Use of web services for remote distributed-memory visualization

Bob Haimes∗
M.I.T. Dept. of Aeronautics and Astronautics

Mike Giles†
Oxford University Computing Laboratory

Abstract

This paper discusses the use of web services as a replacement for message-passing in remote visualization, linking desktop visualization to a data source on a distributed-memory parallel system such as a PC cluster. The security benefits of web services are presented, along with the details of the emulation of symmetric message-passing through asymmetric client-server web service RPC communication. The ease-of-use and performance of the gSOAP C/C++ web service toolkit is also emphasized.

1 Introduction

1.1 History of pV3

This paper concerns the development of a new version of the pV3 software package [7, 8, 9] for the remote visualization of 3D grid-based data on a distributed-memory parallel system.

pV3 was initially developed in 1993 in response to the increasing use of distributed-memory parallel computers. It was clear that the huge quantities of data being produced by computations on such systems could not be handled by the desktop machines being used for visualization. Conceptually, at its simplest, the model for remote visualization is as shown in Figure 1, with a data source D on a distributed-memory parallel system, and a viewer V on a desktop machine. A small amount of control data passes from V to D, and a large amount of visualization data moves from D to V.

This model encompasses both co-visualization and post-processing (in 2 models). In the former, the data source D is an on-going parallel computation, so the viewer V is visualizing data as soon as it has been computed. This is extremely useful during the development and debugging of new parallel application codes, but it is usually not viable, as is, in a “production” setting because of long execution times. In traditional post-processing, the parallel computation has been performed previously, and the (potentially voluminous) results stored on disk. The parallel data source D reads the data from disk, and perhaps performs some limited amount of computation to calculate derived quantities (such as computing the pressure in a CFD calculation given the values of the density, momentum and energy). In this post-processing mode, the parallel visualization is necessary simply because the quantity of data is too large to be handled on the desktop machine being used for the viewer.

To resolve the issue of long running jobs in a production environment another pV3 post-processing mode (the ‘batch’ module) can be invoked. The data source D (again the active computation) need not know if the results are currently being viewed or are to be viewed at some later time. When a batch job starts, the ‘batch’ pV3 module is also started (instead of the viewer). Data is read on where and what tools and probes are to be active and their locations. The results (the tool “extracts”) are collected and written to disk for later playback. This is different from the normal post-processing in that the entire volume of data is not written to disk every iteration. The purpose of the batch subsystem is to write the visualization data to disk. Therefore some knowledge of the underlying hardware must be used to determine where to execute this module. The machine running the batch subsystem should be the computer that contains the output file-system. The end result is something that is not interactive in the placement of tools, but can be thought of as analogous to a wind-tunnel experiment. Some knowledge of the flow must be used in the placement of probes to extract data of interest. If important information is missed (or only found after viewing these results) the ‘tunnel’ will have to re-run adding (or changing the location) of the probes.

∗haimes@mit.edu
†giles@comlab.ox.ac.uk
The special pV3 ‘post-processing’ viewer module is highly interactive in dealing with time. This is due to the fact that the amount of data has been reduced by orders-of-magnitude. Overlapping the disk IO with display rendering further enhances the batch viewer’s performance. This is accomplished via multi-threading. The ‘post-processing’ viewer is the only self-contained module in the pV3 suite. It does not require PVM or any other message passing. Therefore the simulation can be reviewed away from the hardware used in data generation. All that is required is the disk file, the post-processing viewer and the proper graphics workstation.

pV3 also supports collaborative visualization as illustrated in Figure 2. This capability is very useful when there are individuals or groups in different organizations, or within different units of the same organization, who are working together on a common research or design project. There are now two (or more) viewers connected to the same data source. pV3 does not prescribe to the usual mode of collaboration in which there is a master controlling the production of imagery and then parceling the result to the slave viewers. In pV3, all viewers can control completely independent sessions, although there are also options to display soft cursors of the collaborative sessions, to enable one viewer to lock onto the particular viewpoint of another viewer. The software architecture also allows for a mixture of batch and interactive connections to the data source, although there are some restrictions on solution steering under these conditions.

One of the assumptions in the design of pV3 was that a key-limiting factor constraining the performance of a remote visualization system is the bandwidth of the network linking the data source to the viewer. Therefore, it was necessary to minimize the amount of visualization data being transferred from D to V. When visualizing 3D data, it is extremely rare to use techniques such as volume visualization that are based on data from the entire computational grid. This is because the data is usually not represented on a regular orthogonal grid. Usually, what is viewed are “extracts” (constructed geometry), either surfaces, such as iso-surfaces (e.g. surface of constant pressure) or domain boundary surfaces (e.g. surface of an aircraft), or lines (e.g. streamlines) or points (e.g. unsteady particles being convected by the flow)[6]. Hence, to minimize the bandwidth requirements, and also the memory requirements on the desktop machine, pV3 was designed to transfer only the extracts from the data source to the viewer. The decision to transfer 3D extracts, rather than their 2D projection in screen coordinates, allows the viewer to perform the final step of rendering the 3D extracts. This provides great interactivity as the user can pan, zoom, and rotate the extracts without any delays being introduced by communication with the data source. It also delivers excellent performance, making full use of the OpenGL hardware support of modern graphics cards.

Figure 3 presents an expanded view of the elements within the data source. The elements labeled UA are the user’s parallel application, which is either performing a parallel computation (co-visualization) or simply holding the distributed data (post-processing). The elements labeled pV3 are the pV3 clients that connect to the user’s application to generate the extracts and sends them to the viewer.

The communication between the different UA elements (as required for the parallel computation) is performed through MPI message passing. This became established as the standard for high-performance computation on homogeneous distributed-memory parallel systems around the time of pV3’s development.

The communication between the pV3 clients and the pV3 viewer was originally performed by direct PVM message-passing communication, as indicated in Figure 3. This involves the request for the extracts, the extracts themselves and other miscellaneous information (e.g. streamline or particle hand-offs, time framing data, steering information and etc.). PVM was a natural choice for the original distributed-memory parallel systems, such as IBM’s SP2 and clusters of workstations, for which pV3 was first developed. However, later systems such as the Cray T3D/E did not permit direct communication to external machines from all of its processors. To accommodate such hardware, it was necessary to introduce a concentrator, as indicated in Figure 4. This uses MPI to collect all of the extracts and then uses PVM to pass them in a single message (or at least fewer messages) to the viewer. It also accepts the control data from the viewer and distributes it to all or the appropriate pV3 client(s). A positive side effect of this method of communication is improved scalability by reducing the message count and therefore the accumulated latencies.

In general, the design of pV3 has withstood the test of time remarkably well. In the last 12 years, network speeds have increased by a factor of up to 100, with Gigabit connectivity to the desktop becoming commonplace. However, processor speeds have also increased by factor 50-100, and the number of nodes within distributed-memory parallel systems has increased substantially. Therefore, the ratio of computing power to network bandwidth has not changed substantially, and it remains the case that network connectivity is a major bottleneck in remote visualization.
The pV3 concentrator is also ideally suited to the PC clusters that are now in such widespread use within both academia and industry. Typically, these clusters have a front-end node with a firewall and a large RAID file system, and the back-end compute nodes are inaccessible from the outside world. The concentrator solves this problem by running on the front-end node, accumulating the extracts from the clients on the back-end nodes, and then sending them to the viewer.

### 1.2 Web services

The one aspect of pV3, which reflects its age, is the use of PVM message-passing for the communication between the concentrator and the viewer. PVM still has many attractions. For example, it provides the ability for the viewer to connect to the communicator for a period, then disconnect, and then later reconnect. This is very useful when one wants to view data from a long-running application, and it is a capability that is still not available from MPI. However, the user base for PVM is steadily shrinking, with it having been firmly replaced by MPI for tightly coupled parallel computations.

The new choice today for high-level communication between heterogeneous systems is web services. Driven by the requirements of e-commerce, these are based on a set of universally adopted standards:

- SOAP for the RPC protocol [12]
- XML for the data encoding [13]
- HTTP and HTTPS for the transport protocol

enabling easy communication between all major operating systems, including all flavors of Unix and Microsoft Windows.

Web services have two fundamental differences in their operation compared to PVM communication. The first is that they operate on an asymmetric RPC (remote procedure call) basis. There is a server process that provides a service by responding to requests from clients. The client initiates the communication by sending a message to the server. The server acts on the basis of the content in the message, and then returns a response. The asymmetric nature of this interaction is quite different from message passing in which all processes are peers, with each able to send a message to the others.

The second difference is the security/ownership model. In PVM, the user owns all of the processes in the PVM message-passing group. i.e. the user has to have a user account on each of the machines which is involved, and on each machine the PVM process runs under the user's user-id. This is particularly inconvenient in the case of collaborative visualisation.

In contrast, when using web services the server and client processes can belong to different users. Indeed, in e-commerce this is the normal mode of operation. User A initiates a web service process SA on machine MA. User B then starts a web service client process CB on machine MB. When CB contacts SA, it depends on the programming of SA whether it chooses to respond to CB. It might make this decision based on the identity of user B (based, for example, on a public-key X.509 user certificate) or the identity of machine MB, or on the provision of an application password. If it does choose to respond, then it performs the necessary computation under the user-id of user A, and it controls entirely what data it sends back to CB. Thus, user B obtains carefully restricted access to data from machine MA without having a user account on the machine.

This security model is much better suited to the needs of collaborative visualization in an industrial context. For example, consider the case of an aircraft manufacturer A and an aircraft engine manufacturer B collaborating on some research to developed improved aeroacoustic models. If A wishes to show some visualization data to B, then A can provide the data through a web service, without the need to provide user accounts to employees of B. This becomes even more critical in the case of two engine manufacturers B and C who are collaborating on one engine design project, but competing on others, a common situation in today's aerospace industry.

Even if there are ring-fenced computers set aside for collaborative work to avoid the security problems, the use of software such as PVM for communication between commercial organizations and/or universities may be impossible in practice because of corporate firewalls. One of the strengths of web service communication is that it is based on the HTTP/HTTPS transport protocol using the standard ports which among the very few left open by most firewalls.

### 1.3 gSOAP toolkit

Most web service implementations are based on either Java or C#//.NET, but there is nothing in the specification of the SOAP and XML standards which restricts the language of implementation, and a client written in one language can interact with a service written in a different language, as long as both conform to the established protocols. Most scientific applications are written in FORTRAN or C. It is possible to link Java to C using JNI (Java Native Interface) but it is more natural and much simpler to use a native C web service implementation, and in this work we have used the excellent C/C++ gSOAP toolkit developed by R. van Engelen [4, 5, 2]. This is public-domain software available through SourceForge. It has been used in a number of major computing projects, and is distributed by IBM as part of its Web Service Toolkit for Mobile Devices [1], apparently due to its low memory requirements compared to Java implementations.

It is difficult to give a comprehensive overview and assessment of gSOAP: the reader is encouraged to refer to the documentation and success stories provided at [3]. However, we must emphasize its ease-of-use and its range of security and performance features to be discussed later. At the simplest level, one writes a header file defining the syntax of the RPC calls, and then uses a small collection of library functions to generate the web service and the client RPC calls. A large set of demonstration applications is provided in the gSOAP distribution. These provide an excellent starting point so the learning curve is very short. Even a novice C programmer (such as the second author) can generate their own application codes within a few hours. The web service applications can either be hosted as CGI applications within a web server, or run as standalone services bound to a particular port number. The latter approach is the one we have adopted in our work. The former approach is being used by colleagues in a related project [14].

### 2 Replacement of PVM by web services

#### 2.1 Approach

In order to rapidly develop a web services prototype to assess the viability of such an implementation, it was decided to port the existing software by using gSOAP to emulate all of the PVM functions currently used by pV3. The PVM API was therefore preserved so that simple re-linking of the appropriate pV3 module was all that was required to replace the one mode of communication with the other. PVM machine state (data that was maintained via PVM’s daemons) was stored in the PVM emulation code used by the concentrator.

The procedure that allowed for the use of heterogeneous architectures in PVM was performed during the packing/unpacking of data from the messages. To simplify this port, this PVM technique was maintained even though gSOAP supports this procedure through XML. All message data is passed as a simple stream of bytes with a fixed size preamble.
This header is composed of:

- The 4-byte integer 1
- The PVM message ID
- A source identifier
- A destination identifier

The first 4 bytes are examined when the receiving task gets the message. If it is not 1 and not 16777216 then there is some error. If it is 16777216 then byte swapping is required for all types (except character) and is performed during the unpacking operations. Clearly this will only work for IEEE floating-point based architectures, but currently this is not a real limitation.

The PVM and this gSOAP emulated-PVM generate messages that are basically the same length, and so one would expect that the performance would be the same. Although this was never rigorously tested, the “feel” of both running on systems connected by a good network is identical, indicating that the latency inherent in gSOAP is comparable to that within PVM.

### 2.2 Emulation of symmetric message passing

As explained earlier, web services are inherently asymmetric in their communications. If the data source D acts as a web service, and the viewer V acts as a web client, then sending control data from V to D is easy; V simply sends D an RPC request containing the control data, and D returns a null response.

The problem is how to handle the sending of the visualization data from D to V. One option would have been to also make V a web service. Using the gSOAP toolkit, it is possible for single applications to both provide a web service, and to act as a web client in making calls to other web services. However, we decided it was preferable for V not to be a web server. Because of concerns over security, inappropriate/illegal actions by authorized users, and the potential of hackers and viruses, network system managers are increasingly restricting the services that may be run on desktop machines (i.e. ports left open). On the other hand, the data source D will be run on the front-end node of a PC cluster, or similar system, and it is easier to argue for a special treatment for such systems.

Having decided that V is simply a web client, we are faced with the problem of how to transmit messages from D to V. This is handled through the use of threads and locks:

- V spins off a thread TV which sends a null request to D
- D spins off a thread TD to process the request, but this thread is initially stalled via a lock
- the main D process continues; when it has a message to send to V it posts it on a “message board” and releases the lock
- the thread TD picks up the message and sends it to TV as the response to the original null request
- TV posts the message on V’s “message board”, and then sends a new null request to D to await the next message

This process works very well given an operating system which supports efficient multi-threading with locks. Unfortunately, we experienced significant problems in this regard with Linux Red Hat 9, but these were cured by the latest release of glibc.

### 2.3 Security and performance features

The gSOAP toolkit supports a number of features, which are important for security and performance.

Security is provided through SSL encryption based on the OpenSSL package [10] with X.509 public-key certificates. It is also possible to define passwords to be used by clients to authenticate themselves with the service, but we have not yet made use of this feature.

In addition to the on-going cost of encrypting all data, there is a significant overhead in initiating SSL communication. To avoid incurring this overhead on each message, gSOAP supports a standard SOAP “keepalive” feature that maintains an open channel between the client and the web service. As with the SSL feature, this is invoked within the programs for D and V by specifying options to be used in the communications. This includes timeout constants to be used to close the channel in the absence of any traffic. “keepalive” with long timeouts is also important because messages may be open for extended periods due to our emulation of symmetric message passing.

XML uses UTF-8 encoding of data. Like ASCII, this is a very verbose way of representing integer and real variable data, and given the concerns about the network bandwidth between the data source and the viewer, this threatens to give very poor performance compared to PVM when using very low bandwidth networks. Fortunately, there are two solutions to this problem. One is the use of attachments. Just as with email, SOAP supports DIME attachments and so numeric data can be transmitted in binary format as an attachment. The drawback of this solution is that the programmer would have to cope with any incompatibility in binary format between heterogeneous systems. The second solution, which is the one adopted in this work, is to use GZIP compression. This is another communication option proved by gSOAP. There is of course a computational cost to this, but this is very small compared to the overall communication time.

### 3 Future enhancements

#### 3.1 Concentrator — pv3 client communication

At present, the communication between the concentrator and the pv3 clients is performed using MPI. This was an appropriate choice when these were all executing on the nodes of a machine where the front-end node only differs from the rest by having an active off-machine TCP/IP connection. But it is less well suited to execution on PC clusters where the concentrator needs to run on the front-end node while the pv3 clients run on the back-end compute nodes.

Typically, the back-end compute nodes of a PC cluster are coupled by a high-bandwidth, low-latency interconnect such as Myrinet or Gigabit Ethernet with custom low-level drivers to avoid TCP/IP latencies. The front-end node is not always a member of that interconnect. It may even have a different version of the operating system because of considerations of security, stability and cost. The front-end node in the cluster in the Oxford University Computing Laboratory uses a version of Linux Red Hat 7.3 which is kept up-to-date with the latest security patches, and might move to the commercially-supported Red Hat Enterprise Edition in the near future, while the back-end nodes, safely secured behind the NAT firewall, use a much older stable version of Red Hat 7.3.

Consequently, it is difficult to set up MPI to enable communication between processes on the back-end compute nodes and the front-end node. In the future, if manufacturers move towards diskless compute nodes, possibly with stripped-down operating systems, this may become even more difficult. Vendors remain focused on the use of PC clusters as facilities for running parallel batch jobs. Insufficient attention is given to the needs of parallel visualization
or any requirement for getting data from an active simulation. It is quite odd the extent to which there seems to be a large disconnect between the Grid and Parallel Computing communities when, in fact, they are trying to solve similar problems!

The solution to this issue may be to switch to web services for the communication between the pV3 clients and the concentrator, so that MPI is only used for communication between the processes running on the back-end compute nodes. See Figure 5 for an example of this proposed communication.

3.2 More general data service

The current implementation is not a true web service in that it does not define and publish the specifications for the services that it offers. If, in the future, it is re-implemented as a native web service, with a proper service specification, then it would be natural to specify the services in a way which makes it possible for viewers based on different visualization toolkits to view the same data [11].

Another possibility is to expand on the idea of the web service for other purposes. For example, since its first development, pV3 has provided a mechanism for computational steering, through which some of the control parameters sent from the viewer to the pV3 clients are passed through to the user’s application. However, a more general web service definition could provide computational steering as a separate capability, independent of the use of the pV3 viewer.

The web service mechanism could also be used to link multi-disciplinary application codes, for example linking a CFD simulation (performed in parallel on a PC cluster) of the flow over an aircraft to a structural dynamics calculation (on a small shared-memory system) of the vibration of the aircraft’s wings. As long as the high performance and low latency of MPI is not required, this use of web services is probably the most flexible approach to building multi-disciplinary applications.

The easy interoperability between web services on Linux/Unix and Windows platforms is also a bonus, as commercial organisations often prefer a combination of Windows on desktop machines and Linux/Unix on back-office servers and compute clusters.

4 Conclusions

This work has successfully demonstrated the use of web services for communication between heterogeneous systems for the purposes of remote visualization. It has been accomplished using the gSOAP toolkit, a very easy to use C/C++ web service implementation which is ideally suited to scientific applications written in C. By making use of the advanced features within gSOAP, the communication performance is comparable to PVM. However, the different nature of web services makes them more suitable than PVM message-passing for applications involving collaborative working between different organisations.

Acknowledgements

This research was funded by the UK Engineering and Physical Sciences Research Council through the gViz and GEODISE eScience projects. We have benefited greatly from discussions with the other participants within the gViz project, in particular Prof. Ken Brodlie and Dr. Jason Wood of Leeds University.

We are very grateful to Prof. Robert van Engelen of Florida State University for his development of the gSOAP web services toolkit.

References
