Eurographics Symposium on Parallel Graphics and Visualization (2008)
J. Favre, K. - L. Ma, and D. Weiskopf (Editors)

Time-Critical Distributed Visualization with Fault Tolerance

Jinzhu Gao[†] Huadong Liu[‡] Jian Huang[§] Micah Beck[¶] Qishi Wu[‖] Terry Moore[**] James Kohl[††]

Abstract

It is often desirable or necessary to perform scientific visualization in geographically remote locations, away from the centralized data storage systems that hold massive amounts of scientific results. The larger such scientific datasets are, the less practical it is to move these datasets to remote locations for collaborators. In such scenarios, efficient remote visualization solutions can be crucial. Yet the use of distributed or heterogeneous computing resources raises several challenges for large-scale data visualization. Algorithms must be robust and incorporate advanced load balancing and scheduling techniques. In this paper, we propose a time-critical remote visualization system that can be deployed over distributed and heterogeneous computing resources. We introduce an "importance" metric to measure the need for processing each data partition based on its degree of contribution to the final visual image. Factors contributing to this metric include specific application requirements, value distributions inside the data partition, and viewing parameters. We incorporate "visibility" in our measurement as well so that empty or invisible blocks will not be processed. Guided by the data blocks' importance values, our dynamic scheduling scheme determines the rendering priority for each visible block. That is, more important blocks will be rendered first. In time-critical scenarios, our scheduling algorithm also dynamically reduces the level-of-detail for the less important regions so that visualization can be finished in a user-specified time limit with highest possible image quality. This system enables interactive sharing of visualization results. To evaluate the performance of this system, we present a case study using a 250 Gigabyte dataset on 170 distributed processors.

Categories and Subject Descriptors (according to ACM CCS): I.3.2 [Graphics Systems]: Distributed/network graphics I.3.6 [Methodology and Techniques]: Graphics data structures and data types

1. Introduction

Next-generation scientific applications are increasingly supported by advances in supercomputing technologies, resulting in the generation of vast amounts of scientific data – terabytes to petabytes in near future – which must be stored, transferred, visualized and analyzed by potentially geographically distributed teams of scientists. One example of such large-scale data-based science is the Terascale Supernova Initiative (TSI) project [TSI], which is a collaborative effort at Oak Ridge National Laboratory (ORNL) with several universities across the United States.

The immense scale of datasets generated by projects like TSI requires exploitation of some form of parallelism to effectively and efficiently analyze and visualize the data. Parallel clusters and supercomputers are commonly applied to perform these data analysis and visualization tasks. Often these systems are homogeneous, with all processors assumed to offer a consistent and similar level of performance.

Recently, the visualization community has begun exploring solutions that pool globally distributed and heterogeneous computing resources to enable large-scale visualization [DHB*03, GHJ*05]. The processors in these systems are deployed on the Internet at large with a diverse set of computing and throughput capabilities. Due to the dynamic nature of these systems and their associated sub-networks, these processors are not guaranteed to be highly available; hosts may exhibit arbitrary levels of performance and connections may fail unexpectedly.

† University of Minnesota, Morris, gaoj@morris.umn.edu
‡ University of Tennessee, hliu@cs.utk.edu
§ University of Tennessee, huangj@cs.utk.edu
¶ University of Tennessee, mbeck@cs.utk.edu
‖ University of Memphis, qishiwu@memphis.edu
** University of Tennessee, tmoore@cs.utk.edu
†† Oak Ridge National Laboratory, kohlja@ornl.gov

A different approach must be applied to this harsh environment that addresses performance as well as robustness. In this paper, we propose a fault-tolerant time-critical visualization system that tolerates heterogeneity of processors and the perils of wide-area distribution across the Internet. To share and visualize a large dataset among a team of geographically distributed collaborators, our system leaves the primary data source intact and in situ, without duplicating it onto each local machine. Instead, the dataset is partitioned into data blocks and uploaded to sets of distributed and heterogeneous processing units. Besides leveraging data management techniques that minimize the amount of data to be processed [GHJ*05], we also define an *importance* metric to prioritize the processing of data blocks based on their contributions to the final image. This metric is critical to ensuring that visualization deadlines, e.g. for interactive exploration as specified by end users, can be met with the highest possible rendering quality. To achieve fault-tolerance, including dynamic adaptation to changing computing environments, we design a quality-driven "back-off" scheme to schedule the visualization of distributed data blocks and trade lower rendering quality when necessary .

To illustrate the utility of this scheme, and demonstrate the expected performance in real-world scenarios, a case study was performed. A TSI simulation dataset of 250 GigaBytes (GB) was rendered using 170 Internet nodes deployed throughout North America and Europe.

The remainder of this paper is organized as follows. Section 2 describes the previous work on distributed data management, time-critical visualization and scheduling schemes for parallel distributed computing. Section 3 overviews the system design, and Section 4 discusses importance metric and quality-driven back-off scheme. The case study of experimental results is presented in Section 5, with a summary of contributions and future directions in Section 6.

2. Background

The main focus of time-critical visualization is to guarantee a user-specified level of rendering rate while providing the highest affordable visual quality, no matter how large a dataset is. Since dataset sizes are increasing at an unprecedented space, the limits of the current technology are being strongly tested. Most approaches resort to level-of-detail (LoD) techniques for multi-resolution selections [CMPS97, ECS00, LHJ03] in which time critical visualization is implemented as a trade-off between image quality and rendering overhead. The trade-off is determined dynamically and each data partition is treated independently. In 1993, Funkhouser and Sequin [FS93] first proposed to adaptively render objects at multiple levels of detail in order to generate the image in highest possible quality within the target frame time. Li and Shen in [LS01] described a general criterion for deciding which LoD to use for each data partition. Their criteria is based on the maximum opacity, the distance to the eye, projection area of a partition, and the gaze distance. The threshold of variance is determined automatically using fuzzy logic. Pascucci *et al.* [PLF*03] proposed to reduce the latency by progressively streaming data according to available network resources and desired visual fidelity.

Many researchers, such as Anupam *et al.* [ABSS94], proposed to build a low-cost distributed scientific visualization environment to support collaborative research. However, in the context of time-critical visualization, the large-scale and parallel use of heterogeneous processing units presented here has rarely been investigated before. Due to the time-varying nature of processor performance, time budget allocation methods, as described in [LS02], may be problematic. In addition, most existing methods for selecting LoD levels have focused on using data centric metrics, primarily the variance or Mean Squared Error (MSE) of voxel values, the area of projection on the image plane of each data partition, and some others. View dependent heuristics, such as visibility, or transfer function based heuristics, such as transparency, have not been widely explored.

The fundamental data structures used in time-critical methods are mostly derived from tree-based multi-resolution hierarchy. Such data structures in fact are very general and have a number of other applications in the field. For example, Time-Space Partitioning (TSP) tree [SCM99] is now widely used in the field. Its main skeleton is a complete octree in the spatial domain, or a Space-Partitioning Tree (SPT). On each SPT node, a binary tree is created to capture the recursive bisection of the time span. After its introduction, the TSP tree is further enhanced with wavelet compression (WTSP) [WS04] and other information including visibility as well as value histogram [GHJ*05]. In the work presented here, we use the enhanced TSP (ETSP) tree [GHJ*05] as the core data structure for its effectiveness, flexibility and convenience of managing time-varying data.

Public-resource computing systems [And04] also aim to aggregate a large number of geographically distributed, administratively independent processors to solve large-scale scientific problems. In these systems, volunteer processors typically download a client program and request work units from a central server. The work unit is usually computation intensive. For example, size of the SETI@home [ACK*02] work unit is small enough to be downloaded in a few minutes but could keep a typical processor busy for about a day. Visualization applications do not typically run for that long, however. While many distributed infrastructures could be used to leverage distributed heterogeneous processors, we chose to test our method using the logistical network (LN) infrastructure [BMP02]. On LN infrastructure, we can maintain the datasets directly on distributed heterogeneous servers, like the work in [GHJ*05], but also compute visualization operations in situ on such "network-resident" data on-the-fly. In fact, a very recent work [LBH06] has already demonstrated the feasibility of this approach using embarrassingly paral-

lel visualization as the targeted application. In this work, we use this same system infrastructure to explore parallel time-critical visualization on distributed heterogeneous processors, and investigate the potential of implementing parallel algorithms with much greater interdependence among subdivided rendering tasks.

3. System Overview

The goal of this work is to achieve parallel efficiency and fault tolerance in time-critical visualization over distributed and heterogeneous computing resources. As illustrated in Figure 1, we propose a visualization system that facilitates convenient and effective collaborations among geographically distributed users.

Consider the scenario where a user discovers an interesting result in a large scientific dataset and wishes to share these results with a remote collaborator. The user saves the parameters specifying the specific visualization process and the dataset that was used, in an XML file ("visSpec"). This XML file is then sent (for example, via email) to the remote collaborator to enhance discussions and further joint examinations. After receiving the XML file, the collaborator may tweak any desired parameters and then request the corresponding visualization to be executed, with the visual results to be rendered within a pre-specified time limit. Suppose in this case the collaborator desires only a high-level overview of the result, and so sets a very short turn-around time frame for the visualization process. The XML file is engaged into the collaborative visualization system, pulling the minimal necessary data across for processing, and the coarse results are presented as desired within the given time limit. Next, based on this information, the collaborator resubmits the visualization job with a longer time frame, to obtain more detailed and higher quality visual results, and then arranges a teleconference to discuss the results with the original user, while they both examine full-resolution images.

The proposed system to realize this vision consists of two main portions: data preparation and the runtime time-critical visualization algorithm, respectively. Data preparation, depicted as steps *A* and *B* in Figure 1, is a one-time process that includes generating the dataset, partitioning it into blocks and constructing a multi-resolution hierarchy. This hierarchy is represented by an Enhanced Time-Space Partitioning (ETSP) tree data structure [GHJ*05]. The resulting data blocks are then uploaded onto a large number of distributed processing units, with redundancy for fault-tolerance, such as provided by the Internet Backplane Protocol (IBP) [BMP02]. Each processing unit, referred to as a *depot*, has both local storage and computing resources. To reduce the overhead of data replication, redundant copies are progressively stored at additional depots, using a multi-source copying scheme after the first copy of the dataset has been uploaded. Once the dataset is in place, users at different sites can access and visualize the same dataset without the need for replicating the entire dataset on their local machines.

The runtime time-critical visualization algorithm involves evaluation of the *importance* metric, as well as handling dynamic scheduling. The importance metric is defined by view-dependent, data-dependent and application-dependent factors, which together determine the rendering order and level-of-detail (LoD) of a block. The dynamic scheduling algorithm monitors the performance offered by specific distributed depots, and then budgets the computing resources for the rendering task of each block based on its importance measurement. Fault-tolerance is provided for two distinct scenarios: (i) deteriorated performance of a depot prevents it from completing a scheduled rendering task; and, (ii) a depot is detected as a faulty processing unit.

When the time limit specified by a user is insufficient to render all visible data blocks at the highest LoD, the algorithm, referred to as a *quality-driven back-off scheme*, selects lower resolutions for the less important regions. Upon completion of the visualization process, the imagery results of the rendered data blocks are transmitted to the user for compositing. This entire process for time-critical visualization is illustrated as steps *C* and *D* in Figure 1.

4. Our Method

In this section, we define a general importance metric by considering view-dependent, value-dependent, and application-dependent factors, and describe a dynamic scheduling scheme with fault-tolerance.

4.1. Importance Metric

We use a LoD selection algorithm to meet the user's request when the visualization cannot be completed within the user-specified time limit. In this case, the image quality highly depends on the selection scheme. Previously, Li and Shen introduced the concept of "importance" [LS02]. They assign different time budget for different regions based on their importance. Although their scheme guarantees that more important regions would be rendered in higher resolution, the importance metric it employed does not consider factors such as visibility. In their scheme, the opaqueness of a region is represented by the highest opacity of all voxels inside the region, which may lead to inaccurate importance measurements.

In this paper, we define the importance of a block based on its contribution to the final image. Different from the time budget allocation scheme in [LS02], we evaluate the importance for each block and use this information in a run-time dynamic scheduling scheme to prioritize the rendering order of LoD blocks. The contribution of a block to the final image depends on a number of factors, which can be classified into three categories: application-dependent, value-dependent and view-dependent. In our system, the importance value I of a block is calculated by $I = w_{app} * I_{app} +$

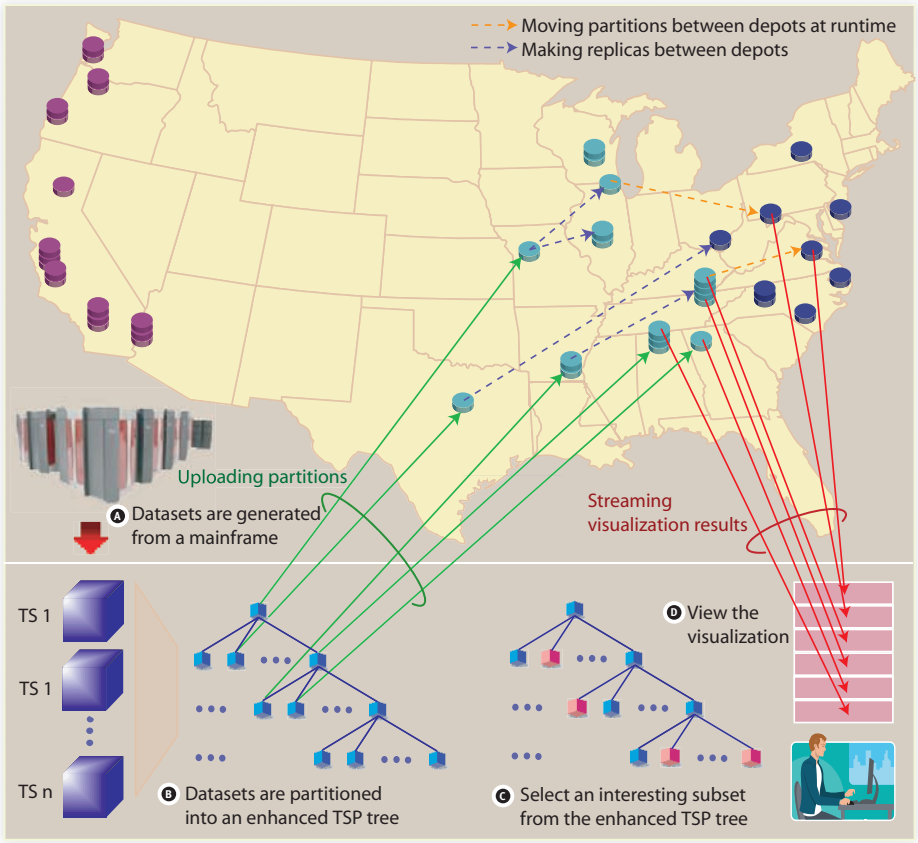

Figure 1: *The system overview. The depots are grouped by geographical locations with different colors.*

$w_{val} * I_{val} + w_{view} * I_{view}$, where I_{app}, I_{val}, and I_{view} denote the importance contribution made by application-dependent, value-dependent, and view-dependent factors, respectively, and w's are the weight coefficients for the corresponding importance components. Note that all weight coefficients mentioned in this paper are provided so that users have more control on the importance calculation. Generally we could set all those coefficients to 1.0, which implies that all factors are equally important.

4.1.1. Application-Dependent

The importance of a block may depend on the underlying applications. For example, in time-critical applications, we choose the highest possible resolution for a region as allowed by the specified time limit. To satisfy this requirement, we calculate I_{app} as $Height_{root} - Height_{node}$, where $Height_{node}$ is the height of the corresponding ETSP tree node for the block and $Height_{root}$ is the height of the ETSP tree.

4.1.2. Value-Dependent

The importance of a block may also depend on the value distribution inside a block as well as the transfer function a user chooses. Generally, a more opaque block is assigned a higher importance value and a block with high variation is more important than a homogenous block.

In our system, we calculate the opaqueness V_{opa} and the value variance V_{var} of a block from its value histogram and the given transfer function. For a low resolution block, its spatial error V_{serr} also affects its importance. A higher error often indicates a lower importance. We define I_{val} as $w_{opa} * V_{opa} + w_{var} * V_{var} + w_{serr} * (1 - V_{serr})$, where w's are the weight coefficients for the corresponding components.

4.1.3. View-Dependent

The eye position also determines the importance of a block, which is proportional to its distance to the eye. Note that an invisible block does not have an importance value and should be excluded from the visualization process. We parameterize the view-dependent factor with the sequential index $ID_{traversal}$ during the front-to-back traversal and the total number of blocks N_{blocks}, as $I_{view} = 1 - ID_{traversal}/N_{blocks}$. By including the traversal order of blocks in the importance calculation, we are able to assign a higher importance value to a closer block.

We wish to identify and cull away empty and invisible blocks to avoid evaluating the importance for those blocks. Proposed in [GHSK03], the visibility culling scheme based on Plenoptic Opacity Functions (POF) is still used in our system because of its effectiveness and scalability. Thus, during preprocessing stage, we need to pre-compute and encode the opacity information for each block in a plenoptic opacity function.

4.1.4. Data Structure

For fast runtime importance measurements, we pre-compute and store the parameters discussed above in an enhanced TSP (ETSP) tree as in [GHJ*05]. ETSP tree extends TSP trees by incorporating more metrics, such as transfer function dependent opacity and visibility, to support dynamic discovery of voxel blocks that are un-occluded and non-transparent, at adaptive levels of detail. When a user issues a visualization request, the system first performs visibility test to cull away empty and invisible blocks, and then traverses the tree in a front-to-back order to update the importance value for each tree node. Upon the completion of the traversal, the system produces a list of rendering tasks for visible LoD blocks sorted by the importance values. This list will be sent to the scheduler, which determines the rendering order of blocks.

4.2. Dynamic Fault-Tolerant Load Balancing

Our run-time algorithm runs in the common master-worker model. The distributed and heterogeneous depots are the *worker* processors that perform rendering tasks. The client's local machine serves as a *master* processor that schedules the entire parallel run and composites the final image for display.

In this section, we present a dynamic scheduling algorithm, referred to as *scheduler*, for the master processor to achieve fault-tolerance and load balancing. The scheduler dynamically collects performance measurements for each depot on the fly. The main principle of the scheduler is to assign more tasks to faster depots and avoid being stalled by slow or faulty depots. The scheduler is responsible for assigning data blocks to a set of strategically selected depots and scheduling rendering tasks.

4.2.1. Adaptive Scheduling of Rendering Tasks

The scheduler maintains two generic data structures: (i) a dynamically ranked pool of depots, and (ii) a two-level priority queue of tasks. We rank the depots in the order of their performances estimated by the rendering time for a task.

This performance measurement is updated adaptively by computing a smoothed average of depot performances, $\tau^i_{n+1} = \alpha t_n + (1-\alpha)\tau^i_n$, where τ^i_n is the average of the previous performance measurements of depot i, and t_n is the latest measurement. We could adjust the value of α to assign different weights to the historical measurements.

A two-level priority queue manages 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. In the HPQ, each task has two keys, with the primary key being its importance value and the secondary key being the optimal task processing time, which is the minimum processing time of all depots that render the same data block. In the LPQ, each task is keyed by the estimated time left for completion. We sort tasks in both the HPQ and the LPQ using their keys in a decreasing order.

Based on the adaptively ranked depot pool and task queue, we design a task assignment scheme to avoid competitions between depots with different levels of performances. Initially the LPQ is empty while the HPQ contains all rendering tasks. When a distributed visualization session starts, rendering tasks in the HPQ are sequentially assigned to available depots and demoted to the LPQ. A depot can perform a task only if it has the required data stored locally. Each time a depot finishes its assigned task, it will be immediately assigned the first task in the HPQ it can perform. In this way, faster depots will be assigned more tasks as slower depots do not compete for tasks with them. If the depot is not capable of performing any task in the HPQ , the first task in the LPQ it can perform is considered. This is the slowest unfinished task that the depot can help with. In this way, multiple depots work in parallel on the same unfinished task to ensure the best delivery.

Figure 2 shows an example of the task assignment. In this example, task T_i, T_j and T_k have importance value 7, 6 and 6 respectively. Performance of each depot (time to process a task) is listed after the depot ID. An arrow in the figure represents that the depot has the data block required by a task. In this example, If D_1 becomes available, task T_j will be assigned to it because T_k can be performed by a even faster depot D_3; When D_2 becomes available, the scheduler has to check the LPQ for tasks because D_2 cannot work on any of the tasks in the HPQ; When D_3 becomes available, it will be assigned T_i which has a higher importance value than T_k.

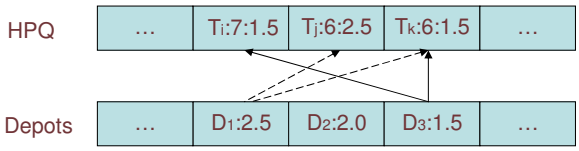

Figure 2: *An example of task assignment.*

4.2.2. Dynamic Scheduling of Data Movement

At any particular time, some data blocks might only reside on a set of depots that are slow or heavily loaded. In that case, faster depots cannot help because they do not have a copy of the required block to work on. A natural thought is to move data blocks from slower depots to faster depots at runtime. In order to make sure that time spent on data movement does not exceed the benefit we gain from migrating the

task, bandwidth information between depots needs to be acquired. Instead of injecting extra testing traffic into the network, we use a multi-source partial download scheme with deadline for data movement between depots.

The scheduling of data movement works in parallel with the scheduling of rendering tasks. Ideally, the shortest overall execution time occurs when all depots finish roughly at the same time. For each depot, once the number of tasks it can do in the HPQ is less than its proportion of all unassigned tasks according to its performance, the scheduler tries to assign a task to this fast depot before it becomes idle. The scheduler starts from the first task in the HPQ, which has the highest importance but potentially least likelihood to be done by a faster depot. In this way, a more important task with the higher priority will always be assigned to a faster processor. To avoid moving tasks out of the same set of slow depots, the task is transferred only if all depots that can perform this task have sufficient work to remain busy. The data movement, or task migration, is done by applying the multi-source partial download with deadline as described in [LBH06].

4.2.3. Dealing with Faults

Once a visualization session is launched, it is concurrently spawned on hundreds of distributed depots that are usually built from commodity PCs. Since those depots cannot be assumed to be fully reliable, we implement a fault-tolerance mechanism in our scheduler. Although it may catch transient faults (e.g. memory bit errors) or cheating processors (e.g. to minimize resource expenditure), its primary goal is to discover and remedy persistent errors (e.g. software configuration errors and hardware failures) in addition to perceivable runtime errors.

To deal with faults indicated by computation timeout or network connection failure, we promote the failed task in the LPQ back to the HPQ so that other depots can take it over. To deal with faults that produce incorrect computation result but with a valid return code, we employ a majority voting scheme. A checking task is assigned to all participating processors at the beginning of each visualization session. The checking task does the same visualization operation on a data block that is available on every processor. When the results come back, they are compared and the majority is considered as the correct result. Processors that return incorrect results are disabled in the rest of the computation. Performing the checking task at the beginning helps to discover a faulty depot earlier. However, it might miss the chance to catch a depot that fails in the middle, which can be resolved with more frequent checkings during the session.

4.2.4. Quality-Driven Back-off

For time-critical visualizations, system response time is more important than image quality. In some situations, image quality must be sacrificed in order to obtain guaranteed delivery of visualization results by a deadline. In a heterogeneous distributed environment, it is hard to predict time-varying computing performance. And resource reservation is not an option to guarantee the delivery of quality images within the time limit. To meet the user-specified time limit, our system dynamically measures the system throughput and evaluates whether it is necessary to perform *back-off*, that is, replacing several tasks that would operate on high resolution data with one task that operates on lower resolution data.

Since the tasks have similar workload, we characterize system throughput as the number of tasks finished per second. The number is updated adaptively in the same way that we measure the performance of a depot in Section 4.2.1. Using the system throughput, the user-specified time limit is compared periodically with the estimated time to finish all the required tasks. If the deadline cannot be met under the current system throughput, the back-off module in the scheduler is invoked. We mark less importance tasks and replace them with a smaller number of lower resolution tasks.

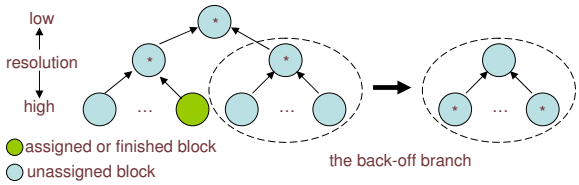

Figure 3: *Backoff branch selection. Tasks marked with a '*' will not be rendered.*

In order to ensure that tasks with high importance are rendered with the highest possible resolution, back-off tasks are selected from the tail of the HPQ. To ensure back-off efficiency, as shown in Figure 3, we only select branches of the multi-resolution tree in which all leaf tasks have not been assigned for rendering. This marking process is dynamic, i.e. if the system throughput improves, we can de-mark tasks; and if the system throughput decreases, we can mark more tasks.

Since the data blocks are replicated on several depots, if we have to choose from several back-off branches that have the same importance, we select the branch that can be done faster. For example, suppose $\{T_x, ..., T_y\}$ can be reduced to a lower resolution task T_m and $\{T_{x'}, ..., T_{y'}\}$ can be reduced to a lower resolution task T_n. If T_m can finish earlier, we choose T_m to replace $\{T_x, ..., T_y\}$.

5. Experimental Results

5.1. Test Environment

To demonstrate the effectiveness of our system, we tested it on 160 depots from the PlanetLab project [Pla] and another 10 from the National Logistical Networking Testbed (NLNT). Visualization operations are loaded and executed in a sandbox to ensure security and stability of the depot. Although these depots are server-class machines, they are shared among a large community. PlanetLab nodes are even

virtualized as “slices" to enable large-scale sharing. Loads on these nodes differ dramatically and vary over time.

We used a 128 time-step subset (250GB in total) of a simulation dataset produced by the Terascale Supernova Initiative (TSI) project in our experiments. The dataset is of $864 \times 864 \times 864$ spatial resolution. We generated and partitioned the multiresolution data, and produced 3160 data blocks ($64 \times 64 \times 64$ each), totaling 2.9GB per timestep. The $2.9 \times 128 = 371$ GB data is uploaded onto 170 depots with 3-way replication, thus over 1TB of data was stored. The time to upload the first copy of data is comparable to the time to move data between two locations. The remaining $k-1$ copies are replicated in parallel using multisource multicasts. The preprocessing including data partition, construction of the tree structure, and data uploading and replication usually completes between 10 to 20 hours.

5.2. Testbed Deployment

The foundation of the infrastructure used to support our distributed visualization system is Logistical Networking (LN), and in particular the Internet Backplane Protocol (IBP), which provides a mechanism for sharing storage and compute resources within a collaborating community. The design of IBP parallels the design of the Internet Protocol (IP), adhering to the End-to-End Principles in order to obtain scalability of deployment.

Like IP, the service IBP defines is both generic and limited, which has important implications for the architecture of LN. In terms of technology substrate, building on a highly generic protocol means that LN can incorporate heterogeneous underlying storage and computational technologies. In the case of storage, IBP provides a model that is somewhat more abstract than block-serving protocols such as iSCSI, and has more in common with the ANSI T10/Object Storage Device standard. However, IBP is more adapted to being used in a scalable, wide area community than those standard protocols. In terms of computation, IBP provides a model of remote execution which is generic in that it allows the application of arbitrary operations.

5.3. Performance Evaluation

To evaluate the parallel utilization of our system, we performed a test on 170 depots distributed in North America from PlanetLab and the NLNT. For testing purpose, we simply set all the weight coefficients to 1.0 when computing the importance value for each block. It took about 51 seconds on average to process four time steps of the TSI dataset and generate an 800×800 image for each time step using software raycasting. Knowing that it is not a rigorous comparison, but only to provide context, the same volume rendering took 62 minutes on a dedicated node with 2.2GHz P4 CPU, 512KB cache and 2GB RAM. The performance achieved with these 170 shared, distributed heterogeneous depots roughly equals that of a dedicated 80-node cluster with nodes similar to the test system described above, assuming 90% parallel utilization on the cluster.

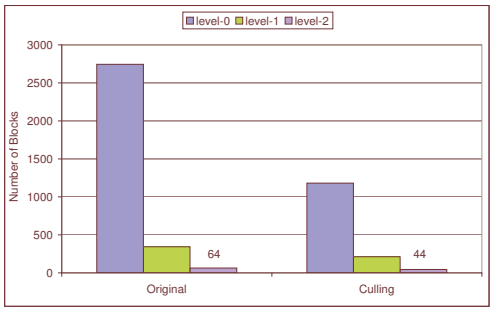

Figure 4: *The number of original blocks and visible blocks after culling at resolution level 0, 1 and 2 of a TSI dataset.*

Our visibility culling scheme is very effective in culling away empty or invisible blocks. The algorithm consistently culls away 50%-60% blocks at each of 128 time steps, thus allows our time-critical visualization algorithm to be focused on only visible blocks. As an example, in Figure 4, we plot the number of original blocks and visible blocks after culling at resolution level 0, 1 and 2 of the TSI dataset at the 31st time step.

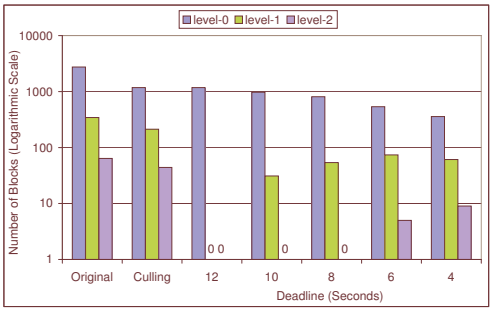

Figure 5: *Logarithmic plot of the number of blocks rendered at different resolution level with different running deadline.*

User-specified rendering time requirement decides the number of blocks at each resolution level to be rendered. Taking the rendering of visible blocks of the TSI dataset at the 31st time step as an example, we plot the actual number of blocks rendered at each solution level with different deadline in Figure 5. To present the results in the logarithmic scale, we re-plot Figure 4 in the leftmost two columns of Figure 5. As shown in the figure, a longer deadline allows more blocks with high resolution (e.g. level 0) to be rendered. Conversely, a shorter deadline forces the scheduler to select lower resolution blocks (e.g. level 2).

We performed a simple experiment to evaluate our time-critical and fault tolerance scheme. Initially 8 depots were used to render the data at the 31st time step and the deadline

was set to be 31 seconds. all 1181 level-0 blocks can be rendered if all 8 depots were available. After we disabled one depot, we were still able to finish the rendering. However, to meet the deadline, only 1025 level-0 blocks can be rendered and the remaining 156 level-0 blocks have to be replaced by 32 level-1 blocks. If we disabled one more depot, only 876 level-0 blocks can be rendered and the remaining 305 level-0 blocks were replaced by 52 level-1 blocks. Currently we are using a fault tolerance scheme that solely depends on replication. If all replicas of a particular block fail, that block cannot be visualized. In the future, we plan to incorporate erasure codes to reconstruct data blocks. It will server as another level of fault tolerance that takes advantage of the computation capability the NFU provides.

6. Conclusion and Future Work

In this work, we explored the possibility to use hundreds of geographically distributed, free, unreserved, heterogeneous processors for time-critical visualization. Our method achieves not only parallel efficiency but also reliable fault-tolerance. In total, 250 GB of a real-world simulation dataset has been visualized using our system and the best affordable visualization results were delivered to users in the given time limit. Our results have demonstrated a great potential to use distributed heterogeneous processors as a fundamental computing platform. In the future, we plan to further develop methods to support wide-area collaborations by large groups of users on a common testbed.

References

[ABSS94] ANUPAM V., BAJAJ C., SCHIKORE D., SCHIKORE M.: Distributed and collaborative visualization. *Computer 27*, 7 (1994), 37–43.

[ACK*02] ANDERSON D. P., COBB J., KORPELA E., LEBOFSKY M., WERTHIMER D.: Seti@home: an experiment in public-resource computing. *Commun. ACM 45*, 11 (2002), 56–61.

[And04] ANDERSON D. P.: Boinc: A system for public-resource computing and storage. In *GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 4–10.

[BMP02] BECK M., MOORE T., PLANK J.: An end-to-end approach to globally scalable network storage. In *ACM SIGCOMM* (2002).

[CMPS97] CIGNONI P., MONTANI C., PUPPO E., SCOPIGNO R.: Multiresolution representation and visualization of volume data. *IEEE Transaction on Visualization and Computer Graphics 3*, 4 (1997), 352–369.

[DHB*03] DING J., HUANG J., BECK M., LIU S., MOORE T., SOLTESZ S.: Remote visualization by browsing image based databases with logistical networking. In *Proceedings of Supercomputing* (2003).

[ECS00] ELLSWORTH D., CHIANG L., SHEN H.-W.: Accelerating Time-Varying Hardware Volume Rendering using TSP Trees and Color-based Error Metrics. In *Proceedings of 2000 Symposium on Volume Visualization* (2000), pp. 119–128.

[FS93] FUNKHOUSER T., SEQUIN C.: Adaptive Display Algorithms for Interactive Frame Rate During Visualization of Virtual Environment. In *Proceedings of SIGGRAPH 93* (1993), pp. 247–254.

[GHJ*05] GAO J., HUANG J., JOHNSON C. R., ATCHLEY S., KOHL J. A.: Distributed Data Management for Large Volume Visualization. In *Proceedings of IEEE Visualization '05* (2005), pp. 183–189.

[GHSK03] GAO J., HUANG J., SHEN H.-W., KOHL J. A.: Visibility Culling Using Plenoptic Opacity Functions for Large Data Visualization. In *Proceedings of IEEE Visualization '03* (2003), pp. 341–348.

[LBH06] LIU H., BECK M., HUANG J.: Dynamic co-scheduling of distributed computation and replication. In *CCGrid '06: Proceedings of the 6th International Symposium on Cluster Computing and the Grid* (Singapore, 2006).

[LHJ03] LAMAR E. C., HAMANN B., JOY K. I.: *Efficient Error Calculation for Multiresolution Texture-Based Volume Visualization*. Springer-Verlag, Heidelberg, Germany, 2003, pp. 51–62.

[LS01] LI X., SHEN H.-W.: Adaptive Volume Rendering using Fuzzy Logic Control. In *Proceedings of Joint Eurographics-IEEE TCVG Symposium on Visualization* (2001).

[LS02] LI X., SHEN H.-W.: Time-Critical Multiresolution Volume Rendering using 3D Texture Mapping Hardware. In *IEEE/ACM 2002 Symposium on Volume Visualization and Graphics* (2002), pp. 29–36.

[Pla] PlanetLab. http://www.planet-lab.org/.

[PLF*03] PASCUCCI V., LANEY D. E., FRANK R. J., SCORZELLI G., LINSEN L., HAMANN B., GYGI F.: Real-time Monitoring of Large Scientific Simulations. In *Proceedings of the 2003 ACM Symposium on Applied Computing* (2003), pp. 194–198.

[SCM99] SHEN H.-W., CHIANG L.-J., MA K.-L.: A fast volume rendering algorithm for time-varying fields using a time-space partitioning (tsp) tree. In *IEEE Visualization'99* (1999), pp. 371–377.

[TSI] Terascale supernova initiative. http://www.phy.ornl.gov/tsi.

[WS04] WANG C., SHEN H.-W.: A framework for rendering large time-varying data using wavelet-based time-space partitioning (wtsp) tree. In *Technical Report No. OSU-CISRC-1/04-TR05, Department of Computer Science and Engineering, The Ohio State University* (2004).