

SmartPointers: Personalized Scientific Data Portals In Your Hand

Matthew Wolf
Zhongtang Cai
Weiyun Huang
Karsten Schwan
{mwolf,ztcai,wyhuang,schwan}@cc.gatech.edu
College of Computing
Georgia Institute of Technology

Abstract

The SmartPointer system provides a paradigm for utilizing multiple light-weight client endpoints in a real-time scientific visualization infrastructure. Together, the client and server infrastructure form a new type of data portal for scientific computing. The clients can be used to personalize data for the needs of the individual scientist. This personalization of a shared dataset is designed to allow multiple scientists, each with their laptops or iPacs to explore the dataset from different angles and with different personalized filters. As an example, iPac clients can display 2D derived data functions which can be used to dynamically update and annotate the shared data space, which might be visualized separately on a large immersive display such as a CAVE. Measurements are presented for such a system, built upon the ECho middleware system developed at Georgia Tech.

Introduction

High speed networks and grid software [Globus, AG] have created new opportunities for scientific collaboration, as evidenced by past work on remote instrumentation [OAC98], remote visualization [MaC00,SSL00], the use of immersive systems across the network [FoG97,CSD93], and by new programs like the Terascale Supernova Initiative at the Department of Energy [TSI]. In all such applications, scientists and engineers working in geographically different locations collaborate, sometimes in real-time, by sharing the results of their large-scale simulations, jointly inspecting the data being generated and visualized, running additional analyses, and sometimes even directly running simulations through computational steering [SLM00] or by control of remote instruments [PTC98].

A common paradigm used by the community is that of a workbench or portal via which end users interact with such grid applications and services. Portals facilitate the use of distributed services and systems by providing application-specific means of interacting with both. Portal architectures [GaBF, KBG01] provide the basic set of abstractions and components based on which application-specific portals may be built.

This paper explores a common problem with data portals, which is the heterogeneous nature of the machines and platforms on which they must operate. Specifically, we ask how end users may meaningfully interact across highly heterogeneous machines and network connections. This issue has arisen, for example, for the Terascale Supernova Initiative in which end users from the national labs operating across Gigabit links must interact in real-time with collaborators on PCs operating across standard Internet links. Taken to an extreme, we ask how end users operating with very large data sets displayed only on high end systems like Immersadesks driven by large SMP machines can usefully interact via low end engines like laptops and PDAs.

The specific interaction paradigm explored in this paper is one in which a low end device essentially acts as a 'smart pointer' into the large data space existing in the distributed system and/or displayed on devices like a CAVE or Immersadesk. That is, while a CAVE may be used to render to an end user the entire data space (or large portions thereof), the handheld device cannot hope to render any meaningful subset of this data in the same fashion. Instead, its role is to provide alternative views of specific data elements, to activate analyses meaningful for these elements and display analysis results, and to track and interact with collaborators. When located in the same room as the immersive device, the smart pointer may present certain details or complementary information about the data displayed in its entirety. When operating in a distributed system, a smart pointer may be viewed as presenting similar information about the large, distributed, and shared data space that defines end users' distributed collaboration. In both cases, a smart pointer permits an end user to interact with the large, shared data space as per his/her current interests and needs, where most such interactions entail the activation of services that transform, analyze, filter, and sample shared data.

The remainder of this paper describes a specific implementation of the smart pointer paradigm, using an iPaq handheld PDA communicating via a wireless link with the servers that provide 'smart pointer' services and that can efficiently access or 'tap into' the large data space shared by real-time collaborators. The data used is output from a standard parallel molecular dynamics simulation used by physicists and mechanical engineers to explore a number of atomistic phenomena, from melting to crack propagation to friction. In the experiments described in this paper, such output data is continuously streamed from the simulation to the server systems and to a high end display device, an Immersadesk[CPS97]. The smart pointer device taps into the same data stream by interacting with the same server systems.

While experimental results are focused on our specific experimental setup, the smart pointer system and architecture presented in this paper can extend its operation across wide area systems. The ideal is to support distributed collaboration and steering of computational science, with a strong emphasis on providing the personalized data portals that the smart pointers represent to every collaborator. The collaborative environment is provided already by the AccessGrid toolkit[AG]. By adding the ECho event channel infrastructure, the middleware which underlies the current implementation, as a parallel data transport to the Access Grid, we can utilize ECho's high performance communications infrastructure for heterogeneous binary transport of the scientific data. In addition, ECho's facilities for runtime, source-based filtering of data help to optimize performance of the clients as well as enhancing their customizability.

The SmartPointer Application

For this research we have focused on the molecular dynamics (MD) application area, since it is of interest to computational scientists in a wide variety of fields, from pure science (physics and chemistry) to applied engineering (mechanical and aerospace engineering). Molecular dynamics codes aim to replicate atomic-scale motions, using various representations of the forces involved. These codes are used to study some very large problems, sometimes involving hundreds of thousands to millions of atoms. The run-time of such problems can be very long, even on massively parallel machines, and as such the task of visualizing and steering of the codes can be very important. Traditional methods of dealing with such data flows involve complete state logging for later viewing (which does not scale well to large sizes or long runs) or the storage of partially interpreted data such as auto-correlation functions or time averages (which may fail to preserve data needed in subsequent interpretation).

The SmartPointer system has been designed to address these concerns by providing a flexible and customizable user interface which can support multiple levels of user interaction both in real time and in post-production (through the use of logging clients). In this way, a scientist can “jump into” a running simulation, check on statistics of interest, perhaps tweak or tune parameters, or even decide to terminate a run. Then, when he or she has finished, the visualization software can cleanly detach from the system.

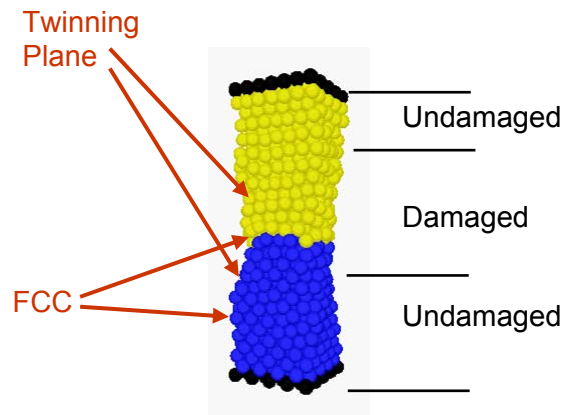


Figure 1. An example of molecular dynamics data that might benefit from user-specific viewpoints. For this single simulation of a block of copper being stretched, on the left we see the sorts of things that a physicist might want to highlight, while the right side shows the higher-level synthesis that a mechanical engineer might want.

More than just the single scientist viewpoint, though, the smart pointers are intended to be a facilitator for multi-disciplinary interactions. Each investigator can interact with the same data stream through a visualization mechanism of his or her choosing. For example, as seen in Figure 1, a physicist might choose to examine a deforming crystal to look for regions of reformation of FCC crystals inside the deformation zone. On the other hand, a mechanical engineer might be more interested in statistics which might be

able to relate damaged to undamaged sections of the crystal, which would have more utility in describing mechanical strengths of the material. Since each individual can interact with the data stream simultaneously but independently, additional realms of interdisciplinary cooperation can be enabled.

This is possible due to the ECho event channel middleware, which will be described in a later section. The instrumentation of the MD code is done through replacement of the subroutines which would normally log the restart files. Since most MD codes have such facilities, and certainly all of the large-scale codes already do, this is a clean and very portable way to extract the desired information. It also minimizes the degree of knowledge of the ECho middleware which the application scientist needs, which is a considerable help in pursuing its adoption by multiple communities.

The SmartPointer System

The following is a discussion of the data flow within our distributed visualization environment. Atomic data originates with a parallel molecular dynamics computation. Before it can be utilized by the end visualization clients, this data needs to be parsed and manipulated in several client-specific ways.

Scientific Data Flow

In our distributed visualization environment, atomic data (coordinates, atom types) originates with a parallel MD computation. This data is then streamed to a server which does some core filtering and extension of the data, namely the calculation of the radial distribution function of the data (to be described shortly) and the determination of bonded atom pairs. From that server the extended data is distributed to clients and data staging servers. Through ECho's filtering capabilities, users could further specify different analysis of the data and observe the process with different types of clients from different locations.

The data flow within the SmartPointer system is characterized by several different types of server nodes and of clients. The overall flow is shown in Figure 2, with each of the key components described in the following text.

Bond server. The bond server receives the coordinates and types of atoms from the MD code, computes the bonds between the atoms according to the distance thresholds (which could be changed by a radial distribution client), and broadcasts the information from two channels. One channel, the main "data" channel, contains the previous data and adds the bond lists. The bond server calculates the bond list by first computing all the pairwise distances between atoms (the same computation is also used for the radial distribution function). The user must specify some pair of parameters r_1 and r_2 . If the distance between atom i and j is within the interval $[r_1, r_2]$, then there is a bond between atom i and atom j , and it is copied into the bond list data structure.

The other channel exported by the bond server contains the distribution of the distances between the atoms and the current thresholds used to compute the bonds. The calculation

of these quantities is described in more detail in the next section. Feedback on this channel is used to modify the parameters r_1 and r_2 used for the data channel.

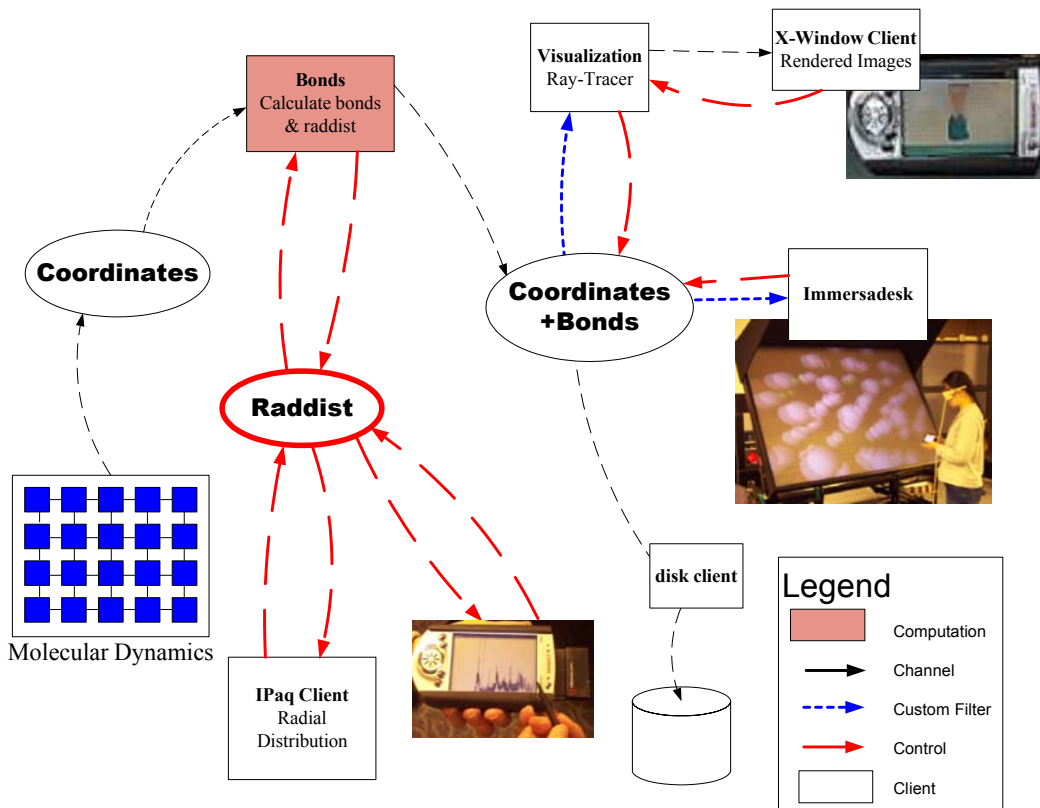


Figure 2. A logical diagram of the current implementation of the SmartPointer infrastructure.

Radial distribution client. The radial distribution function is a histogram of all of the pairwise distances between atoms at a given time within the simulation. The radial distribution is recorded in a one-dimensional integer array of length $binmax$, where each element of the array corresponding to the number of the distances falling in a specific range. For example, given a distance d and a maximum distance of interest $rmax$, the following formula would apply:

$$\text{Radial_dist}[\min(binmax, \text{int}(binmax*d/rmax))] += 1 \quad (1)$$

Thus the size of the data sent to the client is of a set and clearly controlled size, which can be tuned by the client through setting the display parameters $binmax$ and/or $rmax$. The iPacs that run the client application have wireless connections, so the data sent to them must be carefully sized to achieve real-time interactions, as we shall see later.

In addition to modifying the display parameters, the iPaq client can set the thresholds used by the bond server to calculate the bonds to be added into the data channel (labeled “Coordinates + Bonds” in Figure 2). Viewing the radial distribution function allows the end user to see where the natural cut off points are for the data, making the choice of bonding distances faster and more accurate. It also allows the user to dynamically explore particular subsets of bonds. For instance, by changing r_1 and r_2 on the client, the

user could select only the nearest neighbors that have bonding lengths shorter than the equilibrium (or “normal”) bonding distance, and the visualizations in the shared data space would consequently be updated.

OpenGL/Immersive Display. As an example of a high-end display, we have an ImmersaDesk as a subscriber of the bond server’s data channel. This type of immersive display client provides a virtual three-dimensional display of complex structures and lets the users navigate within them. Since the bond server and these clients are connected by a high bandwidth network, and since the Immersadesk is connected to a large SMP with substantial graphics hardware, the bond server can send large amounts of raw data to the client without additional parsing of the data.

Visualization Server and 2-D clients. The visualization server, which uses a parallel ray-tracing algorithm, serves both as a subscriber of the bond server’s data channel and a source for two-dimensional display clients. The server utilizes a persistent master-worker parallelization model, and the master receives the atomic information from the bond server’s data channel and generates the necessary internal description of the data. The worker processes then converts the description into a ppm image, which is submitted onto a new channel. This image is sent to a bandwidth- or cpu-limited client, such as an iPaq on a wireless link or a remote workstation client.

The visualization server also receives control information from the display client. The user can change the ray-tracer’s parameters, such as the camera’s location, the camera’s orientation, etc, by pressing the iPaq’s buttons and joystick. The iPaq then sends the user’s request for a new viewpoint to the visualization server, and the ray-tracer uses this new viewpoint when it generates the scene description.

Other clients in the system. So as to preserve the output of the logging subroutines we instrumented to gain access to the MD code, a disk client can receive atoms and bonds information from the bond server and store it to as disk files.

SmartPointer Enhancements.

In addition to the basic visualization system already described, there are a number of enhanced features which enable even greater flexibility for the scientific end-user’s personal data portal, which depend in part upon the particular features of the middleware infrastructure we use. Some of these have been partially described above

- Parallel stream based filtering for the visualization server and the bond server. By establishing a persistent parallel stream filtering process the processing time should be decreased as well as improving scalability for larger numbers of clients. The publish-subscribe infrastructure simplifies the parallelization process, since each process can independently subscribe to the same channel and receive the incoming data.
- Client-specified filtering. The middleware infrastructure makes it relatively easy for us to add client-specific filters, such as viewer-specific cuts through large data sets (only atoms with $x > 0.0$) or filtering by atom type (type == “Oxygen”). This type of

filtering is not only scientifically valuable, it also provides a further savings in bandwidth for remote or limited end devices.

- Additional scientific annotation tools. The bond server generates scientific annotation of the data (in this case, neighbor lists based on inputs to the radial distribution client), which serves as a starting point for other scientific annotation algorithms such as common neighbor analysis. These tools can subscribe to the existing data channel and create their own modified versions, which the existing end clients can then use.

Publish-Subscribe Middleware Support

For this application, we require a communications infrastructure which can be flexible, adaptive, and yet support high performance. Traditional HPC-style communications systems like MPI offer the required high performance, but rely on the assumption that communicating parties have a priori agreements on membership lists and on the basic contents of the messages being exchanged. For the sort of system we have described, however, both data types and subscription lists of communicating parties must be flexible. This need for flexibility has led some designers to adopt techniques like Java's RMI or meta-data representations such as XML+SOAP. These methods have high costs which interfere with total performance, because data marshalling becomes a key issue.

The ECho middleware[ECho,EBS00] addresses these concerns in several different ways. In particular, it provides the following:

- Information flows are represented as **event streams**, using the publish/subscribe model.
- There is no centralization; channels are represented by distributed data structures.
- Connections are managed so as to preserve transparency of local versus remote receivers, with implementations for underlying peer-to-peer communications over TCP, UDP, Wireless, multicast, and others.
- Efficient, fully typed binary data transmission based on dynamically defined event formats (PBIO).
- Dynamic extension of existing formats and discovery/operation on format contents is enabled through run-time dynamic code generation of subscriber-specified filters.
- Interoperation with CORBA and Java (JECho) via IDL and XML is enabled through conversion of data at the far endpoint.

Further information is available in the references cited in [ECho].

Experimental Setup

The interconnection of machines closely matches the logical diagram drawn in Figure 2. For the measurements described below, the following hardware was used for each of the pieces:

- The originating MD code runs on an SGI Origin 2000. Performance of this code is not measured directly.
- The parallel bond server and the visualization server both run on a Dell 8450 8-way Pentium III Xeon box with 4 GB memory.
- The Immersadesk is connected to a 16 processor SGI Origin 2000 with an InfiteReality3 graphics engine.
- Finally, for the end-point wireless clients, we are using Compaq iPaq's with 64MB memory, running the Linux Intimate distribution.

The size of each ppm image sent from the visualization server to the iPaq client is 182428 bytes, and the control message sent from the iPaq to the ray-tracer contains 28 bytes of actual data. An additional note to observe is the high level of heterogeneity of the platforms. The support for heterogeneity is transparently provided by the ECho middleware infrastructure.

Results

iPaq visualization performance. In this experiment, the round-trip time from the client iPaq to the visualization server is measured for data sets between the sizes of 10 and 1000. In each case, the iPaq sends out a request for a change of viewpoint. The server generates a new frame for iPaq and sends it back. Since the number of atoms doesn't affect the size of the image, the changes in response time with respect to number of atoms are mainly because of the server's computation time. Although this yields a reasonable visualization update time, further optimization of the ray-tracing algorithm to better reflect the computational environment should cause substantial improvements.

We compared the performance of our solution to an OpenGL program running natively on the iPaq using the Mesa library. The iPaq doesn't have a floating point coprocessor, therefore it takes the OpenGL program an average 3.996 seconds to display 24 atoms. The iPaq OpenGL visualization client is a slightly modified version of the OpenGL client which runs on the Immersadesk, whose performance is detailed in Figure 4. The ray-tracing visualization server does not provide the same level of interactive visualization as the Immersadesk, but its refresh rate is fast enough to make it usable, and it is clearly far more scalable than the native OpenGL solution on the iPaq.

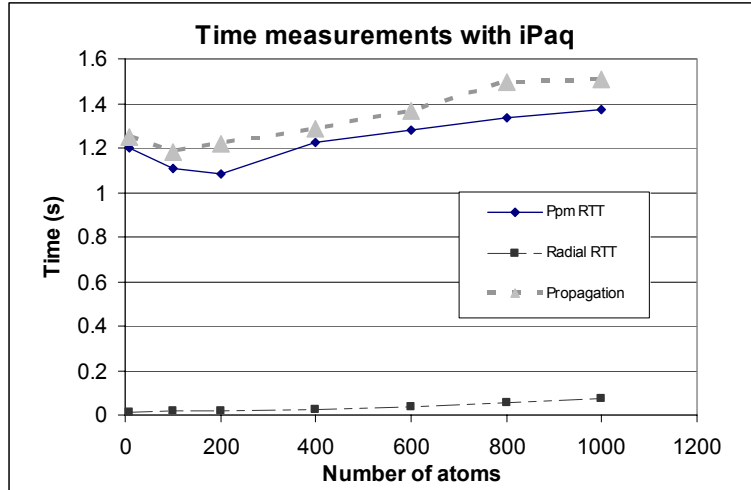


Figure 3: Response time for iPaq visualization. The propagation line is the time it takes to propagate a new frame from the server out to the client. The two return trip times (RTT) are response times for re-rendering the data as a result of user input (a zoom operation). For comparison, an OpenGL client running natively on the iPaq using the Mesa library took 3.996 seconds to re-render a 24 atom scene. The propagation time measures the unidirectional time from the bond server to a rendered image on the iPaq. It is greater than the PPM RTT time due to the added communications and the set-up costs for a new visualization data frame.

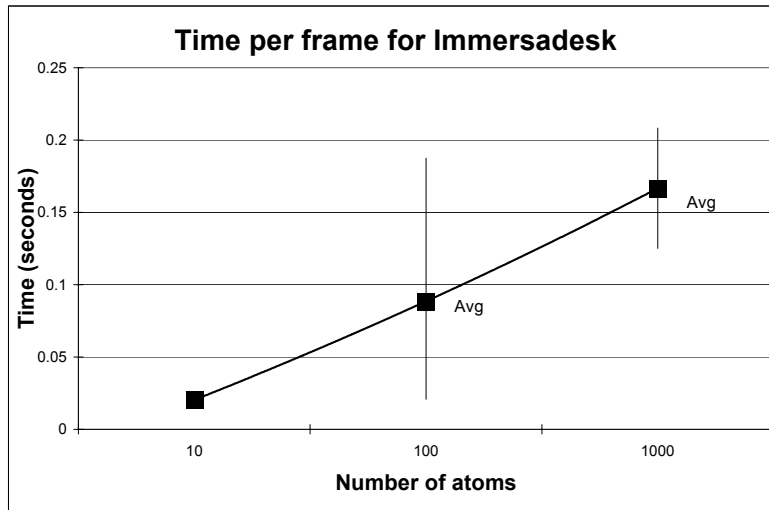


Figure 4: Time per frame for Immersadesk. The time for the Immersadesk to render one frame of data, when number of atoms varies from 10 to 1000. Time spread for each number of atoms due primarily to the variable number of bonds in the frame.

Radial distribution performance. In this experiment we measure the response time for the radial distribution client. The bottom trace in Figure 3 shows the round-trip-time (RTT) from a user-requested modification on the radial distribution client (decrease r_{max} to “zoom in”) till the modified data is received. Note that the RTT for the radial distribution data feed is very close to that of the Immersadesk OpenGL client (as seen in Figure 4),

requiring an average 74.0 ms for a user-requested update at 1,000 atoms, as compared to 166. ms for the Immersadesk client. These refresh times are sufficiently close to real time to provide a very usable interface for data monitoring, and the small relative difference between the two update times makes the SmartPointer control loop feasible.

The amount of data being transmitted from the bond server to the iPaq remains constant as the number of atoms changes. The bond server needs to compute pairwise distances of atoms to get the bond information, and we can expect that it would overwhelm the iPaq if we computed the bonds locally, not to mention the lack of scalability in moving that amount across the wireless link.

Performance of SmartPointer Enhancements

Although the system described so far works quite effectively for small number of clients, as the number of simultaneous users increases, the bond server may become overloaded. The parallelization of the bond server is one measure we have taken to address this problem. Since the calculation of neighbor lists and radial distribution functions requires a relatively tightly coupled data interaction, we have chosen to pursue a shared memory parallelization strategy initially.

OpenMP directives are utilized within the code, since most of the core of the code is composed of the pairwise comparison loops. The thread pool is initialized only once, with the non-master threads blocking until the next atomic update event arrives. Figure 5 shows the measurements of scalability for numbers of clients and numbers of concurrent threads within the bond server.

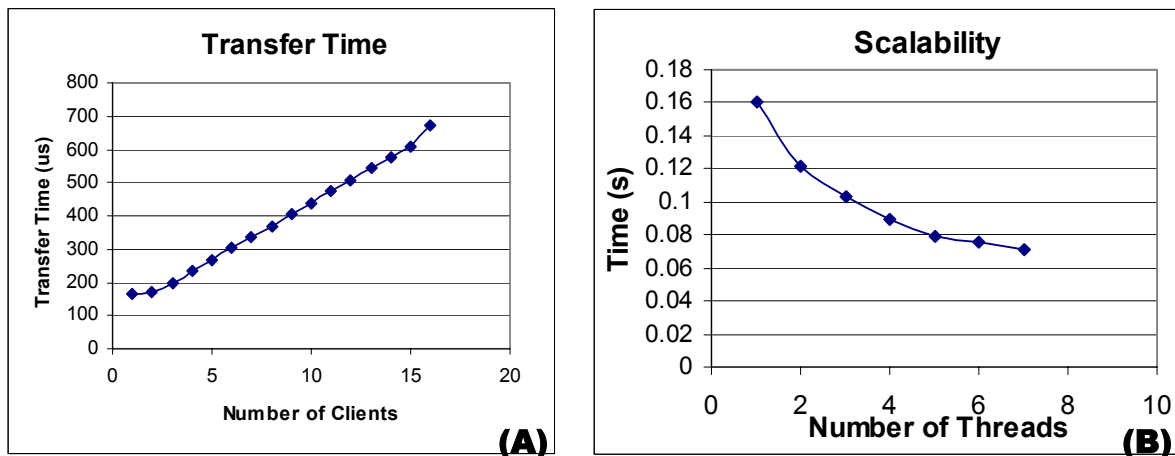


Figure 5. Response time as a function of the number of clients. (A) shows the amount of time, in microseconds, for a data frame to be transferred from a single-threaded server to multiple clients. (B) shows the decrease in time to compute the pairwise distances within the bond server as a function of the number of threads. Since these two are independent contributions to the total run-time of the bond server, the multi-threading can be used to offset the increase in transfer times.

Another key advanced feature is the ability to provide custom, client-specified cut-throughs of data. The ability to section data is both scientifically and practically useful. Particularly for large atomic systems, many meaningful structures may be hidden beneath the surface. Client-specified cutting planes, determined in real time by the end user, can be deployed back to the server. This provides an immediate savings in bandwidth, as the

client no longer receives unwanted data, as well as exposing the personalized data viewpoint for collaborators. Rather than inputting the parameters of the filter into their own client, the collaborators can subscribe to the same feed with the filter already applied. This again enables the handheld (or indeed any client) to drive the visualization of others within the shared data space.

```

Sample Scientific Filter

{ for (nIn=0, nOut=0; nIn<input.natoms;) {
  if (input.xatoms[nIn*3]>5.0)
  { output.oitype[nOut]=input.itype[nIn];
    output.oatoms[nOut*3]=input.xatoms[nIn*3];
    output.oatoms[nOut*3+1]=input.xatoms[nIn*3+1];
    output.oatoms[nOut*3+2]=input.xatoms[nIn*3+2];
    nOut=nOut+1;}
  nIn=nIn+1; }
  output.onatoms=nOut;
}
```

Figure 6. Example of filter code. This simple code takes the input data structure (input.*) and copies it out to a corresponding output data structure (output.*) only if the atoms to be visualized lie to one side of the cut off plane $x = 5.0$. This example can be easily generalized to a generic plane ($ax + by + cz \leq d$) or to an atomic species filter (`itype == "oxygen"`).

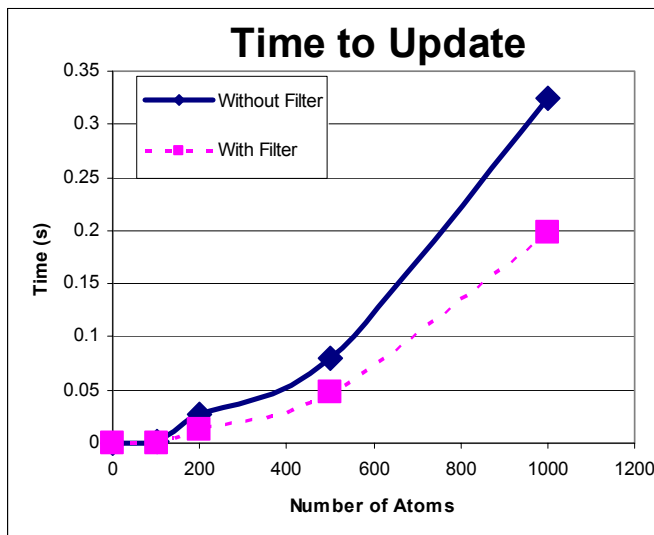


Figure 7. Time for the transfer of filtered and unfiltered data sets between server and iPaq. A data filter similar to that shown in Figure 6, but selecting only appropriate bonds, is applied to the raw data stream before visualization. The reduction in total transmitted data improves both visualization and transmission times.

The filters are implemented in ECL, which is a restricted subset of C[Eis02]. The familiarity of the C-link interface makes scientific filters relatively easy to create and deploy. The ECho middleware exports the ECL filter from the client as text and then utilizes dynamic code generation on the server. The cost of filtering the data on the

server is relatively minor compared to the savings in bandwidth and endpoint cpu cycles, particularly for visualization and down-sampling transformations.

Related Work

Much of the recent scientific portal work has focused on advanced tools and databases that could be shared by a community of users via web technology[KBG01]. These web-based Scientific Portals provide mechanisms to access and manage remote resources needed for large scientific codes. They have explicit interfaces to grid software packages and other distributed run-time tools[GaBF]. However, these portals do not directly address the needs of a dynamic visualization environment, where large volumes of data may need to be sifted and analyzed through highly customized filters.

Another related area of research is the studies of distributed collaboration in shared environments like in the 3D virtual environment of the CAVE[SSL00,LJD99]. Immersive distributed collaboration is required to meet the vast need of cutting-edge computational science (Chemistry, Astrophysics) and engineering (Aerospace, Automotive) for distributed access to shared data and computations[RGC97].

Such distributed collaboration needs not only the virtual reality environment and appropriate tools for data exploration, but also new communication middleware and architecture support for both asynchronous and real-time synchronous data exchanging. I-WAY[DeF96,FoG97] is an experimental environment for building distributed virtual reality applications and for exploring issues of distributed wide area resource management and scheduling. I-WAY has five application types that showcase how supercomputers and CAVE environments interact with each other. The personalization of the data streams and the inherent heterogeneity of the SmartPointer clients are distinguishing features, although there certainly are interesting possibilities for interoperation.

The Cumulvs project[KoP98,GKP97] is an attempt at a middleware infrastructure for dynamic attachment and detachment of visualization processes. Much of its development has focused on dynamic fault tolerance and heterogeneous task migration, which is a complementary toolset to the work we have presented here. These tasks are all carried out through user-space library calls, so it has the potential to be used side-by-side with the existing SmartPointer processes. Future work is intended to explore this approach more fully.

Conclusions

The SmartPointer system introduced in this paper provides a unique paradigm for the creation and use of lightweight visualization clients which operate cooperatively in the context of an online scientific visualization infrastructure. The SmartPointer infrastructure forms a new type of data portal for scientific computing, providing the scientific end user with the ability to dynamically generate customized viewpoints that can be used to modify, annotate, and control a larger shared data space. Achieving this

level of customization in a portal environment requires a communication infrastructure that not only can efficiently transfer large amounts of data, but can also supports plug-and-play data flow subscription and smart filtering. The ECho event system fills this role in the SmartPointer infrastructure, and we presented measurements which both support the viability of the SmartPointer concept and show how ECho features are exploited to improve the scalability and performance of the system. In particular, we showed that the use of dynamically code generated filter functions in ECho's derived event channels can nearly halve the SmartPointer display update time by specializing the client's data flows to more closely match its needs.

Our future work will focus on creating SmartPointer displays and tools for other applications and on enhancing the collaborative aspects of the system. As we explore new applications and displays, we expect to encounter situations which require more elaborate filtering and data transformation than the relatively simple filters that we have deployed so far, requiring heavier use of ECho's facilities to relocate computation. For example, global climate models often represent atmospheric data as a set of orthogonal spherical basis functions. In this environment, data display requires a computationally expensive filtering (conversion from spectral to grid space), that would be prohibitively expensive for light-weight clients and potentially burdensome for the server. In such cases, ECho's ability to relocate computation to third parties may be required to provide reasonable visualizations. We also expect to exploit ECho's ability to tag data with additional meta-data, such as on-demand XML markup information, in order to allow interoperation with many additional visualization tools while maintaining the customizations enabled by ECho. These extensions will extend the SmartPointer infrastructure to new ranges of applications and collaboration environments.

REFERENCES

- [AG] The Access Grid is a project within the Global Grid Forum to support high performance collaborative infrastructures over the Grid.
<http://www.accessgrid.org>
- [All96] A. D. Alexandrov, M. Ibel, K. E. Schauer, and C. J. Scherman. SuperWeb: Research Issues in Java-Based Global Computing. In Proceedings of the Workshop on Java for High performance Scientific and Engineering Computing Simulation and Modelling. Syracuse University, New York, 1996.
- [BaK96] A. Baratlo, M. Karaul, Z. Kedem, P. Wychoff. Charlotte: Metacomputing on the Web. In In Proceedings of the 9th Conference on Parallel and Distributed Computing Systems, 1996
- [BBB96] J. E. Baldeschwideler, R. D. Blumofe, E. A. Brewer. ATLAS: An Infrastructure for Global Computing. In Proceedings of the Seventh ACM SIGOPS European Workshop on System Support for Worldwide Applications, 1996.",
- [ChR96] K. M. Chandy, A. Rifkin, P. A.G. Sivilotti et al. "A World-Wide Distributed System Using java and the Internet". Presented at High Performance Distributed Computing (HPDC-5) - Focus Workshop on Multimedia and Collaborative Environments, Syracuse, August 1996. Also available as Caltech CS Technical Report CS-TR-96-08 and CRPC Technical Report Caltech-CRPC-96-1
- [CSD93] C. Cruz-Neira, D. Sandin, and T. Defanti. "Surround-screen projection-based virtual reality: The design and implementation of the CAVE". In Proceedings of Siggraph '93 Computer Graphics Conference. 1993, pp. 135-142.
- [CPS97] M. Czernuszenko, D. Pape, D. Sandlin, T. DeFanti, G. Dave, and M. Brown. "The ImmersaDesk and Infinity Wall Projection-Based Virtual Reality Displays". Computer Graphics, vol. 31, no. 2, May 1997. pg. 46-49
- [DeF96] T. A. T. DeFanti, I. Foster, M. Papka, R. Stevens, and T. Kuhfuss. "Overview of the I-WAY: Wide Area Visual Supercomputing". International Journal of Supercomputer Applications, vol. 10, no. 2, 1996. pg. 123-130
- [DiP95] T. L. Disz, M. E. Papka, M. Pellegrino, and R. Stevens. "Sharing Visualization Experiences Among Remote Virtual Environments". In Proceedings of the International Workshop on High Performance Computing for Computer Graphics and Visualization, pages 135--142. Springer-Verlag, 1995.
- [EBS00] G. Eisenhauer, F. Bustamente and K. Schwan. "Event Services for High Performance Computing". Proceedings of High Performance Distributed Computing (HPDC-2000).
- [ECho] G. Eisenhauer. "The ECho Event Delivery System". See the web site at <http://www.cc.gatech.edu/systems/projects/ECho/>.

- [Eis02] G. Eisenhauer, "Dynamic Code Generation with the E-Code Language". Technical report GIT-CC-02-42, College of Computing, Georgia Institute of Technology, July 2002. ftp://ftp.cc.gatech.edu/tech_reports/2002.
- [FoG97] I. Foster, J. Geisler, W. Nickless, W. Smith, S. Tuecke. "Software Infrastructure for the I-WAY High Performance Distributed Computing Experiment". Proc. 5th IEEE Symposium on High Performance Distributed Computing, pp. 562-571, 1997.
- [FoK97] I. Foster, C. Kesselman. "Globus: A Metacomputing Infrastructure Toolkit". Intl J. Supercomputer Applications, 11(2):115-128, 1997.
- [GaBF] D. Gannon, R. Bramley, G. Fox et al. "Programming the Grid: Distributed Software Components, P2P and Grid Web Service for Scientific Applications". see the web site at <http://www.extreme.indiana.edu/~gannon/ProgGrid/ProgGridsHTML.htm>
- [GKP97] G. A. Geist, J. A. Kohl, P. M. Papadopoulos, "CUMULVS: Providing Fault-Tolerance, Visualization and Steering of Parallel Applications," International Journal of High Performance Computing Applications, Volume 11, Number 3, August 1997, pp. 224-236.
- [Globus] Documentation on the Globus project, funded through the NSF and NCSA, can be found at <http://www.globus.org>
- [HeH99] M. Hemy, U. Hengartner, P. Steenkiste, and T. Gross. "MPEG System Streams in Best-Effort Networks". Proc. Packet Video'99, April 1999, New York (CD ROM Proc. only)
- [KBG01] S. Krishnan, R. Bramley, D. Gannon, et al. "The SCAT Science Portal". SC2001 November 2001.
- [KoP98] J. A. Kohl, P. M. Papadopoulos, "Efficient and Flexible Fault Tolerance and Migration of Scientific Simulations Using CUMULVS," 2nd SIGMETRICS Symposium on Parallel and Distributed Tools, Welches, OR, August 1998.
- [LJD99] J. Leigh, A. Johnson, T. DeFanti, S. Bailey, and R. Grossman. "A methodology for Supporting Collaborative Exploratory Analysis of Massive Data Sets in Tele-Immersive Environments." Proceedings of the 8th IEEE International Symposium on High Performance and Distributed Computing, Redondo Beach, CA. pg. 62-69
- [MaC00] K. Ma, D. M. Camp. "High performance visualization of time-varying volume data over a wide-area network status". Proceedings of the 2000 conference on Supercomputing
- [OAC98] G. M. Olson, D. E. Atkins, R. Clauer, T. A. Finholt, F. Jahanian, T. L. Kilean, A. Prakash, and T. Weymouth. "The Upper Atmosphere Research Collaboratory". Interactions, vol. 5, no. 3, 1998. Pages 48-55
- [PTC98] B. Parvin, J. Taylor, and G. Cong. "DeepView: A Collaborative Framework for Distributed Microscopy". IEEE Conference on High Performance Computing and Networking, November 1998

- [RGC97] D. A. Reed, R. C. Giles, and C. E. Catlett. "Distributed Data and Immersive Collaboration". Communications of the ACM, November 1997/Vol. 40. No. 11.
- [SLM00] J. E. Swan II, M. Lanzagorta, D. Maxwell, E. Kuo, J. Uhlmann, W. Anderson, H. Shyu, and W. Smith, "A Computational Steering System for Studying Microwave Interactions with Space-Borne Bodies", Proceedings IEEE Visualization 2000, October 8-13, Salt Lake City, Utah: IEEE Computer Society Press, 2000, pages 441-444.
- [SSL00] N. Sawant, C. Scharver, J. Leigh, , A. Johnson, G. Reinhart, E. Creel, S. Batchu, S. Bailey, R. Grossman. "The Tele-Immersive Data Explorer: A Distributed Architecture for Collaborative Interactive Visualization of Large Data-sets". Proceedings of the Fourth International Immersive Projection Technology Workshop 2000, Ames, IA,
- [TSI] The Terascale Supernova Initiative is a project funded under the SciDAC program. Details can be found at <http://www.phy.ornl.gov/tsi/>
- [ZhS00] D. Zhou, K. Schwan, G. Eisenhauer and Y. Chen. "JECho - Supporting Distributed High Performance Applications with Java Event Channels". Cluster2000, IEEE International Conference on Cluster Computing.