Visualization Viewpoints

Editor:
Theresa-Marie Rhyne

Dynamic Sharing of Large-Scale Visualization

Jian Huang, Huadong Liu, Micah Beck, and Andrew Gaston
University of Tennessee, Knoxville

Jinzhu Gao
Oak Ridge National Laboratory

Terry Moore
University of Tennessee, Knoxville

Visualization is a research tool that computational scientists use for qualitative exploration, hypothesis verification, and result presentation. Driven by needs for large user groups to collaborate across geographical distances (see the "A Driving Application—The Terascale Supernova Initiative" sidebar for detailed user needs), visualization must now also serve as an effective means to share concrete data as well as abstract ideas over the Internet. Yet there is simply no expeditious and practical way for users collaborating in this wide area to share large visualizations in a dynamic fashion.

In many cases, the only practical mechanism available for sharing visualization is to use a Web server to exchange precomputed images or videos. Without having a copy of the source data set on a locally accessible computing system, a collaborating user cannot dynamically interact with and tweak a visualization to answer questions formed spontaneously in his or her mind. If we could enable users to share and modify a visualization across the Internet without requiring local data replication, the visualization community would have an even greater impact on the conduct of today's research.

An adequate systematic study is obviously necessary to develop an optimal solution to this problem. In fact, several existing approaches—such as remote visualization methods[1,2]—could potentially be used. However, in this work we present a viewpoint tangential to those previous works. We believe that ordinary application scientists can use free, unscheduled, and unreserved resources on remote networked computers to visualize cutting-edge caliber data sets. Computing components do not need to be batch-scheduled. Those resources could sporadically become unavailable or faulty, but fault tolerance as well as scalability can be achieved. Scientists can use such resources not only for their own work but also, as a novel capability, to effectively share results among peers.

Using distributed heterogeneous resources as a basic parallel infrastructure to compute visualization could provide great potential usability, scalability, and cost efficiency. To justify our viewpoint, we describe a sample system of this nature and demonstrate its efficacy with a recently generated real-world large data set.

Using large-scale heterogeneous systems

Large-scale parallel systems are indispensable when visualizing large data. Currently, systems used for this purpose are usually locally accessible homogeneous clusters. Researchers less often consider scenarios in which processing nodes have different configurations or background workloads. Nor are many current packages designed with redundancy so that they can properly handle sporadic faulty nodes. While these issues might not matter greatly for parallel visualization of locally stored data, we argue that they do matter when using dynamic distributed resources. Native support of heterogeneity and fault tolerance is crucial to wide-area systems.

For the purpose of discussion, we refer to a prototype system used in our research. Our system's underlying infrastructure consists of a large number of shared heterogeneous processors connected by the wide-area Internet (see Figure 1 on page 22). Every processor has both a certain processing power and local storage space. We call each processing/storage node a *depot* (see the "Distributed System Infrastructures" sidebar near the end of the article).

To use the system, a user first partitions and uploads the data set onto a large number of depots, with k-way replication. k determines the degree of redundancy, which is usually as small as 3. We use a 30 time-step subset of a simulation produced by the Terascale Supernova Initiative (TSI)[3] for experiments, totaling 75 Gbytes. The data set has a spatial resolution of $864 \times 864 \times 864$ voxels. We partition each time step into 64 blocks and obtain 1,920 partitions. We upload the data set with three-way replication onto a total of 100 depots. Figure 1 illustrates the three replicas, each with a different color. The result is three complete copies of the TSI data set in the system. On average, each depot holds a 2.25-Gbyte subset of the data set.

The time to upload data is comparable to the time to move data between two locations. After the first copy is made, the remaining $k - 1$ copies are replicated in parallel using multisource multicasts. In our case, uploading 75 Gbytes with three-way replication almost always completes within 3 hours. This one-time cost of data movement allows a dispersed group of users to efficiently share their study and visualization of the same data for a few weeks, without a need to move the data again.

The multiple replicas of each partition of data are described by one common XML file called an *exNode*. Like Unix inodes, exNodes describe the mapping from logical files to stored blocks. Unlike inodes, exNodes are external and describe segments of logical files striped

 Published by the IEEE Computer Society

A Driving Application—The Terascale Supernova Initiative

In the past few years, computational science has been incorporated into the scientific enterprise as a third basic element, complementing theory and experimentation.[1,2] This field employs computationally intensive methods to simulate complex phenomena that are too expensive, too dangerous, or physically impossible to study by more traditional means. Near-term hopes of progress in answering a number of basic scientific questions, such as the production of heavy elements in supernovae, depend fundamentally on this new approach.

Experience has shown that research programs that revolve around big simulations, like those that revolve around big instruments, usually involve large, multidisciplinary, highly distributed teams of researchers. Such teams are necessary to create, maintain, and run the simulations, and even more so to digest and analyze their huge data outputs. For instance, in astrophysics, the Terascale Supernova Initiative (TSI) draws on special expertise from many physicists, as well as from applied mathematicians and computer scientists.[3] The data that results from massive TSI runs on the world's fastest supercomputers must be analyzed by scientists distributed across the US, including the Oak Ridge National Laboratory (ORNL), North Carolina State University, University of California at San Diego, Clemson University, Florida Atlantic University, State University of New York at Stony Brook, and University of Washington. The work presented in the main article used actual TSI simulation data.

So the general application scenario is one in which a group of users work jointly but often asynchronously across the wide area network on large data. For instance, TSI simulations run on supercomputers operated by ORNL and generate terabyte or larger data sets. After a simulation is finished, all TSI scientists would like to have access to this data so that each can work on (for example, analyze and visualize) the aspects of it in which he or she is interested. Now it would often be productive if they could share the visualizations they create with their colleagues, especially if they could communicate them not just as movies, but instead in a form that can be interactively modified. However, there are two major obstacles to realizing this advantageous possibility. First, the data is big, and therefore moving or copying it in the wide area is an expensive operation; after initial positioning (which might include replication), further movement needs to be minimized or performance will suffer. Second, interactive visualization requires access to substantial computing resources with the necessary software installed and access to the data with reasonable performance. Simultaneously satisfying both of these conditions for a community like TSI is a nontrivial challenge.

To adequately support such demanding needs by highly distributed research communities, not only do computer scientists need to make substantial improvements in hardware, software, networking, and data management components, but we also need to find more innovative ways to use those components. While we acknowledge that other alternative approaches exist, the viewpoint that we presented in the main article is one idea to support just this kind of application community.

References

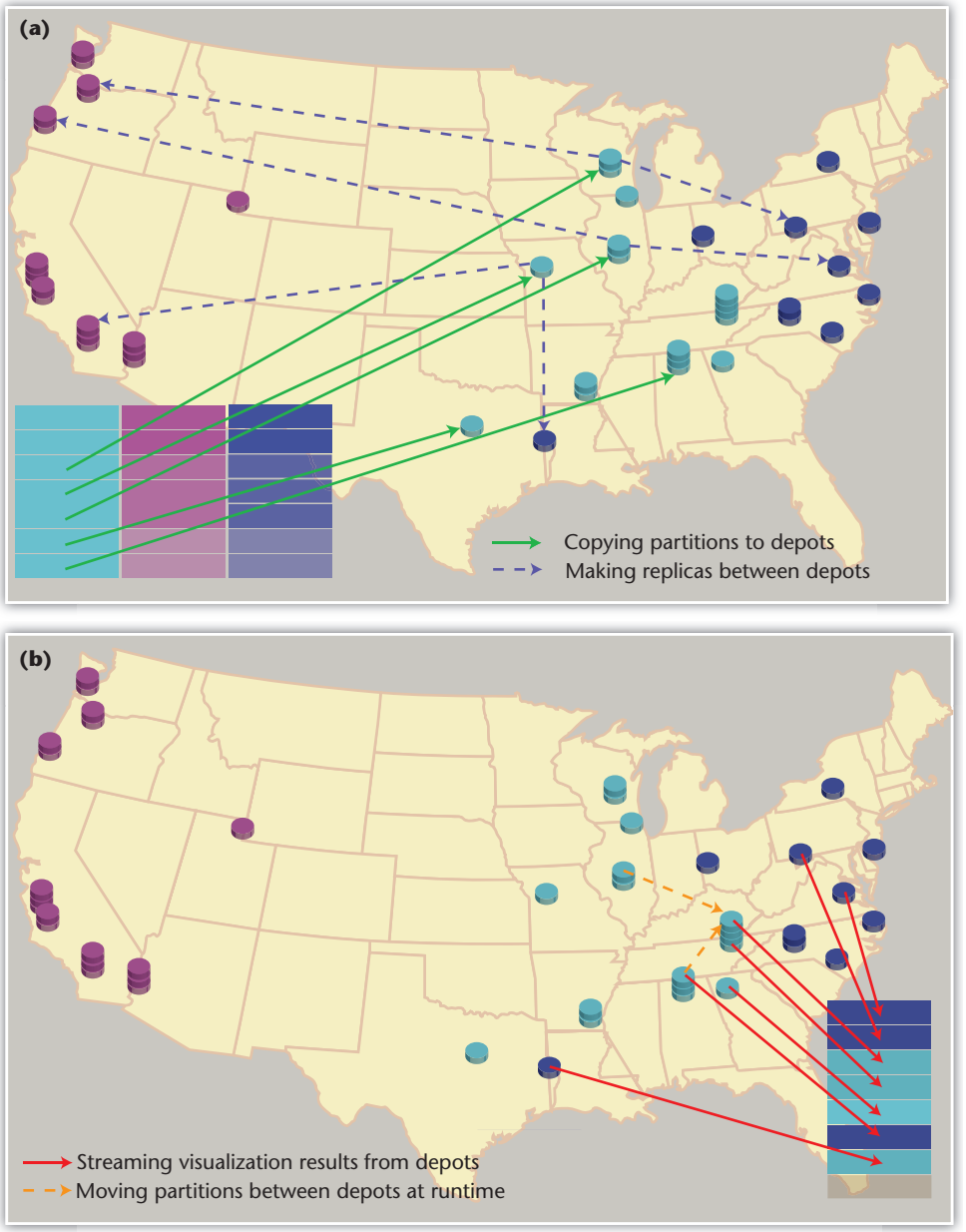
1. President's Information Technology Advisory Committee, "Computational Science: Ensuring America's Competitiveness," report to the president, Nat'l Coordination Office for Information Technology Research and Development, 2005; http://www.nitrd.gov/pitac/reports/index.html.
2. National Research Council, *Getting Up to Speed: The Future of Supercomputing*, National Academy Press, 2005.
3. R. Irion, "The Terascale Supernova Initiative: Modeling the First Instance of a Star's Death," *SciDAC Rev.*, vol. 2, no. 1, 2006, pp. 26-37.

across multiple networked depots. Besides showing how the data is distributed, exNodes also implicitly describe all processors available for use as well as the partitions associated with each.

Once the user uploads a data set and its exNodes are in hand, a user edits a template XML visSpec file (see Figure 2) to set up a visualization run. A user can also use a viewer program's GUI to generate a visSpec file. This way the user can avoid potential formatting mistakes or typos by not directly editing the underlying ASCII file. This visSpec file is similar to the lookmark in Paraview (see http://www.paraview.org) and contains all needed visualization parameters—for example, transfer functions specified as interval ranges, camera paths and matrices, and so on. However, unlike a lookmark, a visSpec file is not data set specific, and only defines a visualization when combined with exNodes describing an uploaded data set. By separating data from operation, this is suitable for use on distributed processors.

The same lightweight viewer program, run from a client machine, can open a visSpec file as well as data exNodes. The program spawns off parallel rendering operations by sending the visSpec file to all depots holding data partitions, as described by the exNodes. To orchestrate the distributed depots for parallel efficiency and fault tolerance, the viewer program schedules the parallel visualization run using methods detailed in the next section. The depots transmit the computed visualization results back to the viewer for assembly and final viewing. When an isosurface is requested in the visSpec file, the surface portions can simply be concatenated; for volume rendering, the viewer program composites imagery from all partitions in depth order.

A visSpec file is no larger than 4 Kbytes, while the compressed exNode files describing 75 Gbytes of TSI data with three-way replication is less than 600 Kbytes. A user can send them via email to collaborators, who can open the visSpec and exNodes using the viewer program on a laptop and spawn the visualization. The results are delivered to the viewer program on the fly. The use of XML as the uniform interface makes access to the visualization platform-independent. A user can also



1 Dynamic sharing of large-scale visualization using distributed, heterogeneous resources in parallel. (a) Large data sets are uploaded and striped in *k*-way replication for redundancy on distributed computers. (b) Given the XML specification of a visualization job and the data involved, any user can access and edit the specified visualization from a laptop using a lightweight viewer program, without the need for a priori resource reservation or going through a batch queue. Sharing an interactive visualization with a remote collaborator is as simple as emailing a generated ASCII XML file. In a way, this illustrates an interactive and cyclic process of specifying/modifying and inspecting visualization, by a distributed group of concurrent users.

make modifications to the visSpec using the viewer program. The compact visSpec files that arise during research can be saved and shared as well.

This prototype system makes several cumbersome tasks convenient for our end users, the application scientists. First, moving data is a long and error-prone process. The sustained transfer rates across today's Internet are in general no higher than 10 Mbytes per second. Moving just 75 Gbytes of data between two locations takes around two hours. In our system, once the data has been uploaded, no more data movement is necessary. Second, application scientists are not experts in programming parallel, distributed systems for fault tolerance and scalability. Our system makes all such important but tricky technical details transparent. Third, current large-scale systems are often operated by reservation schemes like batch queues. The undetermined amount of wait time through a batch queue is not the ideal for investigating spontaneously formed questions by visualization. Our system exclusively uses unreserved resources with no batch scheduling.

Fault-tolerant and scalable scheduler

The scheduler within the viewer program must be fault tolerant, since each depot in the wide area is only as reliable as the network to which it connects. Like the Internet, our system of hundreds of depots is best-effort with few absolute guarantees. In addition, geographically distributed users access the system with competing goals. To get the best performance out of heterogeneous, nondedicated depots, dynamic scheduling of computation and replication need to be tightly coupled. Scheduling of computation involves the assignment of parallel tasks, while scheduling of replication deals with runtime data movement among depots. They both aim at simultaneously balancing the computation workload and addressing fault tolerance. Our scheduler implements such coscheduling by adaptively measuring depot performances to dynamically assign visualization tasks and direct runtime data movements.

Adaptive computation scheduling

Our scheduler aims to discover fast depots on the fly, assign as many tasks to them as possible, and avoid being stalled by slow or faulty depots. We devised three generic mechanisms for this purpose:

- a dynamically ranked pool of depots,
- a two-level priority queue of tasks, and
- a competition avoidant task assignment scheme.

The scheduler ranks each depot by its estimated time to process a task of unit size. This measurement roughly reflects performance of depots delivered to the experiment. It is updated adaptively by computing a running average of measured depot performance, with older measurements given an exponentially decreasing weight.

A two-level priority queue maintains unfinished tasks. The higher priority queue (HPQ) contains tasks that are ready to be assigned, and the lower priority queue (LPQ) contains tasks that have been assigned to one or more depots but not finished. Initially, only the first w tasks are placed in HPQ, where w is the task window's size. It controls the degree of parallelism and number of tasks that can be finished out of order. In most cases, w is greater than the number of available depots so that every depot can contribute. However, w is decreased in case of severe resource contention as a processor back-off strategy. Each task in HPQ is keyed by the minimum unit task processing time of all depots holding the required data partition. This priority ranks new tasks by their likelihood of being finished by a fast depot. Each task in LPQ is keyed by its estimated waiting time. Tasks in both HPQ and LPQ are sorted by their keys in decreasing order.

When a parallel visualization starts, tasks in HPQ are sequentially assigned to available depots and demoted to LPQ. In case of failure, the task in LPQ is promoted back to HPQ so that other depots can take it over. When tasks are completed, every available depot will be directly assigned the first task in HPQ that it can handle. In this way, slow depots do not compete for tasks with fast depots to ensure fast depots be assigned as many tasks as possible. If a depot is not assigned a task in HPQ, the first task in LPQ that it can handle is considered. This is the slowest task among all unfinished tasks that the depot can help. Thus, multiple depots can work in parallel on the same unfinished task. If any instance of the duplicated tasks is done, other instances are aborted.

Dynamic replication scheduling

At any particular time, some data partitions might only reside on a set of slow or heavily loaded depots. In that case, fast depots cannot help because they do not have a local copy of the required data partition. A natural thought is to move data partitions to fast depots at runtime. To make sure that time spent on data movement does not exceed the benefit that we gain from migrating the task, bandwidth information between depots needs to be acquired. Instead of injecting extra testing traffic into the network, we devised a novel multisource partial download scheme with deadline for data movement between depots.

Ideally, an experiment's shortest execution time occurs when all depots finish roughly at the same time. Once the total of work each depot can do in HPQ is less than its proportion of all unassigned work according to its performance, the scheduler tries to transfer a task to it. The scheduler starts from the first task in HPQ, which has the least likelihood to be finished by a fast depot. To avoid always moving tasks out of the same set of slow depots, the task is moved only when all depots that can perform this task have sufficient work to remain busy.

```
<viewing_parameters>
        <lighting num_lights=2> ... <lighting>
        <viewport_transform> ... <viewport_transform>
        <data_specific> ... <data_specific> ...
<viewing_parameters>
<raycasting>
        <scale> ... </scale>
        <color_scheme> ... </color_scheme>
        <interval_range> ... </interval_range>
</raycasting> ...
```

2 A partial illustration of the XML visSpec file.

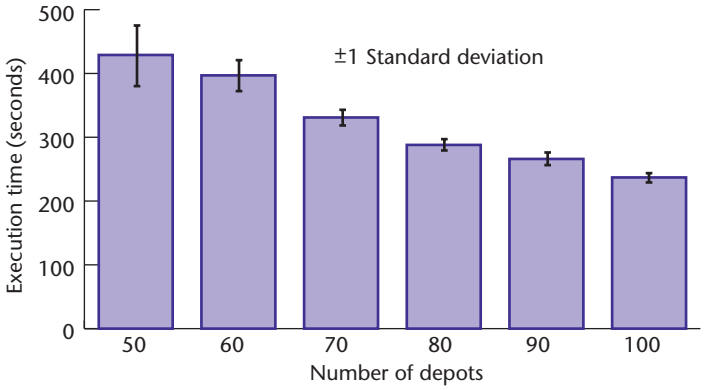
When a target depot and a slow task are picked, the maximum data transfer time allowed for the data movement is calculated as the deadline. We begin with a small fraction p of the partition. If the partial transfer completes in p of the deadline, we proceed to move the rest of the partition, otherwise, the data transfer is aborted. Since each partition is replicated on k depots, the destination depot receives data from multiple sources by using the progressive driven redundancy algorithm.

Results and discussion

We have only implemented volume visualization—in particular, software ray-casting and isosurface extraction—in our distributed system. To practically deploy our system to hundreds of depots—including Unix, Linux, and Mac servers—all source codes were written entirely in ANSI C. The resulting library is called the Visualization Cookbook Library (vcblib) with a compact binary size of 200 Kbytes. Although vcblib does natively handle differences in endian orders among processors, unlike other visualization packages, vcblib does not include function-rich but system-dependent modules, such as handling multiple data formats and user interfaces. We only support embarrassingly parallel visualization jobs.

For this article, we have successfully run experiments on 30 time steps of the TSI data set using 100 depots distributed across the US, Canada, and Europe—made available by the National Logistical Networking Testbed and the PlanetLab project (see http://www.planet-lab.org). In all experiments, we deployed vcblib as a dynamic library on every participating depot. Visualization operations in vcblib are loaded and executed in a sandbox to ensure the depot's security and stability, with each visualization job in a separately forked process. Multiple concurrent visualization jobs can simultaneously run on the same depot. No depots were reserved or running with a controlled workload.

The choice of which depots to use affects the overall performance. Unfortunately, there might not be a robust way to always make the best choice. Hence we designed the following experiment to measure performance (see Figure 3 on the next page). We randomly select a subset from the 100 nodes we use. Since the more we select, the more overlap there is among the selections, we randomly select 50, 60, 70, 80, and 90 nodes 10, 8, 6, 4, and 2 times, respectively. With each selection, we record the wall clock time in three back-to-back tests. We also run



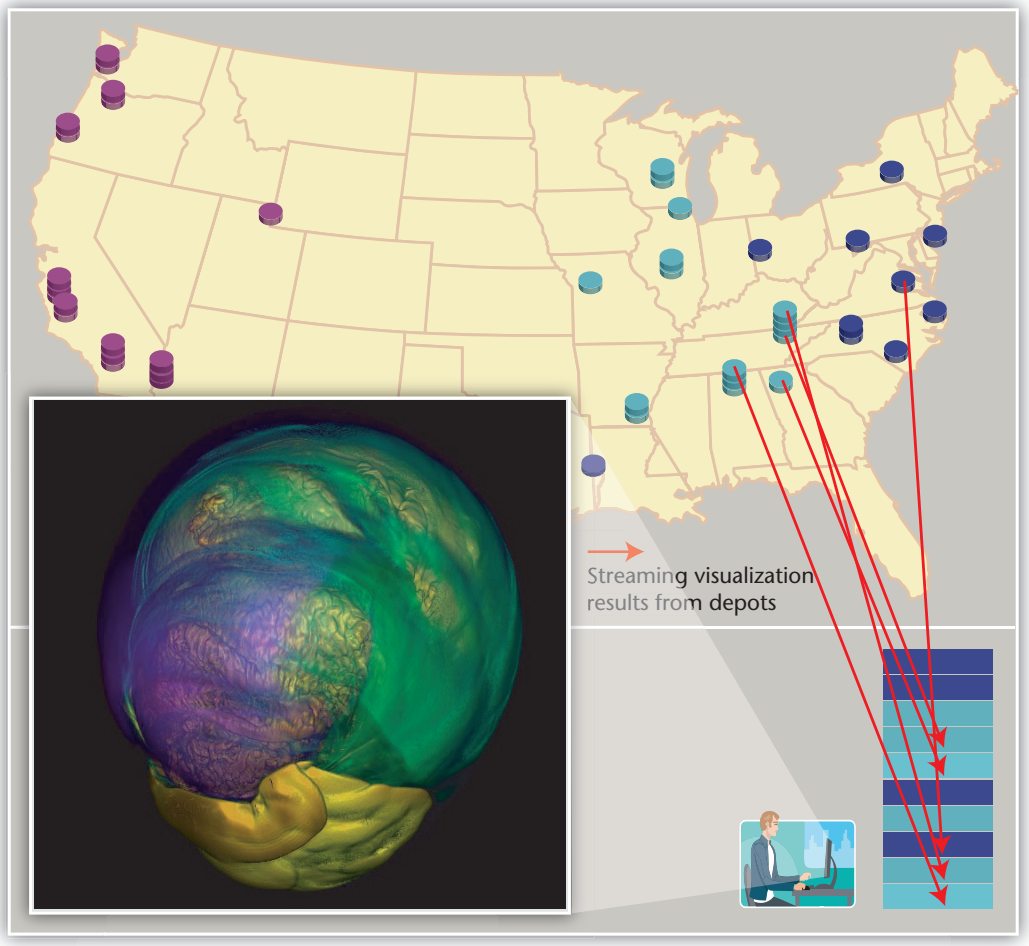
3 Average wall clock time shown with standard deviation. There are 30, 24, 18, 12, 6, and 3 separate measurements when using 50, 60, 70, 80, 90, and 100 depots, respectively.

tests using all of the 100 nodes for 3 times. All tests render a same 30 time-step animation.

On 100 depots, it takes on average about 237 seconds to complete software ray casting of 75 Gbytes of TSI data (with 800 × 800-pixel image resolution and a 0.5 step size). To provide a context, the same ray casting takes 219 minutes on a dedicated 2.2-GHz P4 CPU (with a 512-Kbyte cache). In other words, the performance achieved by 100 nondedicated, distributed computers of dispersed types (including Linux, Unix, and Mac) roughly equals that of a dedicated 64-node, 2.2-GHz cluster, assuming a 90-percent parallel utilization on the cluster. In the case of isosurface extraction, the comparison of performance remains similar. We have not incurred network bottlenecks in our tests. The highest rate of runtime network transfer was just a little over 1 megabit per second.

Conclusion

We still need to fully study the limits of this approach, particularly scalability versus data sizes, number of concurrent jobs, and number of users. Nevertheless, we hope to have demonstrated the feasibility and potential to use distributed computing and storage resources for serious visualization tasks. As Figure 4 illustrates, even without specially provisioned resources, a group of users can already share ideas by asynchronously viewing and dynamically modifying visualizations, using shared depots in a wide area in parallel. It seems a ripe time for the community to consider distributed and shared computing resources as a viable parallel infrastructure to support data-intensive remote collaborations. ■



4 An illustrated snapshot of our system in operation. The red arrows represent visualization results delivered by depots. The image is a frame from a volume-rendered animation of the TSI data set.

Distributed System Infrastructures

To implement a distributed visualization system like ours, we need as much support from distributed system infrastructures as available. Otherwise, the complexity necessary to achieve functionality, scalability, robustness, and performance will quickly overwhelm visualization researchers.

We could leverage several distributed infrastructures, but choosing among them is not simple. When making our choice, we had three primary concerns. First, the infrastructure should offer both storage and computing resources without the requirement of stringent authentication. Without this characteristic, it's extremely difficult for any individual visualization team to come up with a testbed of any practical significance. For instance, the 100 nodes we used are distributed across the continent of North America and Europe over 30 sites. Second, the infrastructure must be sufficiently robust, and hence redundancy for fault tolerance is required as a native ingredient of the infrastructure. It's rather unrealistic for software engineers to focus on the visualization and fault-tolerant distributed computing at the same time. Third, the infrastructure must have some basic constructs that can compose slices of storage and computing resources and form a higher-level abstraction. For instance, one such abstraction is the Unix inode, which hides the details of how bytes are organized on disk to form a file. The visualization community needs similar constructs in the distributed domain so that data partitions can be transparently striped across distributed computers.

We have chosen the Logistical Networking infrastructure, which meets all the noted requirements. Please refer to other works[1-2] for details of the Logistical Networking infrastructure and for its previous use in visualization.[3,4]

References

1. M. Beck, T. Moore, and J. Plank, "An End-to-End Approach to Globally Scalable Network Storage," *Proc. Sigcomm*, ACM Press, 2002, pp. 339-346.
2. M. Beck, T. Moore, and J. Plank, "An End-to-End Approach to Globally Scalable Programmable Networking," *Sigcomm Computer Comm. Rev.*, ACM Press, vol. 33, no. 4, 2003, pp. 328-339.
3. J. Ding et al., "Remote Visualization by Browsing Image Based Databases with Logistical Networking," *Proc. 2003 ACM/IEEE Conf. Supercomputing,* IEEE CS Press, 2003, p. 34.
4. J. Gao et al., "Distributed Data Management for Large Volume Visualization," *Proc. IEEE Visualization*, IEEE CS Press, 2005, pp. 183-189.

Acknowledgments

Our work was supported in part by National Science Foundation grant CNS-0437508 and US Department of Energy grants DE-FG02-04ER25610 and DE-FG02-06ER06-04. Data was provided by John Blondin and Anthony Mezzacappa under the auspices of the DOE SciDAC Terascale Supernova Initiative. We also thank the PlanetLab project for depot access.

References

1. W. Bethel, "Visualization Dot Com," *IEEE Computer Graphics and Applications*, vol. 20, no. 3, 2000, pp. 17-20.
2. J. Ding et al., "Remote Visualization by Browsing Image Based Databases with Logistical Networking," *Proc. 2003 ACM/IEEE Conf. Supercomputing,* IEEE CS Press, 2003, p. 34.
3. R. Irion, "The Terascale Supernova Initiative: Modeling the First Instance of a Star's Death," *SciDAC Rev.*, vol. 2, no. 1, 2006, pp. 26-37.

Contact author Jian Huang at huangj@cs.utk.edu.

Contact editor Theresa-Marie Rhyne at tmrhyne@ncsu.edu.