Appears in Parallel and Distributed Processing Techniques and Applications (PDPTA99), Aug, 1999, Las Vegas, NV.

Preliminary Report on the Design of a Framework for Distributed Visualization

Martin Aeschlimann, Peter Dinda, Loukas Kallivokas,
Julio López, Bruce Lowekamp, and David O'Hallaron

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA, U.S.A.
{aeschli, pdinda, loukas, jclopez, lowekamp, droh}@cs.cmu.edu

Abstract *The paper is a preliminary report on the design of a framework, called Dv, for building interactive distributed visualizations on computational grids. The framework is based on a form of mobile object, called an active frame, that consists of application data and a program that manipulates the data. The system provides a flexible framework for building distributed applications that are performance-portable in the presence of heterogeneous resources and that adapt to dynamic changes in the status of system resources such as processor cycles and network bandwidth.*

Keywords: distributed visualization, active frames, computational grids.

1 Introduction

In a typical Internet application, a client requests content from a server, which obtains the content and sends it back to the client. The content that the server obtains can be either static or dynamic. *Static content* is stored on the server in files or databases, while *dynamic content* is generated on-the-fly by the server. For most applications, the amount of computation needed to satisfy a request is fairly small. However, there is an emerging class of Internet applications that require large amounts of computation before the dynamic content can be sent back to the client. An important example of this kind of compute-intensive Internet application is the interactive visualization of the massive scientific datasets that are generated by computer simulations.

For example, we are part of a group of seismologists and engineers (called the Quake project) that is developing the capability for predicting, by computer simulation, the ground motion of large basins during strong earthquakes in the Greater Los Angeles and Kobe, Japan basins [1]. The Quake ground motion simulations, which must be run remotely on supercomputers because of their complexity, can produce datasets on the order of hundreds of gigabytes to terabytes of floating point numbers that represent displacements of points in the earth during an earthquake. In order to interpret the numbers, the datasets must be transformed into visualizations through a sequence of complex transformations such as interpolating an irregular grid onto a smaller regular grid, computing isosurfaces, setting up the scene, and rendering the scene into an image. This process is described in more detail in Section 2.

Currently, Quake visualizations are created on request by the graphics department of the supercomputing center where the simulations are computed, with a response time of weeks or months. This is unsatisfactory to the scientists and engineers, who would rather manipulate the datasets interactively from their personal desktop and laptop systems. However, manipulating huge datasets interactively is difficult because the necessary resources are limited, heterogeneous, and dynamic.

Limited resources. In an ideal world, we would simply copy the entire dataset from the remote supercomputing center and manipulate it locally. Unfortunately, copying multi-terabyte files is not feasible because of limited network bandwidth. And even if we had access to sufficient network bandwidth, we would not have the resources at our local site to store and backup multi-terabyte files. The crucial implications are that (1) Large scientific datasets must reside on the remote site where they are computed; and (2) Interactive visualization applications that manipulate these remote datasets must be distributed, with the computation partitioned across hosts at the remote and local sites. Thus, we need a framework for building distributed visualization applications. Since visualization algorithms are complex and difficult to develop, the framework must also be able to incorporate existing visualization packages such as AVS [9] and vtk [7].

Heterogeneous resources. The resources available for distributed visualizations are wildly heterogeneous. Networks have different bandwidths and hosts have different computing power, memory, and display capabilities. Thus, the appropriate partitioning of a distributed visualization application depends on the available computing, networking, and storage resources. A distributed visualization framework must provide tools for building applications that are *performance portable*, in the sense that applications can sense the available resources at load time and then configure themselves to run as well as possible given those resources. For example, we should be able to run the same distributed visualization application from a PDA attached to a wireless link, a laptop attached to a slow 10 Mb/s LAN, or a powerful desktop system attached to a fast 1 Gb/s LAN.

Dynamic resources. Finally, we must also deal with the fact that most computing and networking resources are shared. Networks become more or less congested, processors become more or less loaded, and thus the availability of these resources changes over time. In general, a distributed visualization framework must provide tools for building applications that are *adaptive*, in the sense that applications can sense the availability of compute and network resources at run time, and dynamically readjust themselves to run as well as possible given those resources.

This paper is a preliminary report on the high-level design of a framework called *Dv* (distributed visualization) for building interactive distributed visualizations of massive scientific datasets. A Dv application runs on a *computational grid* of hosts[5] that communicate over the Internet. Section 3 introduces the Dv model for providing services on such a grid.

The Dv framework is based on the central notion of an *active frame*, which is an application-level mobile object that contains both application data and a program that manipulates the data. Active frames are processed by *active frame servers* running on hosts at the remote and local sites. Each server waits for an input active frame to arrive at a well-defined port, then executes the active frame program, which applies one or more transformations to the active frame data, generates a new output active frame, and computes the address of the next active frame server. Finally, the server sends the output frame to the next server in the grid.

Active frames and their servers are described in more detail in Section 4. The intent is to provide an application-level mechanism that is simple enough to deploy widely, flexible enough to build applications that are both performance-portable and adaptive, and yet reasonably efficient. Also, the idea of active frames is quite general and could prove useful for other types of distributed signal processing and multimedia applications. However, our work is still preliminary and we do not yet have any strong evidence to support these claims. Nonetheless, performance results from a prototype implementation of an active frame server (detailed in Section 4) are encouraging and suggest that we do not have to pay an inordinate cost for the flexibility of the active frame mechanism.

Finally, Section 5 describes how a collection of active frame servers are customized and composed to form a Dv application. This section also describes in a general sense the hooks that Dv provides — in conjunction with system monitoring tools like the CMU Remos system [3, 6] or Network Weather Service [11] — to create applications that are both performance-portable and adaptive.

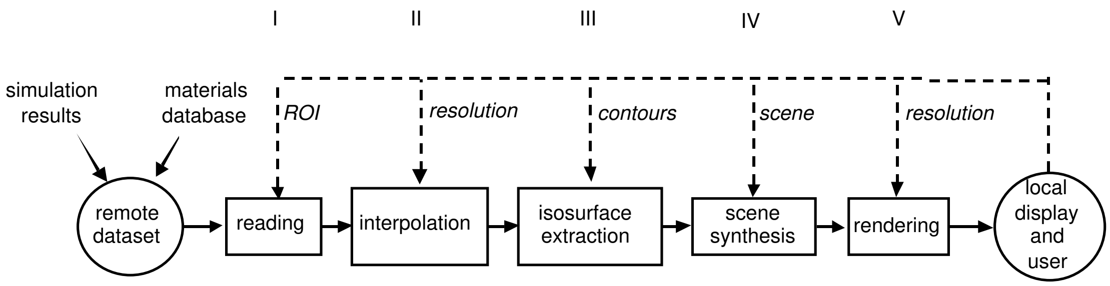

Figure 1: An earthquake visualization application.

2 Motivating application

The initial motivating application for the Dv framework is the visualization of massive datasets that model the motion of the ground during strong earthquakes. The datasets are produced by simulations developed by the Quake project [1]. Quake simulations manipulate large unstructured finite element meshes. For a moderately sized basin such as the San Fernando Valley of Southern California, the meshes consist of 15M nodes and 80M tetrahedral elements. A simulation of 60 seconds of shaking can generate on the order of 6 TB of data.

In a typical Quake visualization, the dataset is stored at a *remote site* and consists of thousands of 3D *frames*, where each frame records the displacement amplitude (i.e., the amount of motion at that spot in the earth) at each node of a large unstructured mesh. A user at the *local site* interactively requests data from some *region of interest* (ROI) in the dataset. The region of interest can be expressed both in space within a frame or in time across multiple frames. If the user requests data from a single frame, then the result at the local site is a still image. If the user requests multiple frames, then the result at the local site is an animation.

The process of interactive visualization can be thought of as a series of queries to a massive dataset. Before the data is presented to the user it is processed by a sequence of filters. The computations are usually expressed in the form of a flowgraph.

Figure 1 shows the form of a typical Quake visualization flowgraph. Stage I reads the appropriate part of the dataset, as indicated by the ROI. Stage II interpolates the displacements from the original unstructured mesh onto a smaller regular mesh whose size is determined by an adjustable resolution parameter. Stage III computes isosurfaces (contours) based on the soil densities and the displacement amplitudes. In stage IV the scene is synthesized according to various scene parameters such as point of view and camera distance. Finally, the scene is rendered from polygons into pixels that are displayed on a monitor. The resolution of the displayed image is determined by a resolution parameter.

3 Grid service model

We envision that applications such as the distributed earthquake visualization in Figure 1 will be provided as services that operate on computational grids of hosts.

Figure 2 shows the basic idea. Suppose that a scientist at some site has an interesting dataset that he wants to make available to members of the research community to visualize and otherwise manipulate. We will refer to him as the *service provider* and to his site as the *remote site*. As part of creating the service, the service provider designates a collection of $m \geq 1$ hosts under his administrative control that are available for running programs that might be required to satisfy service requests from other sites. At least one of these m hosts has access to the dataset and in general the m hosts will be shared with other applications. The m hosts will typically be physically located at the remote site, but in general this is not a requirement; some of the m hosts could be acquired using grid-based resource management services such as Globus [4].

A scientist at another site (i.e., a *service user* at

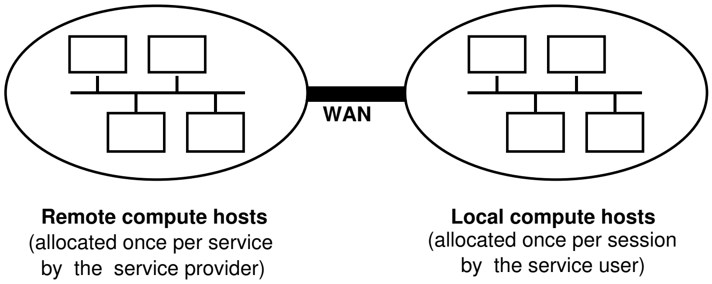

Figure 2: Grid service model.

the *local site*) visualizes or manipulates the dataset at the remote site by issuing a finite series of requests to a host on the remote site. The period of time between the first request and the last request is called a *session*. Before the scientist begins a particular session, he must designate a collection of $n \geq 1$ hosts that are available for running any programs that are needed to satisfy service requests. As with the m remote hosts, the n local hosts are not necessarily physically located at the local site, although most likely they will be.

The main idea is that there are a total of $m + n$ hosts available to perform the necessary computations during the session, m of which were designated by the service provider to handle all requests from all service users, and n of which were allocated by the service user at the local site for that session. The motivation for choosing this particular model is that it allows service providers to bound the number of compute cycles that will be consumed at their site by any particular service. However, it also gives the service user the option to contribute resources that might help reduce the response times for their service requests. A service user who needs better response times can always do so by increasing the value of n.

Note that our model is a simple generalization of some other grid service models. For example, the usual client/server Internet model assumes $m = n = 1$. Netsolve, a network-enabled solver toolkit from University of Tennessee [2], has $m \geq 1$ and $n = 1$.

4 Active Frames

At the core of the Dv framework is the notion of an *active frame*, which is an application-level mobile object that contains *frame data* and a *frame program* that manipulates the data. The frame program implements a single *run()* method

```
interface ActiveFrame {
    NextHost run();
}
```

that computes an output frame and returns the network location of the destination host for the output frame.

Active frames are processed by *active frame servers*, which are processes that run on each host in the computational grid described in the previous section. A server consists of two components: an application-independent interpreter that executes active frame programs, and an optional collection of application-specific library routines that can be called from the frame program. Figure 3 shows the basic architecture.

At run-time, an active frame server waits on a well-defined port for an input active frame to arrive from the network. The server reads the input from the network, extracts and demarshals the frame program and data, and passes them to the interpreter, which executes the *run()* method on the input frame data to produce a new output frame data. After the execution finishes, the server marshals the output frame data and the frame program into a new output frame, and sends this frame to the destination host.

The idea of bundling programs with network data is certainly not new to active frames. This no-

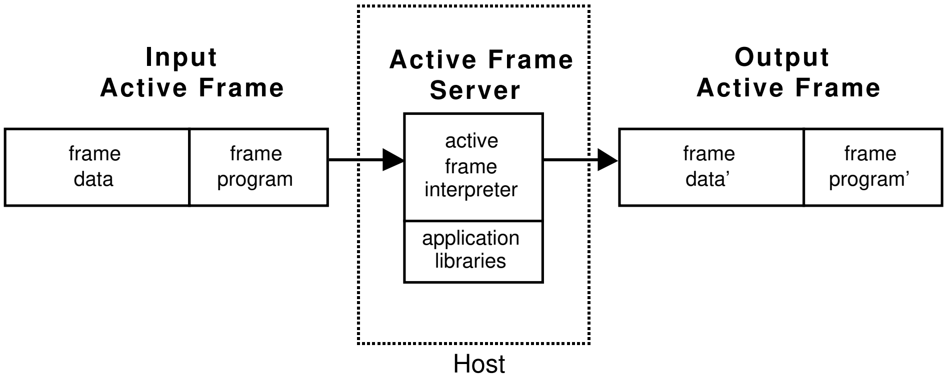

Figure 3: Active frame server.

tion has been exploited effectively by active messages in the context of parallel processing [10], and by active networks in the context of adding additional functionality to network routers [8]. The rationale behind active frames is the suspicion that this same notion will also prove effective in the context of grid computing.

The active frame is clearly a general and flexible mechanism. A potential disadvantage is that this flexibility introduces performance overhead. Additional space, and thus bandwidth, is required by the frame program, and extra processing time is necessary to interpret and execute the frame program. However, for the distributed visualization applications we are interested in, we expect the size of the frame data and the time required to process it to swamp the overhead associated with processing the frame program.

A simple experiment provides some preliminary results to support this conjecture. The experiment compares the cost of processing an active frame to that of processing a "passive frame", which is simply an active frame with no frame program. The operation in each case is the extraction of a plane from a Quake ground motion dataset with 7K unstructured mesh nodes and 35K tetrahedral elements. The active frame version is written in Java with calls to a native C++ method to perform the plane extraction. The passive frame version is written in C++, and the call to the extract operation is defined statically on the server. The size of the frame data is 820 KB, which is quite small and makes it more difficult to amortize any performance overheads associated with the frame overheads. The measurements were gathered on three Pentium II/450 MHz hosts — one for the server, one for the source of the input frame, and one for the sink of the output frame — each with 256 MB of memory running Linux 2.0, and connected to the others by a 100 Mb switched ethernet network.

Figure 4 shows the results of the experiment. The active frame version introduces an additional delay of 119 ms, which represents a 16% overhead of the total processing time. The receive phase in the active frame version is 23% slower than the receive phase in the passive frame version, accounting for 112 ms of the total delay. This is due to the demarshalling of the frame program and its interpretation. The additional delay introduced in the compute and send phases is relatively small.

	Passive frames	Active frames
Operation	Elapsed time (ms)	Elapsed time (ms)
Receive	485.95 (66%)	598.13 (70%)
Compute	172.04 (23%)	175.00 (20%)
Send	80.79 (10%)	84.46 (10%)

Figure 4: Performance of a prototype active frame server.

The results show that any improvement in the receive phase of the process would have a major impact in the overall performance of the system. In particular, the use of a caching strategy to avoid copying the static parts of the datasets with every frame would offer a significant performance incre-

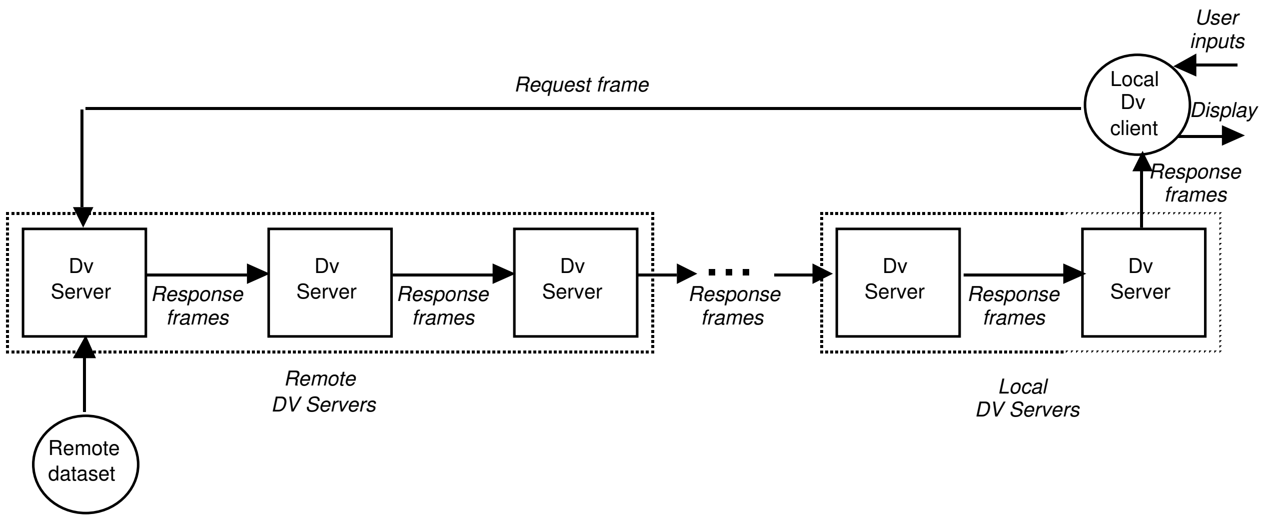

Figure 5: The Dv framework.

ment. For example, in the first stage of the Quake visualization application, the structure of the mesh does not change from one frame to the next one. Only the amplitude values vary. However, in many situations the structure of the output dataset varies in every step because it depends on the attribute values of the input, making it less useful to cache the structure.

5 Dv overview

Figure 5 shows the basic architecture of a Dv system. The system is a collection of identical active frame servers (Dv servers) running on the hosts of a computational grid, plus an additional active frame server (the local Dv client) specialized with a user interface that accepts user inputs and displays rendered images. Each Dv server is an active frame server specialized with an existing visualization library. The initial implementation of Dv supports vtk, a powerful and comprehensive open-source visualization library from GE [7].

During the course of a visualization session, the Dv client sends a series of *request frames* to a Dv server (the *request server*) that has direct access to the remote dataset. Each request frame contains visualization parameters such as region of interest and resolution, a description of the visualization flowgraph, a scheduler that assigns flowgraph nodes to Dv servers, and a frame generator that produces a sequence of one or more *response frames* originating from the request server. Response frames pass from server to server until they arrive at and are processed by the Dv client.

Figure 6 shows an example of a Dv version of the earthquake ground motion visualization from Figure 1. In this example, each frame carries a program that statically assigns stage I to the first remote Dv server, stages II and III to the second remote Dv server, stage IV to the local Dv server, and stage V to ythe local Dv client.)

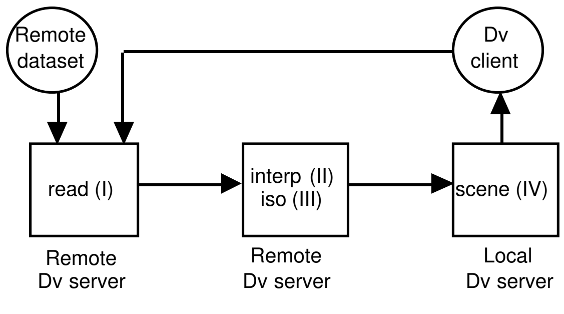

Figure 6: Example Dv Quake simulation.

5.1 Scheduling Dv applications

An interesting aspect of the Dv design is that it provides a flexible framework for investigating and experimenting with different scheduling policies. By making scheduling decisions at different points in a visualization session, we can create applications with different degrees of resource awareness and adaptivity. Here are a number of possible scenarios:

Scheduling at request frame creation time. In this scenario, when the client issues a request frame, a single schedule for all of the response frames that result from the request is constructed. Each of these response frames visits the same sequence of hosts, and each host performs the identical computation on each frame. The decision can be made by the client and passed along with the request frame, or the decision can be made by the server when it receives the request. The application can be endowed with some measure of performance-portability if the scheduler consults a resource monitoring system such as Remos [6] or NWS [11]. However, since all response frames adhere to the same schedule, the application will not be able to adapt to dynamic resource availability.

Scheduling at response frame creation time. In this scenario, the request server schedules each frame when it is created. The application can be scheduled so that it is performance-portable, and also so that it has some degree of adaptivity. However, since the schedule for any individual frame is fixed once the frame is in flight, the application can only adapt at frame boundaries.

Scheduling at response frame delivery time: In this scenario, a new scheduling decision is made each time a Dv active frame server computes a destination host for an output frame. This approach provides the greatest degree of adaptivity, but it may be overkill.

6 Summary and conclusions

We have described a framework called *Dv* for building interactive distributed visualizations of massive remote scientific datasets. The framework is based on the notion of an active frame, which is a form of mobile object that contains both application data and a program that manipulates the data. The idea is quite general and could prove useful for applications besides distributed visualization. Besides having a fairly simple design that could allow it to be deployed on a wide scale, Dv provides a flexible framework for building distributed applications that are performance-portable and adaptive.

References

[1] BAO, H., BIELAK, J., GHATTAS, O., KALLIVOKAS, L., O'HALLARON, D., SHEWCHUK, J., AND XU, J. Large-scale simulation of elastic wave propagation in heterogeneous media on parallel computers. *Computer Methods in Applied Mechanics and Engineering 152* (Jan. 1998), 85–102.

[2] CASANOVA, H., AND DONGARRA, J. Netsolve: A network server for solving computational science problems. Tech. Rep. CS-95-313, University of Tennessee, Nov. 1995.

[3] DEWITT, T., GROSS, T., LOWEKAMP, B., MILLER, N., STEENKISTE, P., SUBHLOK, J., AND SUTHERLAND, D. Remos: A resource monitoring system for network-aware applications. Tech. Rep. CMU-CS-97-194, School of Computer Science, Carnegie Mellon University, Dec. 1997.

[4] FOSTER, I., AND KESSELMAN, C. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications 11*, 2 (1997), 115–128.

[5] FOSTER, I., AND KESSELMAN, C., Eds. *The Grid: Blueprint for a New Computating Infrastructure*. Morgan Kaufman, 1999.

[6] LOWEKAMP, B., MILLER, N., SUTHERLAND, D., GROSS, T., STEENKISTE, P., AND SUBHLOK, J. A resource query interface for network-aware applications. In *Proc. 7th IEEE Symp. High-Performance Distr. Comp.* (July 1998).

[7] SCHROEDER, W., MARTIN, K., AND LORENSEN, B., Eds. *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*, second ed. Prentice Hall PTR, Upper Saddle River, NJ, 1998. www.kitware.com.

[8] TENNENHOUSE, D., AND WETHERALL, D. Towards an active network architecture. *Computer Communication Review 26*, 2 (August 1995), 5–18.

[9] UPSON, C., FAULHABER, T., KAMINS, D., ET AL. The application visualization system: A computational environment for scientific visualization. *IEEE Computer Graphics and Applications 9*, 4 (July 1989), 30–42.

[10] VON EICKEN, T., CULLER, D., GOLDSTEIN, S., AND SCHAUSER, K. Active messages: a mechanism for integrated communication and computation. In *Proc. 19th Intl. Conf. on Computer Architecture* (May 1992), pp. 256–266.

[11] WOLSKI, R. Forecasting network performance to support dynamic scheduling using the network weather service. In *Proceedings of the 6th High-Performance Distributed Computing Conference (HPDC97)* (Aug. 1997), pp. 316–325. extended version available as UCSD Technical Report TR-CS96-494.