Scalable Grid-based Visualization Framework

M. Thiébaux H. Tangmunarunkit K. Czajkowski C. Kesselman

Information Sciences Institute
University of Southern California
Marina del Rey, CA 90292 U.S.A.

Abstract

Recent scientific and engineering advances increase the demands on tools for high performance interactive visual exploration of large-scale, multi-dimensional simulation and sensor-based datasets. For example, earthquake scientists can now study earthquake phenomena in detail via "first principle," physics-based, large-scale simulations in a time-volumetric space. Interactive visualization benefits the iterative scientific process to extract information from data and help scientists adapt their methods. Single-system visualization software running on high-end commodity machines can no longer sustain interactive browsing of these large science data due to their limited I/O and processing capabilities. A distributed and incremental approach is needed, to allow selective filtering of the parts of the data that the scientist wishes to view.

In this paper, we introduce a flexible and extensible Grid-based visualization framework for visual browsing of spatially and temporally large datasets in a Grid environment. Our framework leverages Grid infrastructure for access to shared scalable computation and data storage. In particular, we assume commodity machines for both user-interaction and bulk processing, exploiting distributed parallelism to scale the input data-handling capabilities. We describe the application of our visualization framework with data from the Southern California Earthquake Center ITR project, a large Grid-based science project. We report nearly ideal scaling results from a controlled experiment on small-scale, commodity, parallel storage and processing resources.

1 Introduction

Grid technology is enabling collaboration between scientists and organizations from multiple organizations to generate and archive extremely large datasets across shared, distributed resources [10]. Grid methods are increasingly being used by many scientific applications such as particle physics, climate modeling, astrophysics, earth science, earthquake engineering and science [5, 14, 22, 18, 17, 13]. These applications make use of high-performance computing resources and storage systems to produce large multi-dimensional data sets, with the goal to gain better understanding of studied physical phenomena. For example, earthquake scientists can now study earthquake phenomena in detail via "first principle," physics-based, large-scale simulations [19, 4]. These simulations can generate large amounts of 4D data (high resolution and long time series).

These scientific and engineering advances increase the demands on tools for high performance interactive visual exploration of large-scale simulation and sensor-based datasets. Interactive visualization can benefit the overall scientific process by enabling an iterative feedback mechanism where scientists can interactively explore and analyze the data in order to quickly extract information from it and adapt their methods based on what is learned. The scientist must visually explore the data find and validate this information, except in mature domains where specialized *feature-extraction* methods are known that can operate without human guidance.

To increase the rate of the scientific process, interactive visual exploration should proceed immediately (and selectively) on the raw data. Single-system visualization software running on high-end commodity machines can no longer sustain interactive browsing of these large science data, because the I/O and filtering capabilities of such systems do not scale. Contemporary single-system approaches require pre-analysis in order to reduce the data before starting the visualization session [24, 3]. A distributed and incremental approach is needed, to allow selective filtering of the parts of the data that the scientist wishes to view.

In this paper, we introduce a flexible and extensible Grid-based visualization framework for visual browsing of spatially and temporally large datasets in a Grid environment. Our framework leverages Grid infrastructure for access to shared scalable computation and data storage. In particular, we assume commodity machines for both user-interaction and bulk processing. We exploit distributed parallelism to scale input data-handling capabilities. Thus, aggregate data storage and

input processing capabilities can be increased to match scientific data growth, even when the amount of information viewed at any one time by the scientist remains constrained by display devices [3]. Similar distributed parallelization can also be applied to scale up the information viewing capabilities, for example by parallelizing the graphics and display hardware for large tiled displays [11, 20], a topic not covered in this paper.

Scientific visualization is an effort to integrate visual methods into the scientific process and use them as a means to guide the further work. Many science projects are using Grid methodologies to generate and manage their data. Therefore, visualization tools not only have to support large-scale data sets, but must also be developed to provide this support in the Grid environment. Our visualization framework is designed to satisfy both requirements.

In the remainder of this paper, we motivate the need for scalable techniques. Then, we introduce our framework architecture and explain its main performance-oriented features. Finally, we describe the application of our visualization framework with datasets from the Southern California Earthquake Center (SCEC) ITR, a large Grid-based science project [13]. We report nearly ideal scaling results from a controlled experiment on small-scale, commodity, parallel storage and processing resources. These results show practical interactive performance with SCEC data and suggest that our framework will scale to even larger datasets.

2 Visual Exploration of Scientific Data

Visual exploration allows scientists to interact with their data to make discoveries about its structure and evolution; this process involves interactively browsing the data in both temporal and spatial dimensions, concurrent with the manipulation of visualization parameters that highlight different aspects of the data. Visual exploration supports validation during algorithmic prototyping and implementation, browsing and mining of full-scale scientific data, and communication of scientific results. The overall scientific exploration process is thus iterative at two levels:

Hypothesize scientific ideas that need to be tested.

Experiment to gather data, *e.g.*, run numerical simulations or gather sensor data from physical experiments.

Analyze data to discover information that proves or disproves hypothesis.

- *View selected data* using appropriate methods to project the scientific data into images.
- *Adjust methods* and iterate.

Learn and iterate

In nascent fields such as physics-based earthquake simulation [13], the scientists are seeking a wide range of information about their simulation methods and about the complex behaviors of a physical system governed by their theoretical models. There is great value in having short turn-around times in both the visual browsing "inner loop" and the whole scientific process that motivates their simulation studies.

2.1 Scalability Issues

To understand the need for parallelization, consider the input data-handling capacity of a non-distributed visualization system on high-end commodity machines. Retrieval of data from local storage devices and processing of data are basic operations in any visualization process. Let , , represent the volumetric resolution; represent the number of bytes of field value in the volume; represent the storage reading rate; represents the total data access time; represents the total data processing time; then , and for many simple filters.

We have measured raw disk read rates on multiple high-end personal computers; the highest sustained rate that we observed is 70 MiB/s[1] with "RAID0" striping across three large ATA-100 disks. With actual SCEC data of resolution and one byte scalars, the read time would be over 3.5 seconds simply due to the limited disk I/O rate. Animation rates below one second per image are desired for interactive browsing. Our actual processing algorithm (necessary to convert the volume into a form the graphics hardware can handle) benchmarked on the same machine actually takes 4.9–9.7 seconds including read time, depending on filter parameters. This approximates a stand-alone application performance where we consider the rendering overhead to be minimal compared to the read and process stages, *i.e.*, within the capabilities of the hardware-accelerated renderer.

[1] *Megi-bytes* (MiB) uses an NIST proposed notation for base-2 multipliers, *e.g.*, the "Mi" multiplier equals and "Ti" equals .

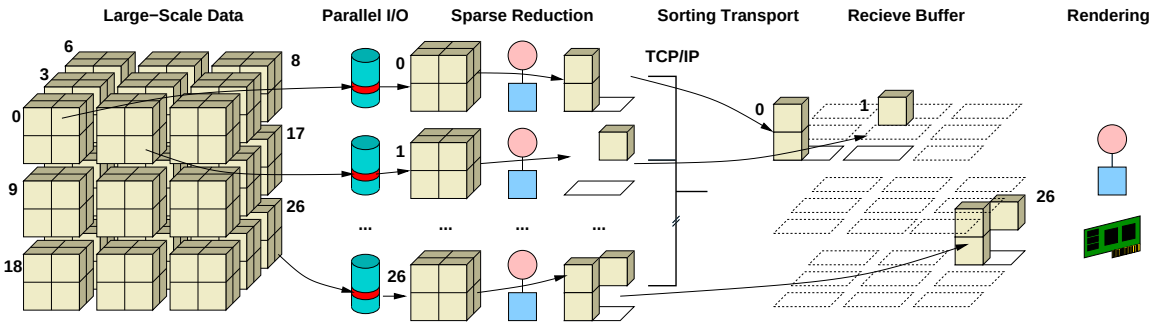

Figure 1. Parallel, Distributed Visualization Datapath. Block decomposition supports scaling of data storage and data processing.

2.2 Parallel Distributed Visualization

Clearly, a commodity system such as the one discussed above cannot support interactive browsing of large scale data due to these limitations in the storage and data processing components. Other researchers have explored specialized hardware solutions to accelerate volume rendering with PCs, but these are stop-gap measures in the face of the cubic growth in input I/O requirements as simulation resolutions are increased. SCEC data is expected to grow to as the project proceeds. Even a modest increase to will cause data read times of 14.6 seconds, and the full resolution data would cause read times of 117 seconds! Assuming a generous doubling in disk performance for the personal computer by then, the data access times would still be prohibitive. Also, the time-volumetric data will require 960 GiB of storage, which is still an uncommon amount of storage to have available in one personal computer.

To address this issue, other researchers have built large-scale parallel renderers (hosted on supercomputers), and various distributed visualization techniques (running on a collection of resources) where data access and data processing can be done in parallel to serve one or more remote display clients. These techniques all exploit inherent reduction in volumetric data to a two-dimensional image space. Such approaches make different trade-offs between resource cost, input data size and time-series length, image rate, and image latency. The framework we propose is one such solution, aiming for data handling scalability and low image latency on commodity hardware.

3 Scalable Grid-based Visualization

Our framework leverages Grid resources for scalable computation and data storage to enhance the performance, functionality and flexibility of a light-weight PC-based graphical tool. A parallel data processing service is distributed out to Grid resources and controlled by the user through the graphical user-interface (Figure 1).

Figure 2 shows the layered architecture of our visualization framework. At the bottom layer, we utilize the following external software: the Globus Toolkit; OpenGL; and GLUT. The Globus Toolkit provides us with APIs for security (single sign-on and authenticated sockets), resource allocation and management, high-performance asynchronous socket communication, and multi-thread programming libraries. We use these features to couple commodity desktops with remote storage and computational resources. OpenGL and GLUT combine to provide a standard programming interface to hardware-accelerated rendering functions without concern for platform-specific details.

3.1 Framework

Our framework realizes an *active storage* model, where online storage is combined with general-purpose processors allowing data processing to be done with high I/O connectivity. Active storage systems are intended to store large amounts of application data in a persistent or scheduled way, while simultaneously hosting scalable, domain-specific or application programs. In addition, by executing our visualization data reduction filters at the storage system, we are able to further optimize the network-bound data for the viewing parameters selected by the user. Figure 3 shows three types of nodes:

Client. The client node is located at the user's graphics workstation and issues requests into the pipeline.

Servers. Server nodes perform the distributed parallel disk access processing that provides results to the client.

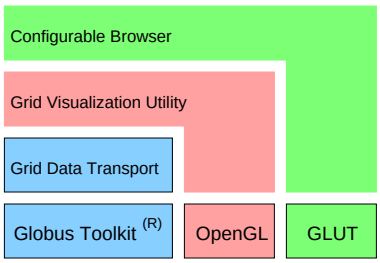

Figure 2. Grid-based Visualization Architecture

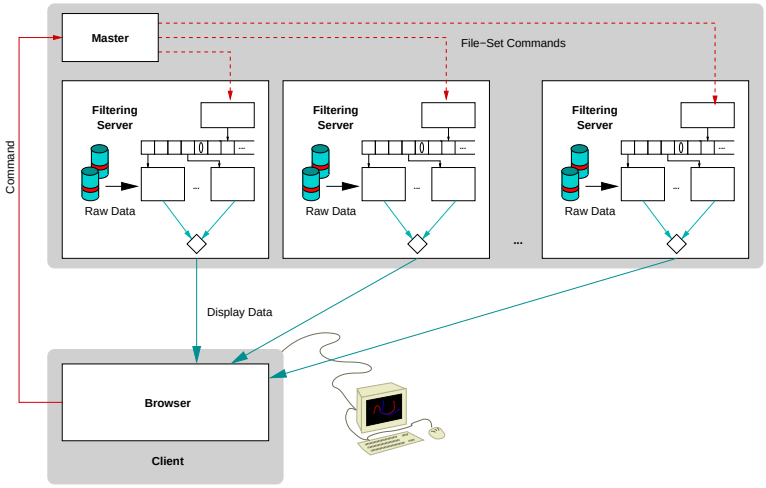

Figure 3. Active Storage Model for Volume Browsing

Master. The master node is responsible for decomposing client requests into the explicit parallel processing tasks performed by the servers.

3.2 Implementation Components

The framework is implemented in layers by three components that add increasingly application-specific functions:

Grid Data Transport (GDT). The GDT library provides the ability to perform efficient *pipelined*, parallel data processing and parallel data exchange among distributed Grid resources.

Grid Visualization Utility (GVU). GVU is designed to assist with dataset management, filtering algorithms, and transport, including efficient bulk data processing in heterogeneous Grid environments. GVU utilizes GDT for resource management, data transport and control.

Our configurable browser application uses GVU and GDT to provide a desktop data exploration tool that uses remote Grid resources in order to scale up its data access and processing capabilities.

We describe each of these components in more detail below.

3.2.1 Configurable Browser

The browser implements our client functionality. It provides a graphical user-interface for users to adjust or modify data selection and viewing parameters, and renders display lists sent by the remote servers. In our study, it uses the distributed datapath for remote synthesis of view-point independent point lists, which allows the browser to control the view point and re-render volume data locally with low view-point latency. With hardware-accelerated drawing, this strategy reduces the client processing work to a bare minimum. Display lists arrive in the format required for the OpenGL rendering API, and the client merely buffers them “off the wire” and to the main draw routine.

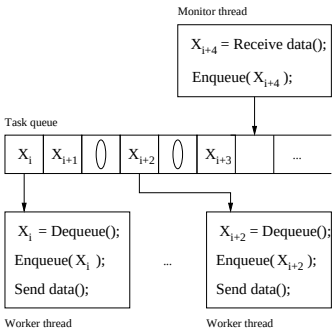

Figure 4. Server Algorithm. Fine-grained processing of file-set elements are partially ordered by inter-set bubbles.

Command flow-control to the master is enforced by the browser to restrict the maximum number of commands and data in transit through the datapath. User-interface events cause commands to be sent only while the *command depth* is below the maximum limit which can be dynamically adjusted by the user. This limit allows the user to trade throughput for control latency to suit her requirements.

3.2.2 GVU: Grid Visualization Utility

The GVU module defines a functional layer of utilities specifically for handling spatial datasets, filter and OpenGL rendering operations, file and transport formatting, as well as a suite of commands used to drive the visualization framework. In short, it extends the underlying GDT transport mechanism by adding visualization-specific functionality for a parallel environment.

The GVU library defines a hierarchy of dataset management and transport classes for a superset of VTK-style (Visualization ToolKit) geometric representations commonly used in scientific visualization [23]. It provides a suite of basic filters for data analysis and reduction, such as volume cropping, down-sampling, point-sampling, and scalar range culling, for interactive visualization of volumetric time-series. The GVU interface also supports plug-in extensions to facilitate the use of existing reduction filters, such as the marching-cubes isosurface algorithm, or other custom domain-specific filters within a scalable framework.

The internal field-data representation of the GVU dataset superclass is modeled after the mutually distinguished index, cell, vector, and scalar arrays of the VTK data structures. These have been extended to include all common data precision types, as well as user defined fields. The GVU input/output handles correct for foreign byte ordering, and provide a clean interface for optional pre-swapping of bytes before or during the GVU send operation.

GVU also provides the queue structures necessary for buffering of datasets as they are received from parallel sources, and handed off to the drawing thread in the browser.

3.2.3 GDT: Grid Data Transport Library

To support the processing requirements of the browser and GVU layers, the GDT library provides a *file-set processing and transfer* mechanism. A file-set is a set of data elements (logically referred to as files) that are a physical decomposition of a dataset. For example, a volume might be split into 64 files in a file-set (each in size). A file-set can contain either control commands or data. At a server node, a file-set corresponds to a set of subcommands defined by the master node in response to a single client command, as well as the results of those commands. At the client, each file-set represents the parts of a filtered volume that the browser may render one or more times (with different viewing parameters). The browser cannot render unless it holds a complete file-set.

While there is no order requirement among elements of a file-set, GDT defines a FIFO order of whole file-sets. GDT provides libraries and templates based on the file-set concept that can be used to integrate data sending, receiving, and processing routines. The data transport routines can be either blocking or non-blocking. These templates can then be used to develop an arbitrary visualization *datapath node* in a distributed environment. GVU relies on GDT to complete each of the distinct processing functions.

An example use of GDT is the server node algorithm shown in Figure 4. There is a task queue, a *monitor* thread, and one or more *worker* threads in each server node. The monitor thread receives commands and puts them into the task queue. When idle, each worker grabs the next command in the task queue, processes it, and sends the processing results to the next tier (*e.g.*, the browser client in our study). The multiple workers exploit fine-grained concurrency in each server node.

Commands related to different file-sets are separated by a "bubble," to robustly detect file-set boundaries while permitting the number of elements in a file-set to change dynamically. GDT ensures that all tasks preceding the bubble begin processing before any tasks following the bubble. Similarly, all results of tasks preceding the bubble must be sent before any results of tasks following the bubble.

In addition to allowing arbitrary data filtering or processing plug-ins, GDT also facilitates multi-tiered pipeline topology construction of compute resources and display units. Given a topology specification (provided at run-time), GDT exploits Globus GRAM [8] and DUROC [9] for remote resource invocations and co-allocation and application bootstrapping among multiple resources. After all the components in the pipeline are running, GDT then establishes inter-tier communication channels ready for the application datapath.

The framework is designed with many generic components that are applicable to more complex node topologies. Our current implementation already supports multiple tiers of server nodes for additional pipeline stages. However, for the application studied in this paper, there is only one tier of servers.

3.3 Striped, Pipelined File-Set Transfer

For high-performance, the GDT file-set transfer mechanism implements *striped pipelining*. *Striping* means that GDT supports parallel processing and bulk data transfer among loosely-coupled parallel senders and receivers, *e.g.*, from parallel server nodes to our client. *Pipelining* means that GDT supports high-throughput streaming: different nodes or the various stages of a single node may operate concurrently on subsequent file-sets, and transfer links may have several "in flight" file-sets in order to fill the bandwidth-delay product of fast networks. Thus, GDT supports a wide-range of high-performance, parallel and distributed datapath scenarios.

Datapath stages apply filtering or transformation algorithms to each file in a streaming fashion. The strict FIFO ordering of file-sets prevents any load-imbalance in a distributed datapath from leading to dangerous priority inversions, *i.e.*, to prevent work on step being blocked by premature work on step . FIFO transport prevents buffer contention where space needed to receive step data is occupied by step data. File-set streaming over parallel TCP streams yields high throughput in practice, while still permitting receiver-triggered flow-control or *throttling* to prevent buffer overruns.

4 Testbed and Performance

In this section, we report on the performance of our Grid-based visualized software using a 4D dataset representing a dynamic rupture model from the Southern California Earthquake Center (SCEC). The original data set consists of 120 time steps of byte scalars. For each pipeline configuration, the original time-volumetric data is decomposed into 256 small volume files per timestep. Each file is approximately , or just under 1 MiB in size. This set of 30,720 files (for all 120 timesteps) is generated and distributed among the participating servers in a round-robin fashion during staging. The same strategy is used by the master to distribute file-processing commands during the visualization sessions.

4.1 Testbed

The visualization pipeline shown in Figure 3 is deployed on:

A 16-node Linux cluster with dual 550 MHz Intel Pentium III processors and two striped Ultra160 SCSI disks on each node. The cluster is installed with an LSF scheduler that can start parallel jobs on a selective set of nodes, and Globus Gatekeeper (Globus Toolkit 2.4.3) for remote task invocation. The master and servers are deployed on this cluster forming an active storage system. Each server runs with 12 working threads and opens 8 channels to the client in the pipeline.

A high-end Linux display unit with a consumer-level NVIDIA graphics accelerator capable of drawing 30 million points per second. This machine is used for the visualization client.

A network switch. The cluster nodes and visualization client are connected together via full-duplex, switched gigabit Ethernet.[2]

[2]For most of the voxel interest-ratios used in this experiment, a switched 100baseT connection to the visualization client would not have limited the observed performance.

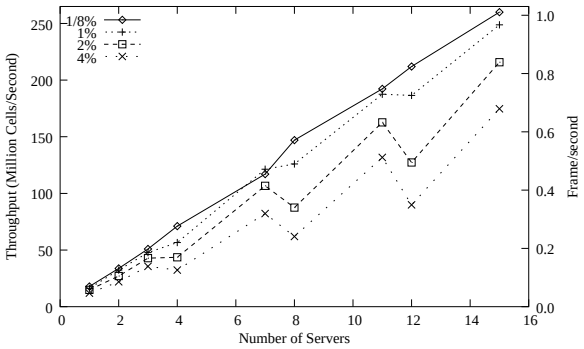

Figure 5. Performance scaling for various numbers of servers. Each curve shows the performance for one fixed interest ratio.

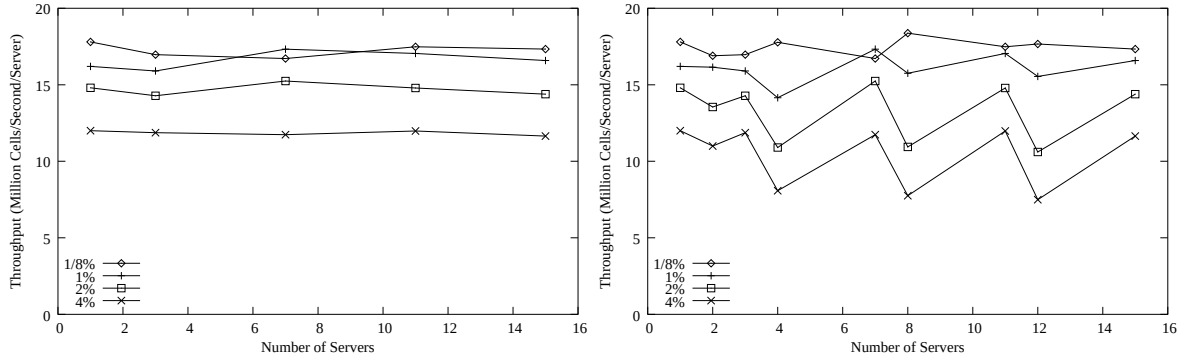

Figure 6. Performance scaling for various numbers of servers, normalized by server count. The left-hand plot omits the node counts that suffered static load-imbalance due to interaction between the node count and round-robin stride in the full plot on the right.

4.2 Experiment

To study the scaling behavior of distributed pipeline, we vary the number of participating servers and measure the sustained throughput in terms of the effective cell-processing rate. The cell rate is the product of the frame rate and the number of cells (voxels) in the input dataset, *i.e.*, 1 frame per second equals 257 million cells per second. We use the cell processing rate to compare system throughput in a way that is independent of the input data size. Such rates help to estimate performance with different datasets and also are more intuitive when studying speed-up due to parallelization.

The *interest ratio* is the percentage of cells contributing to an image, as determined by visualization parameters. This ratio affects performance due to differentiated filter costs for input and output handling. In this experiment, we fix the visualization parameters to ensure repeatability. We choose a representative time-step from the original dataset and duplicate this volume to generate a long constant time-series, forcing the interest ratios to be the same while processing sufficiently large quantities of data. This allows us to have a controlled experiment for studying the scaling behavior of the distributed datapath. For comparison, we also show the 15-server case with the real time-series data in Figure 7.

4.3 Performance

Figure 5 shows the scaling behavior of our pipeline in terms of number of servers. The x-axis shows the number of servers participating in the pipeline, and the y-axis shows the pipeline throughput in terms of million cells per second. Each curve corresponds to different interest ratios. The ragged performance profiles are due to load-imbalance where our naive round-robin data distribution across servers interacts with localized distribution of interesting voxels in the volume. With unlucky stride frequencies, the sub-volumes with most of the filtering work are on a smaller subset of the servers. This effect is amplified as interest ratios increase.

Normalizing by node count makes the scaling properties of the system more apparent in Figure 6. An ideal (linear) scaling condition would appear as a horizontal line. The near-perfect scaling with server counts 3, 7, 11, and 15 are due to

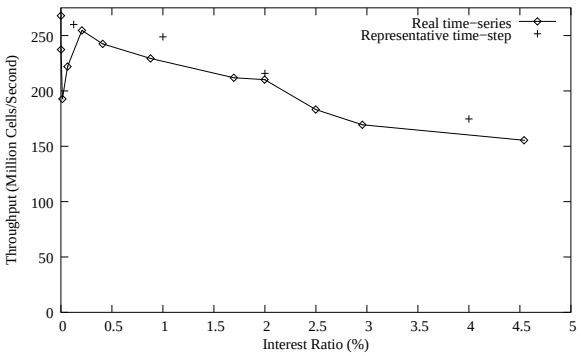

Figure 7. Actual performance with time-series data on 15 servers, still using round-robin data placement. For comparison, four controlled measurements with representative timestep on 15 server case are re-plotted. The fixed-parameter run on real data causes a wider interest-ratio distribution. A load-balancing anomaly is apparent at extremely low interest ratios.

the lack of common divisors between the node count and our block stride, *i.e.*, the round-robin schedule applies a crude form of hashing from subblock address to server. These results suggest that simple changes to the static data distribution could prevent load-imbalance in practice.

We have varied the experiment to exclude the drawing routine, with no significant impact on the results. This is because the output generated during the visualization process is within the capability of our client machine's graphics card. We have also run the client on different machines with various network and processor capabilities and observed similar performance. These experiments confirm that the bottleneck with these large data problems is at the input data handling stage, not at the final rasterization stage.

5 Related Work

The DataCutter system [7, 6] is similar in spirit to the architectural approach we take, in that it assumes tera-scale data and parallel processing to perform data reduction local to parallel storage. However, DataCutter is a more general stream filtering toolkit with an emphasis on more irregular problems and a coarser granularity of request, with a research focus on data indexing strategies and other optimization techniques rather than the bulk data delivery challenge. To our knowledge, it has not been applied to volume rendering nor to interactive visualization at sub-second frame rates.

Underlying this interactive visualization problem is a parallel and distributed control and data transport problem. The GridFTP [2] protocol is an extension to the Internet-standard File Transfer Protocol (FTP) to support parallel and *striped* data delivery by *extended-block* requests. An existing striped GridFTP server supports filtering plugins that operate at the extended-block level, much like our GDT mode supports visualization filters operating on file-sets. However, we are unable to exploit this work due to limitations in the control and data protocol:

> GridFTP cannot pipeline commands, so there is a minimum of one round-trip delay between each adjacent response.
>
> The GridFTP filter model assumes that output size is calculated only based as a function of input block size, rather than input contents and so is ill-suited to streaming of variable-sized file-set elements.

In our work, we have assumed an active storage model where scientific data are relatively local to CPUs, *i.e.*, local data access scales with the computational capability. The Optiputer Project [20] is exploring applications of wide-area "switched lambda" connectivity where bulk data can be transported in a striped fashion to a parallel recipient before being redistributed among the recipients nodes based on application requirements. Our framework can be adapted to work in the Optiputer environment.

A number of parallel rendering systems focus on scaling the filtering and rendering algorithms without any attempt to scale the input I/O, often relying on in-core benefits [15, 12] or hardware accelerators that perform well until their buffer space is exceeded [21, 16]. There are suitable for working with single volumes, *e.g.*, medical data, but do not solve the exploratory browsing problem for volumetric time-series data. The popular VTK [23] environment advocates a "demand-driven data-flow semantics" that prevents streaming pipeline parallelism. Other work on parallelizing VTK [1] does not combine data and pipeline parallelism, although it discusses them separately. Other approaches require extensive pre-processing to build

intermediate data over which a more efficient filtering and rendering algorithm can be applied at or near interactive rates [24, 3].

6 Conclusions and Future Work

In this paper, we have described a Grid-based visualization system that exploits distributed parallelism to scale the input data-handling (*i.e.*, data accessing and data processing) capabilities in a Grid environment. Our framework achieves parallelism at two levels; the coarse-grained resource level by distributing visualization tasks out to multiple distributed resources for parallel execution via the active storage concept, and the fine-grained task concurrency on individual node level by permitting dynamic reordering of data elements under the file-set transfer concept.

We have used our visualization framework with stereotypical Grid resources to construct a simple visualization pipeline for visualizing many scientific applications. Examples are 3D medical images and Geo-physics 4D dynamic fault rupture, anelastic wave propagation, and Fretchet Kernel analysis [19, 4, 25]. Our 4D visualization has helped SCEC geoscientists gain insight into earthquake processes. We have demonstrated the framework's capability during the SC'03 conference.

As mentioned earlier, scientific data archived in a remote storage system has to be bulk-transfered and distributed among the servers prior to browsing. Currently, the data is simply distributed uniformly among the servers in a round-robin fashion. However, different servers may have different capabilities and the pipeline efficiency is limited to the slowest server. Our results illustrate the sensitivity of our static data distributions to load-imbalance, but also illustrate that trivial hashing techniques can balance the load in practice on homogeneous clusters. It is not clear if or when more elaborate dynamic load-balancing might be required to continue scaling—much larger-scale experiments will be required to determine where the static volume decomposition becomes an obstacle.

Distributed parallelization can also be applied to scale up the information viewing capabilities, for example by parallelizing the graphics and display hardware for large tiled displays. We plan to investigate issues regarding the output handling to support multi-tiled display system.

References

[1] J. Ahrens, C. Law, W. Schroeder, K. Martin, and M. Papka. A parallel approach for efficiently visualizing extremely large, time-varying datasets. Technical Report LAUR-00-1630, Los Alamos National Laboratory, 2000.

[2] B. Allcock, J. Bester, A. Chervenak, I. Foster, C. Kesselman, V. Nefedova, D. Quesnel, and S. Tuecke. Secure, efficient data transport and replica management for high-performance dataintensive computing. In *Mass Storage Conference*, 2001.

[3] L. S.-K. B. Csèbfalvi. Monte carlo volume rendering. In *Proceedings of IEEE Visualization Conference*, Seattle, Washington, October 2003.

[4] H. Bao, J. Bielak, O. Ghattas, D. O'Hallaron, L. Kallivokas, and J. X. J Shewchuk. Earthquake ground motion modeling on parallel computers. In *Proceedings of Supercomputing'96*, Pittsburgh, PA, Nov. 1996.

[5] B. Barish and R. Weiss. LIGO and the detection of gravitational waves. *Physics Today*, 52(10):44, 1999.

[6] M. Beynon, C. Chang, U. Catalyurek, T. Kurc, A. Sussman, H. Andrade, R. Ferreira, and J. Saltz. Processing large-scale multidimensional data in parallel and distributed environments. *Parallel Computing (Special issue on Parallel data-intensive algorithms and applications)*, 28(5):827–859, May 2002.

[7] M. Beynon, R. Ferreira, T. Kurc, A. Sussman, and J. Saltz. Datacutter: Middleware for filtering very large scientific datasets on archival storage systems. In *Proceedings of the 2000 Mass Storage Systems Conference*, pages 119–133, March 2000.

[8] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In *The 4th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82, 1998.

[9] K. Czajkowski, I. Foster, and C. Kesselman. Co-allocation services for computational grids. In *Proc. 8th IEEE Symp. on High Performance Distributed Computing*. IEEE Computer Society Press, 1999.

[10] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 2003.

[11] M. Hereld, I. Judson, and R. Stevens. Introduction to building projection-based tiled display systems. *IEEE Computer Graphics and Applications*, 20(4):22–28, 2000.

[12] G. Johnson and J. Genetti. Volume rendering of large datasets on the Cray T3D. In *Cray User Group: 1996 Spring Proceedings*, pages 155–159, 1996.

[13] T. H. Jordan, C. Kesselman, J. B. Minster, and R. Moore. The SCEC community modeling environment—an information infrastructure for system-level earthquake research. *NSF Proposal*, 2001. http://www.scec.org/cme.

[14] LHC computing grid project. http://lcg.web.cern.ch/LCG/.

[15] K. Ma, A. Stompel, J. Bielak, O. Ghattas, and E. Kim. Visualizing very large-scale earthquake simulations. In *SC 2003*, Pheonix, Arizona, November 2003.

[16] P. McCormick, J. Qiang, and R. Ryne. Visualizing high-resolution accelerator physics. *IEEE Computer Graphics and Applications*, 19(5):11–13, Sep/Oct 1999.

[17] NEESgrid project. http://www.neesgrid.org.

[18] National virtual observatory. http://www.us-vo.org.

Machine	Storage Device	CPU Speed (GHz)	Memory (GB)	Raw Read I/O (MBps)
titan	1x7200RPM IDE	1 x 2.2	1	37
crater	3x7200RPM IDE (striped)	2 x 1.7	0.7	60–70
dc-n 1,...,16	2x10000RPM SCSI (striped)	2 x 0.55	1.5	38–65
tubby	2x10000RPM SCSI disks	2 x 2.2	1	22–41

Table 1. Hardware configuration and I/O performance. We present a range on the I/O performance, since different disk sectors (and disk fragmentation) yield different I/O rates.

[19] K. B. Olsen. *Simulation of Three-dimensional Wave Propagation in the Salt Lake Basin*. PhD thesis, University of Utah, Salt Lake City, Utah, 1994.

[20] Optiputer project. http://www.optiputer.net/.

[21] H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler. The VolumePro real-time ray-casting system. In *Proceedings of SIGGRAPH '99*, pages 251–260, July 1999.

[22] Particle physics data grid. http://www.ppdg.net.

[23] W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit: An Object Oriented Approach to 3D Graphics*. Prentice Hall, 2nd edition, 1998.

[24] M. Xiaoyang. Splatting of non rectilinear volumes through stochastic resampling. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):156–170, June 1996.

[25] L. Zhao, T. Jordan, and K. Olsen. Frechet kernels for imaging the los angeles basin structure based on three-dimensional reference models. Submitted for publication.