal solution of a problem in a sequence of on-the-fly trial applications. The scripting capabilities of WebFlow coupled with database support for session journalizing will facilitate playback and reconstructing optimal designs discovered during such rapid prototyping sessions.

For the parallel object and module developers, we will also provide finer grain visual and scripted parallel software development tools using the new Uniform Modeling Language (UML)[3], recently accepted as an OMG standard. UML offers a spectrum of diagrammatic techniques that allow one to address various stages of software process and several hierarchy layers of a complex software system. In this way, WebFlow will combine the features of UML based visual tools such as Rational Rose with both high performance and the proven value of data flow based visual programming environments such as Khoros and AVS.

Our technology goal is to build a high-level user friendly commodity software based visual programming and runtime environment for HPDC. We believe that such an ambitious and generic framework can be successfully built only when closely related to some specific large scale application domains which can provide specification requirements, testbeds and user feedback during the whole course of the system design, prototyping, development, testing and deployment. We view NCSA Alliance and DoD modernization programs as attractive application environments for HPcc due to its unique, clear mission and advanced computational challenges, opportunities and requirements.

WebFlow is a particular programming paradigm implemented over a virtual Web accessible metacomputer and given by a dataflow-programming model (other models under experimentation include data parallel, collaboratory and televirtual paradigms). WebFlow application is given by a computational graph visually edited by end-users, using Java applets.

Modules are written by module developers, people who have only limited knowledge of the system on which the modules will run. They not need concern themselves with issues such as: allocating and running the modules on various machines, creating connections among the modules, sending and receiving data across these connections, or running several modules concurrently on one machine. The WebFlow system hides these management and coordination functions from the developers, allowing them to concentrate on the modules being developed.

Three-Tier Architecture of the WebFlow

In our approach we adopt the integrative methodology i.e. we setup a multiple-standards based framework in which the best assets of various approaches accumulate and cooperate rather than compete. We start the design from the middleware, which offers a core or a `bus' of modern 3-tier systems, and we adopt Java as the most efficient implementation language for the complex control required by the multi-server middleware. Atomic encapsulation units of WebFlow computations are called modules and they communicate by sending objects along channels attached to a module. Modules can be dynamically created, connected, scheduled, run, relocated and destroyed.

The Middle-Tier

Our prototype WebFlow system [WebFlow] is given by a mesh of Java enhanced Web Servers [Apache], running servlets that manage and coordinate distributed computation. This management is currently implemented in terms of the three servlets: Session Manager, Module Manager, and Connection Manager. These servlets are URL addressable and can offer dynamic information about their services and current state. Each of them can also communicate with each other through sockets. Servlets are persistent and application independent.

Session Manager. The Session Manager is the part of the system in charge of accepting user commands from the front end, and executing them by sending requests to the rest of the system. The user requests that the Session Manager honors are creating a new module, connecting two ports, running the application, and destroying the application. Since the Session Manager and the front end generally reside on separate machines, the Session Manager keeps a representation of the application that the user is building, much like the representation stored in the front end. The difference between these two representations is that the Session Manager needs to worry about the machines on which each of the modules has been started, while the front-end worries about the position of the representation of the module on the screen. The Session Manager acts like a server for the front end, but uses the services of the Module and Connection Managers. All of the requests received from the user are satisfied by a series of requests to the Module and Connection Managers, which store the actual modules and ports.

Module Manager. The Module Manager is in charge of running modules on demand. When creation of a module is requested, that request is sent to the Module Manager residing on the particular machine on which the module should be run. The Module Manager creates a separate thread for the module (thus enabling concurrent execution of multiple modules), and loads the module code, making the module ready for execution. Upon receipt of a request for running a module, the Module Manager simply calls a run method, which each module is required to have. That method is written by the module developer, and implements the module's functionality.

Upon receipt of a request for destroying a module, the Module Manager first stops the thread of execution of the module, then calls the special destroy method. The destroy method is again written by the module developer, and it performs all the clean-up operations deemed necessary by the developer.

Connection Manager. The Connection Manager is in charge of establishing connections between modules, or more precisely, between input and output ports of the modules. As the modules can be executed on different machines, the Connection Manager is capable of creating connections across the network. In such a case it serves as a client to the peer Connection Manager on the remote WebFlow server. The handshaking between the Managers follows a custom protocol.

Although our prototype implementation of the WebFlow proved to be very successful, we are not satisfied with this to large extend custom solution. Pursuing HPcc goals, we would prefer to base our implementation on the emerging standards for distributed objects, and takes the full advantage of the possible leverages realized by employing commercial technologies. Our research led us to the following observations.

While the "Java Platform" or "100% Pure Java" philosophy is being advocated by Sun Microsystems, industry consortium led by the OMG pursues a multi-language approach built around the CORBA model. It has been recently observed that Java and CORBA technologies form a perfect match as two complementary enabling technologies for distributed system engineering. In such a hybrid approach, referred to as Object Web [ObjectWeb], CORBA is offering the base language-independent model for distributed objects and Java offers a language-specific implementation engine for the CORBA brokers and servers.

Meanwhile, other total solution candidates for distributed objects/components are emerging such as DCOM by Microsoft or WOM (Web Object Model) by the World-Wide Web Consortium. However, standards in this area and interoperability patterns between various approaches are still in the early formation stage. A closer inspection of the distributed object/component standard candidates indicates that, while each of the approaches claims to offer the complete solution, each of them in fact excels only in specific selected aspects of the required master framework. Indeed, it seems that WOM is the easiest, DCOM the fastest, pure Java the most elegant and CORBA the most realistic complete solution.

Consequently, we plan to adopt CORBA as the base distributed object model at the Intranet level, and the Web as the worldwide distributed object model. System scalability requires fuzzy, transparent boundaries between Intranet and Internet domains, which therefore translates into the request of integrating the CORBA and Web technologies. We implement it by building a JWORB [JWORB] which is a multi-protocol extensible server written in Java. The base server has HTTP and IIOP protocol support. It can serve documents as an HTTP Server and it handles the IIOP connections as an Object Request Broker. As an HTTP server, JWORB supports base Web page services, Servlet (Java Servlet API) and CGI 1.1 mechanisms. In its CORBA capacity, JWORB is currently offering the base remote method invocation services via CDR based IIOP and we are now implementing the Interface Repository, Portable Object Adapter and selected Common Object Services.

The new JWORB based WebFlow will addresses the integration of the CORBA component, DCOM component and the Enterprise JavaBeans models. Armed with a Java based CORBA platform such as JWORB to be soon augmented by the CORBA/COM bridge, we will be able to freely experiment with and mix-and-match all these component standard candidates.

The Front End

The WebFlow Applet is the front end of the system. Through it, the users can request new modules to be initialized, their ports connected, the whole application ran, and finally destroyed.

Figure 2. WebFlow Front End Applet

The WebFlow editor provides an intuitive environment to visually compose (click-drag-and-drop) a chain of data-flow computations from preexisting modules. In the edit mode, modules can be added to or removed from the existing network, as well as connections between the modules can be updated. Once created network can be saved (on the server side) to be restored at a later time. The workload can be distributed among several WebFlow nodes (WebFlow servers) with the interprocessor communications taken care of by the middle-tier services. More, thanks to the interface to the Globus system in the backend, execution of particular modules can be delegated to powerful HPCC systems. In the run mode, the visual representation of the metaaplication is passed to the middle-tier by sending a series of requests (module instantiation, intermodule communications) to the Session Manager.

The control of the module execution is exercised not only by sending relevant data through the module's input ports. Majority of modules we developed so far requires some additional parameters that can be entered via "Module Controls" (similarly to systems such as AVS). The module controls are Java applets displayed in a card panel of the main WebFlow applet. The communication channels (sockets) between the backend implementation of a module and its front-end Module Controls are generated automatically during the instantiation of the module.

Not all applications follow closely the data flow paradigm. Therefore it is necessary to define an interface so that different front-end package can be "plugged in" into the middle-tier, giving the user a chance to use the front-end that best fits the application at hand. Currently we offer visual editors based on GEF [GEF] and VGJ [VGJ]. In the future, we will add an editor based on the UML [UML] standard, and we will provide an API for creating custom ones.

When designing the WebFlow we assumed that the most important feature of the front-end should be capability to create dynamically many different networks of modules tailored to the application needs. However, working with the real users and the real applications we find out that this assumption is not always true. WebFlow can be used just as a development tool by taking advantage of our graphical authoring tools to create the application (or a suite of applications). Once created, the same application (i.e., network of modules) is to be run without any changes over and over again with the different input sets by the end user. In such a case the design of the front-end should be totally different. The expected functionality is to provide an environment that allows navigate and choose the right application and right data to solve the problem at hand while any technical nuances of the application should be hidden from the end user.

Another important feature of our design is that we introduce a platform independent, web accessible front-end to a high performance metacomputing environment. Given access to the Internet, the user can create and execute his or her application using adequate computational resources anywhere anytime, even from a laptop personal computer. It is the responsibility of the middle-tier to identify and allocate resources, and to provide access to the data.

In the current implementation of the WebFlow we ignored the issues of security. Again, in agreement with our HPcc strategy, we closely watch development of the industry standards. At this time, the SSL suite of protocols is clearly the dominant technology for authorization, mutual authentication and encryption mechanisms. The most recent release of Globus implements SSL-based security features. In order to access Globus high performance computational resources, the user must produce an encrypted certificate digitally signed by the Globus certificate authority, and it return, the Globus side (more precisely, GRAM gatekeeper) presents its own certificate to the user. This mutual authentication is necessary for exchange of encrypted messages between the two parties. However, the authorization to use the resources is granted by the system administration that owns the resources, and not Globus. We are experimenting with a similar implementation for the WebFlow.

The Back End

The module API is very simple. The module implements a specific WebFlow Java interface, metamodule. In practice, the module developer has to implement three methods: initialize, run, and destroy. The initialize method registers the module itself and its ports to the Session Manager, and establishes the communication between itself and its front-end applet - the module controls. The run method implements the desired functionality of the module, while the destroy method performs clean up after the processing is completed. In particular, the destroy methods closes all socket connections which are never destroyed by the Java garbage collector.

It follows that development of WebFlow modules in Java is straightforward. Taking into account availability of more and more Java APIs, such as JDBC, this allows creating quite powerful, portable applications in the pure Java. To convert existing applications written in languages different than Java, the Java native interface can be used. Indeed, at Supercomputing `97 in San Jose, California, we demonstrated a HPF application run under control of WebFlow [SC97]. The WebFlow front-end gave us control to launch the application on a remote parallel computer, extract data at the runtime, and process them by WebFlow modules written in Java running on the local machine. The runtime data extraction was facilitated by the DARP system [DARP] converted to a WebFlow module.

For a more complex metaapplications more sophisticated backend solution is needed. As usual we go for a commodity solution. Since commercial solutions are practically nonexistent, in this case we use technology that comes from the academic environment: the metacomputing toolkit of Globus. The Globus toolkit provides all functionality we need. The underlying technology is a high performance communication library: Nexus. MDS (Metacomputing Directory Services) allows for the resource identification, while GRAM (Globus Resource Allocation Manager) provides secure mechanisms to allocate and scheduling of the resources. GASS package (Global Access to the Secondary Storage) implements high performance, secure data transfer which is augmented with RIO (Remote Input/Output) library that provides access to parallel data file systems.

WebFlow interacts with the Globus via GRAM gatekeeper. A dedicated WebFlow module serves as a proxy of the gatekeeper client, which in turn sends requests to GRAM. Currently, the proxy is implemented using the Java native interface. However, in collaboration with the Globus development team, we are working on a pure Java implementation of the gatekeeper client.

At this time GASS supports only the Globus native x-gass protocol, which restrict its use to systems where Globus is installed. We expect support for other protocols, notably ftp, soon. This will allow using GASS secure mechanism for data transfer from and to systems outside the Globus domain, under control of WebFlow. We also collaborate with the Globus development team to build support for other protocols, HTTP and LDAP. In particular support for HTTP will allow for implementing data filtering on-the-fly, as the URL given to GASS may point not to the data directly, but a servlet or CGI script instead.

Applications of WebFlow

As a test application for the WebFlow we selected Quantum Simulations [QS]. This application can be characterized as follows (c.f. fig.3). A chain of high performance applications (both commercial packages such as GAUSSIAN or GAMESS or custom developed) is run repeatedly for different data sets. Each application can be run on several different (multiprocessor) platforms, and consequently, input and output files must be moved between machines. The researcher visually inspects output files; if necessary applications are rerun with modified input parameters. The output file of one application in the chain is the input of the next one, after a suitable format conversion.

Figure 3. WebFlow application: Quantum Simulations

This example mataapplication demonstrates strength of our WebFlow approach. The WebFlow editor provides an intuitive environment to visually compose the chain of data-flow computations from preexisting modules. The modules encapsulate many different classes of applications: from massively parallel to custom developed auxiliary programs to commodity commercial ones (such as DBMS or visualization packages). The seamless integration of such heterogeneous software components is achieved by employing distributed objects technologies in the middle tier. The high performance part of the backend tier in implemented using the GLOBUS toolkit. In particular, we use MDS (metacomputing directory services) to identify resources, GRAM (Globus resource allocation manager) to allocate resources including mutual, SSL based authentication, and GASS (global access to secondary storage) for a high performance data transfer. The high performance part of the backend is augmented with a commodity DBMS (servicing Permanent Object Manager) and LDAP-based custom directory service to maintain geographically distributed data files generated by the Quantum Simulation project.

We just started using WebFlow for another type of applications, within DoD modernization project. Here the focus is to provide a uniform access via a Web-based interface to specific simulation codes. We will use Globus to allocate remote resources and data transfer. The project involves development of a custom front-end that allows to import data sets coming from varies sources and in different formats (satellite images, GIS, databases, to name a few), as well as generating visually a new data sets.

Conclusions

To summarize, we developed a platform independent, three-tier system: the visual authoring tools implemented in the front end integrated with the middle tier network of servers based on the industry standards and following distributed object paradigm, facilitate seamless integration of commodity software components. In particular, we use the WebFlow as a high level, visual user interface for GLOBUS. This not only makes construction of a metaapplication much easier task for an end user, but also allows combining this state of art HPCC environment with commercial software, including packages available only on Intel-based personal computers.

References

1. I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit", International Journal of Supercomputing Applications, 1997. See also Globus Home Page: http://www.globus.org

2. G. Fox and W. Furmanski, "HPcc as High Performance Commodity Computing", chapter for the "Building National Grid" book by I. Foster and C. Kesselman, http://www.npac.syr.edu/uses/gcf/HPcc/HPcc.html

3. CORBA - OMG Home Page http://www.omg.org

4. COM Home Page http://www.microsoft.com/com

5. Enterprise JavaBeans http://java.sun.com/products/ejb/

6. Quantum Simulations, http://www.ncsa.uiuc.edu/Apps/CMP/cmp-homepage.html

7. Advanced Visualization System, http://www.avs.com/

8. UML Home Page http://www.rational.com/uml

9. GEF: The Graph Editing Framework home page: http://www.ics.uci.edu/pub/arch/gef/

10. VGJ, Visualizing Graphs with Java home page:

http://www.eng.auburn.edu/department/cse/research/graph_drawing/graph_drawing.html

11. D. Bhatia, V. Burzevski, M. Camuseva, G. C. Fox, W. Furmanski and G. Premchandran, "WebFlow - a visual programming paradigm for Web/Java based coarse grain distributed computing", Concurrency: Practice and Experience, Vol. 9 (6), pp. 555-577, June 1997. See also WebFlow Home Page at NPAC: http://osprey7.npac.syr.edu:1998/iwt98/products/webflow.

12. E. Akarsu, G. Fox, T. Haupt, DARP: Data Analysis and Rapid Prototyping Environment for Distribute High Performance Computations, Home Page http://www.npac.syr.edu/projects/hpfi/

13. G. Fox, W. Furmanski and T. Haupt, SC97 handout: High Performance Commodity Computing (HPcc), http://www.npac.syr.edu/users/haupt/SC97/HPccdemos.html

14. JWORB Project Home Page http://osprey7.npac.syr.edu:1998/iwt98/projects/worb, see also G. C. Fox, W. Furmanski and H. T. Ozdemir, "JWORB (Java Web Object Request Broker) based Scalable Middleware for Commodity Based HPDC", submitted to HPDC98.

15.. Client/Server Programming with Java and Corba 2nd. Ed. by Robert Orfali and Dan Harkey, February'98, ISBN:047124578X

16.Apache http Server project, http://www.apache.org

17.Alliance'98 meeting, http://www.npac.syr.edu/users/haupt/WebFlow/papers/alliance98.html