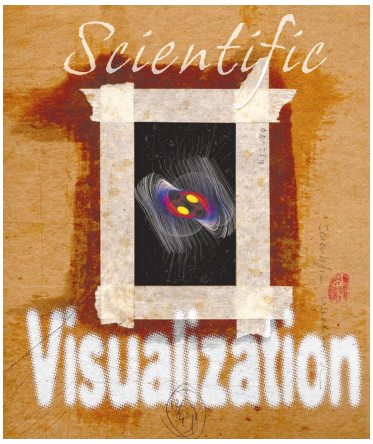
Virtue: Performance Visualization of Parallel and Distributed Applications



The Virtue prototype exploits human sensory capabilities to help performance analysts explore and optimize large-scale, multidisciplinary applications. The visualization environment lets collaborators interact with executing software, tuning its behavior to meet performance goals.

Eric Shaffer
Daniel A. Reed
Shannon Whitmore
Benjamin Schaeffer
University of Illinois at Urbana-Champaign

High-speed, wide-area networks have made it both possible and desirable to interconnect geographically distributed applications that control distributed collections of scientific data, remote scientific instruments, and high-performance computer systems. Such an application might, for example, control a remote radio telescope, transmit raw data from the telescope site to a distributed data archive, and concurrently convolve the data to create images for real-time visualization. Developing just such a distributed application infrastructure is the goal of our partners in the National Computational Science Alliance, one of the NSF Partnerships for an Advanced Computational Infrastructure.

Although interconnecting these applications enables geographically distributed science and engineering teams to collaborate in new ways, the resultant distributed computations pose significant performance analysis and optimization challenges. First, the execution environments of geographically distributed applications are far less deterministic than those of locally distributed, parallel applications.[1] Network bandwidths and latencies, computing resources, and available data repositories can vary from one execution to another, and even during a single execution. Consequently, identifying and correcting performance bottlenecks exposed during one execution may not benefit later executions.

Second, distributed applications are highly complex. Application components execute atop disparate system software and hardware. Real-time instruments often impose scheduling and access constraints. Accessing data repositories sometimes necessitates data translation for correlation with experimental or computational data. Finally, enabling effective remote interaction and visualization necessitates quality-of-service (QoS) guarantees. Incorporating QoS into these hardware and software systems further increases their complexity.

Historically, performance analysis has focused on monolithic applications executing on large, standalone, parallel systems. In such a domain, measurement, postmortem analysis, and code optimization suffice to eliminate performance bottlenecks and optimize applications. Most existing performance analysis systems—for example, SvPablo,[2] Medea,[3] and Paragraph[4]—use only postmortem analysis. To tune the emerging distributed applications, however, a new generation of online performance measurement and optimization tools must adapt application behavior dynamically as resource availability changes.

In addition to providing real-time adaptive control, new performance tools must gather data from multiple sources and software levels (application, library, system, and network). Furthermore, these tools must enable geographically dispersed teams to collaborate in identifying and correcting performance problems. This capability requires support of distributed visualization and control, as well as support of both synchronous and asynchronous collaboration.

0018-9162/99/$10.00 © 1999 IEEE

Distributed visualization, data mining, and analysis tools allow scientists to collaboratively analyze and understand complex phenomena. Likewise, real-time performance measurement and immersive performance display systems—that is, systems providing large stereoscopic displays of complex data—enable collaborating groups to interact with executing software, tuning its behavior to meet research and performance goals.

To meet these needs, we designed Virtue, a prototype system that integrates collaborative, immersive performance visualization with real-time performance measurement and adaptive control of applications on computational grids. The system combines the SvPablo instrumentation system, the Autopilot real-time adaptive-control toolkit, and the Virtue virtual environment for performance analysis. These tools enable physically distributed users to explore and steer the behavior of complex software in real time and to analyze and optimize distributed-application dynamics.

PERFORMANCE MEASUREMENT AND DISTRIBUTED CONTROL

To optimize distributed applications, performance measurement software must capture data from multiple sites, a variety of hardware and software platforms, and application software components in a variety of languages. Complementing measurement, distributed control mechanisms should allow users or control software to modify runtime parameters or resource allocation during the application's execution.

Performance measurement

As Figure 1 shows, Virtue relies on the Autopilot toolkit[2,5] for distributed measurement of executing software components. Autopilot is an extension of the Globus computational grid system,[6] which provides a shared address space across processes, systems, and networks. Globus also supports message-based distributed-application development and resource acquisition. Using Globus services, Autopilot defines a set of software sensors and actuators for instrumenting application code and controlling code behavior.

The Autopilot sensors consist of low-overhead routines designed to capture real-time performance data from distributed software components. User-defined functions can extend the sensors to process raw performance data before transmission—computing, for example, a profile from event-trace data. To minimize the complexity of sensor specification and configuration, Autopilot relies on the SvPablo system for sensor insertion in distributed-application source code. SvPablo parses the code and identifies all instrumentable regions—defined as the outer loops of loop nests—and all procedure calls. Users can instrument code interactively, via the SvPablo graphical user interface, or automatically, using parser command line options. In addition to generating instrumented source code, SvPablo generates static call graphs for later visualization.

To complement application data, we also created a modified Traceroute program that uses Autopilot sensors to report network latency and bandwidth data. All these data, ranging from SvPablo application instrumentation through network performance metrics, are transmitted to Virtue via the Autopilot sensors.

Distributed control

In addition to data capture, Autopilot supports adaptive control of remote applications and systems. Software actuators, inserted in application code or embedded in runtime libraries, provide mechanisms for controlling functions in software modules. At the application level, actuators can change variable values or choose among algorithms for a particular task. Similarly, runtime system actuators can change resource management policies and their parameters (such as file system prefetching or caching policies).

Together, the Autopilot sensors and actuators provide the performance data needed to visualize software structure and dynamics and the mechanism to modify those dynamics. Virtue imports sensor data for real-time, immersive visualization and provides direct manipulation tools for interacting with the Autopilot actuators. This interactive, closed loop allows users to modify software behavior while immersed in the virtual environment.

VISUALIZATION SYSTEM INFRASTRUCTURE

Although the scientific community has widely accepted immersive virtual environments for the analysis of complex scientific data, simple static and dynamic workstation graphics remain the de facto

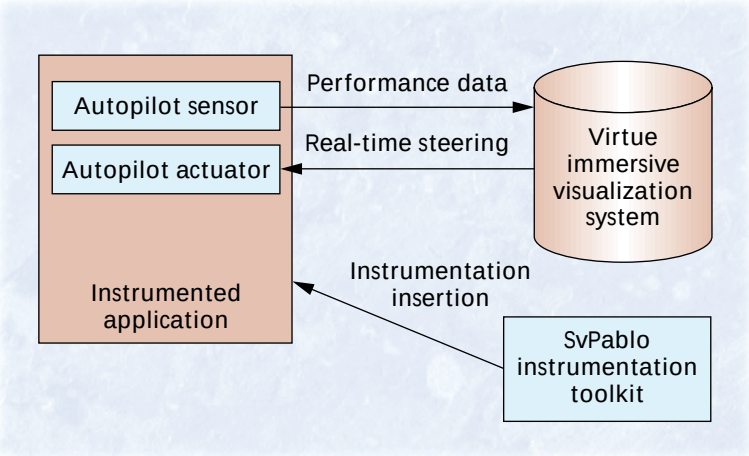


Figure 1. Distributed performance instrumentation and control using the Autopilot toolkit, an extension of the Globus computational grid system.

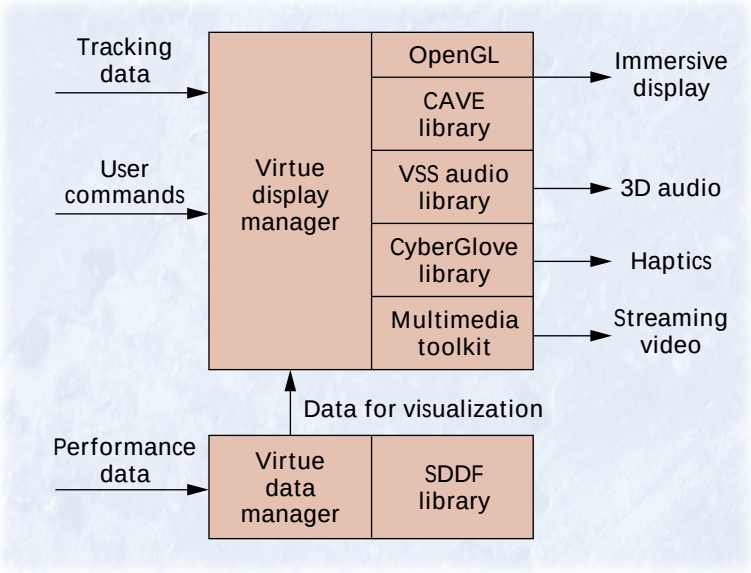


Figure 2. Virtue's software architecture. Key components include the SDDF data and control descriptions, the display manager, and the multimedia toolkit.

standard for performance analysis. Today's distributed parallel systems collectively contain thousands of processors connected by high-bandwidth networks, and they must access distributed secondary and tertiary storage systems. To effectively visualize and optimize the behavior of these systems, performance analysts need performance visualization systems that more fully exploit human sensory capabilities.

Obviously, the number of factors that can affect performance has increased. Moreover, effective performance tuning of distributed applications often requires cooperative exploration by a group of physically dispersed experts in domains such as networking, storage, scheduling, and architecture. Their analysis tools must reflect the collaborative, often asynchronous, nature of their work. To address these needs, we drew on insights from computer-supported cooperative work and virtual environments.

Software

Performance data visualization shares many features with information visualization—many data types and objects, such as software modules and processor utilizations, are abstract and thus lack default physical representation. Supported by appropriate mappings of data to representations, immersive displays can make the abstract concrete and provide intuitive interfaces that allow the same direct manipulation of objects that people perform in the physical world. With such an interface, for example, the user could increase the size of a file cache by "grabbing" and stretching a virtual box that represents a cache.

Clearly, finding intuitive mappings of abstract performance data to physical representations is critical; without appropriate mappings, displays are little more than animated light shows. As yet, we understand intuitive mappings only poorly, and it is likely that the best mappings are domain dependent. Our previous work on building performance environments[2,5,7] taught us that systems designed to visualize very specific data types have limited life spans and utility. Therefore, Virtue provides a language for quickly changing and exploring alternate data mappings.

At its core, Virtue is a general-purpose toolkit for visualizing common computer system and communication network components as hierarchical, three-dimensional graphs. It provides a rich graph description language for mapping data to graph attributes, including edge and vertex size, shape, color, translucency, position, and sound. The language includes powerful mechanisms for manipulating graphs and annotating graph components with audio and video notes. We based Virtue's graph description language on our extensible Self-Defining Data Format.[2,5] SDDF separates data structure from semantics, enabling users to quickly associate a new graph layout or data mapping by changing only a few lines of the SDDF description. Configuration files allow users to explore alternate mappings by quickly remapping variables to new visual or sonic attributes.

Because our SvPablo instrumentation system also uses SDDF as its data representation, integrating graph descriptions with performance-data mappings is straightforward. For example, by mapping procedure invocation counts and execution times to vertex size and color, users can render the call graphs created by SvPablo instrumentation as three-dimensional digraphs. In addition to using SDDF for performance-data representation, we also use it as Virtue's control language, to specify control mappings and virtual tools for data manipulation.

Finally, to support distributed collaboration, Virtue integrates videoconferencing for synchronous interaction, as well as capture and replay of multimedia annotations for asynchronous interaction. Modified versions of Multicast Backbone (Mbone) videoconferencing tools connect an annotation server and remote collaborators, exporting digital video of Virtue imagery and enabling collaboration. Moreover, Java tools for desktop and handheld computers allow desktop and mobile collaborators to control graph displays and examine the associated data.

Figure 2 shows Virtue's overall software architecture. The Virtue kernel consists of the data manager, which accepts SDDF data and control descriptions, and the display manager, which coordinates user interactions and renders the visualization. For collaboration and annotation, the multimedia toolkit coordinates multicast audio and video and user-tracking cameras. The National Center for Supercomputing Applications (NCSA) Vanilla Sound Server (VSS) supports sound spatialization, sonification, and generation of audio

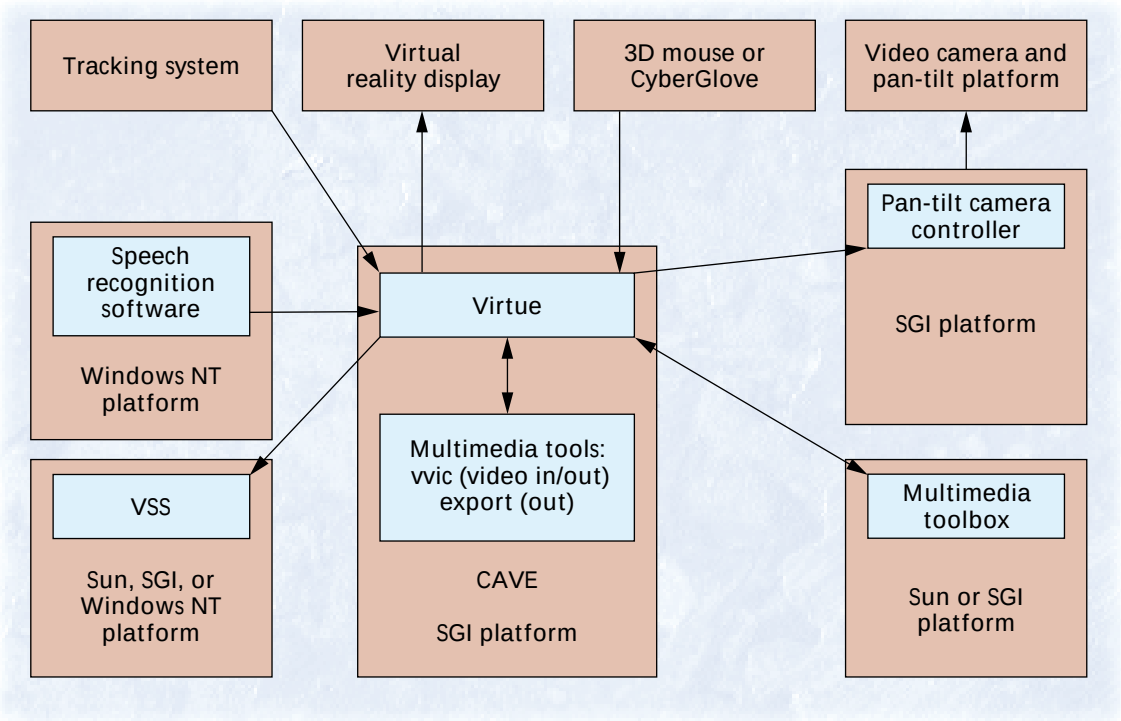


Figure 3. Virtue's hardware components, which center around an SGI platform.

cues. The VRCO CAVE library and OpenGL provide graphical support, and the CyberGlove library enables user input and haptic feedback via a Virtual Technologies CyberGlove. Finally, commercial speech recognition software enables voice commands.

Hardware

Itself a distributed system, Virtue consists of components that execute on a variety of PCs and rendering engines. As shown in Figure 3, the central component is an SGI platform. Currently, we use an Onyx2 with Infinite Reality graphics. This machine drives an immersive display device, such as CAVE or ImmersaDesk, for stereographic viewing. A tracking system, such as Ascension Technologies' Flock of Birds, complements the display device and reports user position and orientation to the tracking camera.

Users manipulate displayed objects with a tracked wand (3D mouse) or a Virtual Technologies CyberGlove, which provides data for hand visualization and tactile feedback from microvibrators. Users can issue commands with the wand or the data glove via a glyph-based drawing interface. Virtue includes a microphone headset to support voice commands.

To provide sound cues for events such as object intersection and to enable mapping of time-varying data to sonic attributes, Virtue provides spatialized audio, creating the illusion that sounds emanate from specified locations. The VSS uses position and orientation data to convolve the audio.

For synchronous collaboration, cameras on tracking platforms follow user movements and transmit the resulting video and associated audio to remote sites over the Mbone. Our multimedia tools manage incoming and outgoing Mbone video and audio streams, as well as the recording and replaying of these streams for asynchronous collaboration via annotations.

HIERARCHICAL GRAPHS AND VISUALIZATION METAPHORS

Each graph displayed by Virtue has an associated graph layout. These include

- explicit layouts, which use mapped data values to place vertices;
- ball-and-spring layouts, which use edge data values as spring tensions for vertex energy minimization;
- cone trees, which arrange the vertices in a 3D hierarchical structure; and
- application-specific layouts, which place graph vertices and edges according to a particular visualization, or display, metaphor.

Application-specific layouts

For visualizing distributed and parallel application performance, we have found layouts based on the following display metaphors particularly useful: geographic displays of network activity, time-tunnel displays of parallel process interactions, and call graphs of source code structure. These layouts reflect the hierarchy of distributed computations, ranging from wide-area communication through execution dynamics within single software modules.

Distributed applications depend on adequate network bandwidth and acceptable latency for interconnection of remote resources. In geographic network displays, the position of graph vertices represents the latitude and longitude of resource sites, vertex and edge attributes represent important link characteristics such as latency and bandwidth, and texture-mapped backgrounds provide context.

Virtue supports spherical backgrounds for global networks and planar backgrounds for local networks. For example, Figure 4 shows a geographic display of a wide-area network.

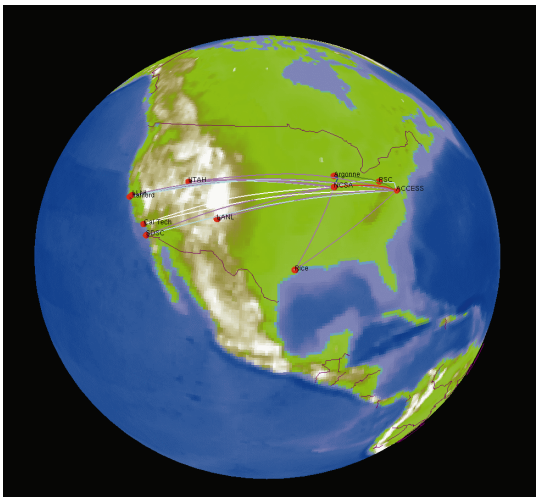


Figure 4. Wide-area geographic display. The link colors correspond to network latency.

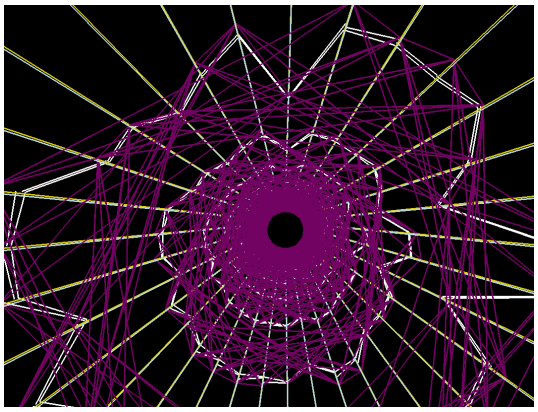


Figure 5. Time-tunnel display that shows the interactions among tasks and threads of a parallel computation. The white and purple chords show interprocessor communication; yellow time lines show task computation.

Figure 6. Call-graph display. The data mappings represent procedure metrics—for example, vertex color and size indicate invocation count and time spent by the procedure. The text labels indicate procedure names.

For understanding interactions among the tasks and threads of a parallel computation within a single machine, such as the locally parallel component of a geographic computation, we developed a system-level display known as a time tunnel.[7] The time tunnel consists of a cylindrical array of time lines, each line corresponding to the temporal sequence of behaviors in the associated task, with color indicating the activity type. Chords through the interior of the cylinder connect the time lines of interacting tasks. Figure 5 shows the interior of a time tunnel.

Finally, the procedure call graph is a familiar metaphor that pictures code blocks as vertices, with links indicating the static call hierarchy. As Figure 6 shows, Virtue uses a hierarchical layout of the vertices in three dimensions, with a variety of data mappings to represent procedure metrics. Finally, animation can show the evolutionary time pattern of invocation or metric variation.

Graph hierarchies

Virtue can also nest arbitrary graphs, each using a different display metaphor and layout. To nest graphs, users attach any graph to a vertex of any other, with each graph supporting separate mappings of data to graph attributes, including size, shape, and color. This nesting can extend to an arbitrary number of levels, and interactive controls can be used to expand and contract nested graphs.

An example of nesting is the creation of hierarchical views of geographic computations. Beginning with a geographic visualization of network traffic, the user selects a site and "drills down" to a time-tunnel view of the parallel computation at that site. Selecting a specific task yields a call graph for the executing code in that task. The user can expand multiple elements of the graph hierarchy at separate sites. In aggregate, this feature allows performance analysts to conduct "search-and-destroy" optimization, identifying problems and exposing more detail to correct their root causes.

INTERACTIVE DATA EXPLORATION AND ADAPTIVE CONTROL

Data visualization is only the first step to analysis, understanding, and control. With a large data volume such as that found in parallel and distributed applications, data visualization systems must also enable interactive pruning of unnecessary or inappropriate data. Moreover, these systems must facilitate comparison of alternate data representations to highlight critical data. Finally, systems must enable remote steering of the hardware and software on the basis of observed performance.

To aid data exploration and control, Virtue includes a set of virtual tools for manipulating and interrogating visual data representations. We based many of these, such as the generalized magnifying lens (Magic Lens) shown in Figure 7, on the research experiences and usability studies of other groups. When held over an object in query mode, the Magic Lens reveals hidden text (labels of geographic sites, procedure names in a call graph, or textual representations of numerical data). In data mode, it triggers an alternate map-

ping of data to visual attributes, providing a different view of the same data.

The second tool, a cutting plane, computes metrics on the data values associated with bisected graph edges. For example, with the cutting plane, the user can bisect a set of links in a network visualization and compute either the maximum latency or the bandwidth across the links. Figure 8 shows such an action.

Complementing Virtue's virtual tool set, a set of direct manipulation controls lets users change application or system behavior. Within the virtual environment, these controls take the form of three-dimensional sliders. By coupling the sliders with Autopilot actuators, Virtue translates slider manipulations to changes in remote system or application policy. Using these controls, an analyst can perform online steering of an application.

COLLABORATIVE VISUALIZATION

Large-scale, geographically distributed applications integrate a broad range of software components, developed in a variety of languages and executing on a diverse set of hardware and software platforms. By their nature, they execute at multiple sites with possible bottlenecks at many levels. Moreover, developers and performance analysts themselves frequently work at distributed locations. Hence, a team of scientists must often work with performance analysts across both space and time. To promote information sharing among these collaborators, Virtue includes multimedia annotation mechanisms for asynchronous collaboration and combines immersive visual display with desktop and handheld mobile systems.

Because Virtue graphs are tangible objects, users can annotate them in the same way they would annotate a physical object—by attaching a note. Simply by touching a Virtue object and issuing a voice command, users can attach an audio and video annotation of their insights about the displayed data. General observations about the visualization are suspended at the user-specified location in the display.

Although seemingly simpler than multimedia recording, textual annotations are poorly suited to virtual environment displays—their low resolution makes small windows of text illegible. Larger windows obscure the visualization, making it difficult to correlate the annotation with the display context. Moreover, text annotation requires either an obtrusive text input device or highly accurate speech-to-text translation.

To increase visual coherence, annotations that detail observations about a specific graph vertex appear as wireframe bounding boxes, as shown in Figure 6. The wireframe indicates the presence of annotations while minimizing visual obstruction. For general annotations not attached to a specific graph object, Virtue has more display flexibility. For example, we have used spheres with texture-mapped images applied to their surfaces to represent these annotations. The images can provide the author context in the form of a picture or a visual representation of the subject.

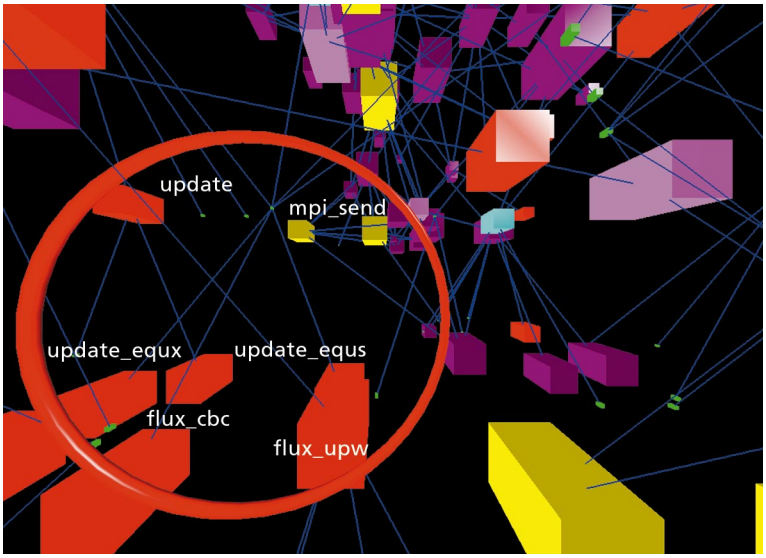


Figure 7. The Magic Lens tool, which operates in two modes. Held over an object in query mode, it reveals hidden text; in data mode, it triggers an alternate mapping of data to visual attributes.

We based Virtue's annotation system on modified versions of the Mbone VIC and VAT multicast videoconferencing tools. To record an annotation, a Virtue user issues a voice command and begins speaking. Tracking cameras center the speaker in their field of view, and the videoconferencing tools generate files that are stored in the annotation server's database. When a user opens an annotation, Virtue contacts this server, which then multicasts the digital audio and video files. Virtue then maps the received video to floating windows in the display.

Virtue uses the same multicasting software to transmit and receive live multicast audio and video streams. This enables videoconferencing between the immersive display and remote desktop and mobile devices. To further strengthen the coupling of immersive displays and remote systems, we also export Virtue's graphical displays via a separate multicast video stream.

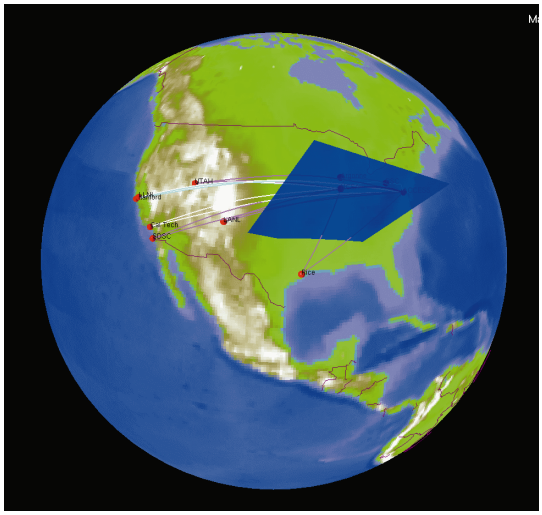


Figure 8. The cutting-plane tool. The user can bisect a set of network-visualization links and compute either the maximum latency or the bandwidth across the links.

Virtue includes a Java-based system that allows remote users to control aspects of the main display.

Although the desktop imagery has low resolution, it provides important context for collaboration.

Visualization and discussion are only part of the collaboration equation. To be equal partners, remote collaborators must also be able to interact with and control the visualization. Hence, Virtue includes a Java-based system that allows remote users to control aspects of the main display. It also allows them to locally display and control remote software behavior via Java interfaces to Autopilot sensors and actuators.

EXPERIENCES WITH VIRTUE

The true test of any performance tool is its usefulness in real applications. To evaluate Virtue's effectiveness, we instrumented a large, parallel application from the Department of Energy's Center for Simulation of Advanced Rockets, a part of the Academic Strategic Alliance Program. CSAR seeks to create and validate a fully three-dimensional, integrated simulation of the complex component interactions in solid rocket boosters. Recent well-publicized and expensive miscarriages of solid rocket boosters have generated great interest in understanding their potential instabilities and failure modes. The multidisciplinary problem is highly complex, and the associated simulation, written by a variety of cooperating researchers, is dynamic and continually evolving.

A parallel code written in Fortran 90, the CSAR application employs the Message-Passing Interface (MPI). Currently, the code models the fluid flow, combustion, and structure components of the space shuttle solid rocket booster (SRB); researchers continue to add new, more complex models and components. CSAR estimates that modeling the first 0.5 seconds of SRB burn will require 200 hours on a 128-processor SGI Origin 2000 supercomputer. The ultimate goal of modeling the entire two minutes of SRB burn depends on optimizing the code.

To analyze the CSAR application's performance, we created a real-time visualization detailing the code's execution on an SGI Origin 2000 at the NCSA. The visualization consists of three levels. At the highest level, shown in Figure 4, a geographic display captures the logical connectivity of the wide-area network joining several supercomputing sites and national laboratories in the continental US. Our modified Trace route program, augmented with Autopilot sensors, captures latency data, which Virtue maps to link color. Although the CSAR application currently executes on only one parallel system at a time, this geographic visualization allowed us to validate Virtue's real-time network traffic visualization capabilities.

At the midlevel of the display hierarchy, we represented processor activity and message-passing data from the single-platform, parallel execution. We instrumented the source code automatically via SvPablo to capture procedure call patterns. We also added Autopilot sensors to report entry and exit from logical code regions and to capture MPI message-passing activity. By selecting the NCSA site on the geographic display, users can drill down to a time-tunnel display of the Autopilot data. The color of each lateral time line identifies the currently executing code region (for example, fluid flow or structures) on each processor. In Figure 5, for example, the yellow time lines correspond to execution of the structure solver, with MPI reductions and sends and receives visible at the rear of the tunnel. The chords connecting time lines across the tunnel represent message-passing activity.

From a Virtue time-tunnel graph, a user can select the time line corresponding to a specific processor and drill down to display the associated call graph, as shown in Figure 6. Here, the data mapping associates vertex size with the number of times the corresponding code has executed on the associated processor, and it associates vertex color with the procedure's inclusive execution time.

As a complement to the real-time visualization, we also created a postmortem visualization by recording the Autopilot sensor data for replay. The postmortem visualization allowed us to share the results with various CSAR application developers and obtain two important insights. First, in both the real-time and postmortem visualizations, Virtue revealed major opportunities for optimizing the CSAR code's initialization phase. In the code's current configuration, initialization requires the exchange of many small MPI messages among processors and consumes a substantial portion of total execution time.

Second, we identified opportunities for dynamically adjusting the convergence criteria and execution balance across the code's fluid and structure components. We believe this capability offers the greatest promise for user-directed software tuning. By providing Autopilot sensors and actuators for parameter measurement and adjustment, Virtue will enable CSAR application developers to interactively explore the behavioral balance of code components.

In addition to suggesting possible application optimization targets, we also gathered valuable qualitative data on Virtue itself. First and foremost, the visualization system's graph-theoretical basis proved malleable enough to be successfully applied to a complex, multidisciplinary application, one of Virtue's primary goals. Further, we learned that the success of a large-scale visualization depends on virtual tools, which provide an efficient mechanism for manipulating data and changing views. Indeed, we believe Virtue needs a larger suite of even more powerful tools to fully exploit its visualization capabilities.

Although we have validated our approach with a large-scale application, much work remains. First, we continue to test Virtue with distributed applications as part of the National Science Foundation PACI (Partnership for Advanced Computational Infrastructure) program and the Department of Energy NGI (Next-Generation Internet) and ASCI (Accelerated Strategic Computing Initiative) programs. We are also developing more sophisticated network performance displays and virtual tools for exploring visualizations.

Using the rich set of network measurement data from the Very High Performance Backbone Network Service (vBNS), we are augmenting display metaphors with new visual attributes to represent a greater range of statistics. We are also developing a set of more powerful, domain-specific virtual tools. These include a critical-path finder, which will compute the limiting set of tasks in a time-tunnel execution display, and a statistical clusterer, which will identify the most relevant performance metrics for display.

Finally, we plan to continue our investigation of collaborative techniques. To enhance distributed interaction, we are generating VRML (Virtual Reality Modeling Language) snapshots of graph structures for Web export. We are also exploring navigation paths through annotation suites and implementing "flight recorders" for capturing user navigation paths through complex graphs. ❖

Acknowledgments

Ruth Aydt; Luiz DeRose, now at the IBM T.J. Watson Research Center; Mario Pantano, now at Anderson Consulting; Randy Ribler, now at Lynchburg College; and Ying Zhang all contributed to the development of the Autopilot toolkit.

The work described here was supported in part by the Defense Advanced Research Projects Agency under contracts DABT63-94-C0049, F30602-96-C-0161, DABT63-96-C-0027, and N66001-97-C-8532. Support also came from the National Science Foundation under grants CDA 94-01124 and ASC 97-20202 and the Department of Energy under contracts B-341494, W-7405-ENG-48, and 1-B-333164. The NSF PACI program provided additional support.

References

1. I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan-Kaufmann, San Francisco, 1999.
2. L. DeRose, Y. Zhang, and D. Reed, "SvPablo: A Multi-Language Performance Analysis System," *Computer Performance Evaluation Modeling Techniques and Tools, Lecture Notes in Computer Science*, Vol. 1,469, R. Puigjaner, N. Savino, and B. Serra, eds., Springer-Verlag, New York, 1998, pp. 352-355.
3. M. Calzarossa et al., "Medea: A Tool for Workload Characterization of Parallel Systems," *IEEE Parallel & Distributed Technology*, Winter 1995, pp. 72-80.
4. M.T. Heath and J.A. Etheridge, "Visualizing the Performance of Parallel Programs," *IEEE Software*, Sept. 1991, pp. 29-39.
5. L. DeRose et al., "An Approach to Immersive Performance Visualization of Parallel and Wide-Area Distributed Applications," *Proc. 8th IEEE Int'l Symp. High-Performance Distributed Computing*, IEEE CS Press, Los Alamitos, Calif., 1999, pp. 247-254.
6. I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *Int'l J. Supercomputer Applications*, Vol. 11, No. 2, 1997, pp. 115-118.
7. D. Reed et al., "Virtual Reality and Parallel Systems Performance Analysis," *Computer*, Nov. 1995, pp. 57-67.

Eric Shaffer is a research programmer in the Department of Computer Science of the University of Illinois at Urbana-Champaign. His research interests include computer graphics, visualization, and scientific computing. Shaffer has an MS in computer science from the University of Minnesota and a BS in mathematics and computer science from the University of Illinois at Urbana-Champaign. He is a member of the IEEE and the ACM.

Daniel A. Reed is a professor and head of the Department of Computer Science of the University of Illinois at Urbana-Champaign. His research interests include parallel computing, experimental performance analysis, and parallel input/output systems. Reed has a BS in computer science from the University of Missouri at Rolla and an MS and a PhD, also in computer science, from Purdue University. He is a member of the IEEE, the ACM, and the AAAS.

Shannon Whitmore is a research programmer in the Department of Computer Science of the University of Illinois at Urbana-Champaign. Her research interests include real-time performance analysis and scientific visualization. Whitmore has an MS in computer science from Oregon State University and a BS in computer science from Southern Oregon University.

Benjamin Schaeffer is a research programmer in the Department of Computer Science of the University of Illinois at Urbana-Champaign. His research interests include physics simulations and scientific visualization. Schaeffer has a PhD in mathematics from the University of Illinois and a BS in mathematics from the University of Chicago. He is a member of the Association of Symbolic Logic.

Contact the authors at {shaffer1, reed, swhitmor, schaeffr}@cs.uiuc.edu.