This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

Self-Adaptive Configuration of Visualization Pipeline Over Wide-Area Networks

Qishi Wu, Jinzhu Gao, Mengxia Zhu, Nageswara S.V. Rao, Jian Huang, S. Sitharama Iyengar

Abstract

Next-generation scientific applications require the capabilities to visualize large archival datasets or on-going computer simulations of physical and other phenomena over wide-area network connections. To minimize the latency in interactive visualizations across wide-area networks, we propose an approach that adaptively decomposes and maps the visualization pipeline onto a set of strategically selected network nodes. This scheme is realized by grouping the modules that implement visualization and networking subtasks, and mapping them onto computing nodes with possibly disparate computing capabilities and network connections. Using estimates for communication and processing times of subtasks, we present a polynomial-time algorithm to compute a decomposition and mapping to achieve minimum end-to-end delay of the visualization pipeline. We present experimental results using geographically distributed deployments to demonstrate the effectiveness of this method in visualizing datasets from three application domains.

Index Terms

Distributed computing, remote visualization, visualization pipeline, bandwidth measurement, network mapping.

I. Introduction

THE capability to remotely visualize datasets and on-going computations over wide-area networks is considered a critical enabling technology to support large-scale computational science applications, particularly when large datasets or computations are involved [1]. Visualization systems of different types and scales for these applications have been the focus of research for many years [2]. In general, a remote visualization system forms a pipeline consisting of a server at one end with the dataset, and a client at the other end with rendering and display capabilities. In between, zero or more network nodes perform a variety of intermediate processing and caching operations. Due to the computational, functional, and bandwidth limitations at the client end, it is usually impractical to transfer or replicate entire large datasets at the client. Instead, a common design goal is to achieve interactive visualizations over wide-area networks by optimally matching the modules of a visualization pipeline with the resources at the network nodes to minimize data transfer and processing times.

Wu is with Dept of Computer Science, University of Memphis, Memphis, TN 38152, Email: qishiwu@memphis.edu. Gao is with Division of Science and Mathematics, University of Minnesota, Morris, MN 56267, Email: gaoj@morris.umn.edu. Zhu is with Dept of Computer Science, Southern Illinois University at Carbondale, IL 62901, Email: mengxia@cs.siu.edu. Rao is with Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831, Email: raons@ornl.gov. Huang is with Dept of Computer Science, University of Tennessee, Knoxville, TN 37996, Email: huangj@cs.utk.edu. Iyengar is with Dept of Computer Science, Louisiana State University, Baton Rouge, LA 70803, Email: iyengar@bit.csc.lsu.edu. Please send all correspondence to Wu.

Digital Object Indentifier 10.1109/TC.2007.70777 0018-9340/$25.00 © 2007 IEEE

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

In the past, the visualization field has focused on extending individual algorithms to network environments with various optimization techniques such as compression [3], [4], [5] or latency-hiding schemes [6]. Many current systems are based on statically configuring and mapping the visualization pipeline onto available network nodes, often using ad-hoc approaches. In this paper, we propose a systematic framework for remote visualization systems to self-adapt according to visualization needs and time-varying network and node conditions. The need for adaptive optimization and configuration of a visualization pipeline arises for the following reasons: (i) scientific computing communities have demonstrated the potential of pooling in globally distributed users to achieve unprecedented data collections, visualizations, simulations, and analysis; (ii) system resources including supercomputers, data repositories, computing facilities, network infrastructures, storage devices, and display units have been increasingly deployed around the globe; (iii) resources are typically connected via the Internet or a dedicated network to many users, which could make their availability, utilization, capacity, and performance very dynamic; and (iv) users in scientific, medical, and engineering areas require different visualization modalities specific to their domains with different visualization parameters. A self-adaptive visualization system is needed to optimally utilize the networked resources under the above diverse and dynamic environments.

We develop cost models to estimate processing times of visualization modules, including isosurface extraction and raycasting, as well as data transfer times over network connections. Using these cost models, we propose a dynamic programming solution to compute optimal pipeline configurations. Specifically, we decompose the visualization pipeline into groups of modules and map them onto network nodes to minimize the end-to-end delay. Self-adaptation and efficient reconfiguration is achieved by the low polynomial time complexity of our dynamic programming method. We employ a message-based control flow scheme for runtime adaptation in response to user requests and dynamic system conditions. A central management node maintains the information about data sources and node capabilities to facilitate the pipeline adaptation. This node maintains *Visualization Routing Table* (VRT) consisting of a sequence of network nodes that the request flows through, which is dynamically updated to maintain an optimal configuration.

The main contributions of our work include an analytical formulation of a remote visualization problem in a networked environment, the development of cost models for transport and computing times, and the design of a self-adaptive system for visualization pipeline configuration. This system is implemented

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

under a prototype called *Distributed Remote Intelligent Visualization Environment* (DRIVE) and is tested over the Internet. DRIVE consists of a number of virtual service nodes deployed over the network that work together to achieve minimal end-to-end delay through self-adaptive pipeline configuration. On a large testbed involving supercomputers, PC clusters, and workstations deployed across the United States, we show that our system computes and maintains an optimal pipeline configuration in response to user interactions and dynamic system conditions. The cost models for computation and transfer times as well as the dynamic programming method, can be easily extended to other remote visualization systems, such as Vis5D+, ParaView, ASPECT, and EnSight [7], [8], [9], [10] to optimize their wide-area network deployments. However, to achieve self-adaptation under dynamic environments, it is important to employ an inherently reconfigurable underlying system to support our message-based control mechanism.

This paper is organized as follows. In Section II, we discuss previous related work on remote visualization systems. In Section III, we first describe a visualization pipeline along with an analytical model suited for decomposition and mapping, and then present a polynomial-time solution based on dynamic programming for computing optimal decomposition and mapping. In Section IV, we present domain-specific methods for processing time estimates and link bandwidth measurements. A message-based control flow method for pipeline self-adaptation is discussed in Section VI-D. Implementation details and experimental results are provided in Section VI. We conclude our work in Section VII.

II. Background and Related Work

Remote visualization is an intriguing topic for its immense potential research impact and its imperative overarching needs of many different kinds of expertise. It requires advanced study of remote visualization algorithms, working middleware that support development of real-world remote visualization applications, in-depth understanding of various system-level factors that affect system latency, and the integration of remote visualization systems with simulation modules to form a comprehensive system of practical applicability. In this section, we will give a general survey of each of these subjects and discuss our novel contributions in optimizing a complete pipeline involving simulation as well as remote visualization with self-adaptive configuration.

There have been several works in developing remote visualization algorithms, particularly in achieving the functionality and performance of “single site” systems over wide-area networks. Many remote visualization systems aim to significantly reduce the amount of data transport by transforming the raw data into

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

an intermediate form of compact sizes. These methods commonly apply various types of compression; for example, [3], [4], [5] use highly compressed imagery data. In addition to the imagery sent across the network as in [3], [4], Bethel *et al.* [5] also transmit the depth buffer resulted from volume rendering. Besides the compression, view dependent techniques have also been widely used to cull down data communications such that only the visible portions of an entire isosurface are transported [11], [12]. This way, data communications can be further reduced by limiting the amount of data sent to the client to only when necessary. Compression and view-dependent data culling can be combined to achieve even more reductions in data communications [6]. In addition, extensions to hardware acceleration methods for remote visualization settings have also been studied and shown to be feasible [13].

At the same time, several works are underway in developing advanced middleware for remote and distributed visualization systems. Foster *et al.* [14] studied the distance visualization in widely distributed environments; major technical challenges are identified and an online collaborative system is described to reconstruct and analyze tomographic data from remote X-ray sources and electron microscopes. Grid Initiative and projects such as the Globus Toolkit [15], [16] provided toolkits and infrastructure to deploy Grid computing systems. As a middleware extension, the Grid Visualization Kernel (GVK) [17], [18] was proposed to exploit the power of the grid to provide visualization services to scientific users. GVK is able to rearrange the visualization pipeline by moving filtering, visualization, and/or rendering away from the client towards the server, according to changing network conditions. Each visualization module in GVK is executed on either the client or the server side. Besides, since large datasets require optimized network transmission to achieve the desired performance goals, optimization techniques, such as data compression, level-of-detail filtering, occlusion culling, reference and image-based rendering, are also studied.

Several visualization applications have been built using the services provided by those middlewares. Shalf and Bethal [19] examined the impact of the Grid on visualization and compared pipelines running entirely on a local PC, partially on a cluster, and totally (apart from display) on a cluster. They demonstrated that the local PC would give the best performance for small problem sizes, and thus argued that dynamic scheduling of the pipeline is required. Although their approach lacks the self-adaptive capability for task management, they suggest the need for a simulation environment in the context of Grid research. Grimstead *et al.* [20] presented a distributed, collaborative grid enabled visualization environment that supports automated resource discovery across heterogeneous machines. Running the Resource-Aware

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

Visualization Environment (RAVE) as a background process using Grid/Web services, the system allows sharing resources with other users as RAVE supports a wide range of machines, from hand-held PDAs to high-end servers with large-scale stereo, tracked displays. Many other works, including image streaming via Common Object Request Broker Architecture (CORBA) [21], remote image-based rendering on Logistical Networking [4], and problem solving environment using the grid environment [22], also have similar focused goals. A major advantage of leveraging these middleware technologies is a clean interface to distributed systems and a reasonable guarantee of achieving the needed functionality. Our scheme can be directly added as an extra middleware layer to compute and manage the optimal mappings of different stages of the visualization pipeline onto a set of distributed nodes.

Recently, many researchers have been investigating frameworks for remote and distributed data analysis and visualization. Beynon *et al.* [23] developed a filter-stream programming framework for data-intensive applications that can query, analyze and manipulate very large datasets in a distributed environment. They represented the processing structure of an application as a set of processing units, referred to as filters. A set of filters collectively performing a computation for an application forms a filter group. As they developed the problem of scheduling instances of a filter group, they were trying to seek the answer to the following question: should a new instance be created, or an existing one be reused? They experimentally investigated the impacts of instantiating multiple filter groups on performance under varying application characteristics. Ahrens *et al.* [24] presented an architectural approach to handle large-scale visualization problems with parallel data streaming. Their approach requires less memory than other visualization systems while achieving high reuse of the code. Later, Luke and Hansen [25] proposed a general framework capable of supporting multiple scenarios to partition a remote visualization system, which was tested on a local network. Compared to their work, our approach has a more general scope and a more systematic framework in addressing the performance of remote visualization systems over wide-area networks. Boier-Martin [26] presents the idea of a unifying framework that allows visual representations of information to be customized and mixed together into new ones. It is a fine-grained approach to representing data, and is better suited to accessing and rendering it over networks. Although the focus is on geometric models and 3D shape representations, many issues discussed are relevant to network-based visualization in general.

One of our goals is to design effective cost models for various visualization modules so that we could

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

more accurately predict and optimize the end-to-end delay of the entire pipeline. With a similar goal, Bowman *et al.* [27] proposed solutions to examine the prediction of processing times of a visualization algorithm, isosurface extraction, in particular, using an empirical linear model. These predictions are used to allocate the computing resources to the visualization pipeline. Their approach is heuristic without a general gauge of global optimality; furthermore, it requires manual configuration, which makes it difficult to achieve run-time reconfiguration.

Several past works in remote visualization systems have focused on designing visualization systems optimized for monolithic remote settings, which are typically set up in an initial configuration. Furthermore, in configuring remote visualization systems, current methods are often ad-hoc with limited attempts to systematically optimize the performance by taking into account node and network parameters. Even if an initial configuration is optimal, these levels of performance cannot be sustained over time under dynamic processor loads and varying network bandwidths. In particular, few current systems are capable of switching to new processing nodes at runtime in response to increases in loads at currently deployed nodes or decreases in available bandwidths to them. Our goal is to examine the component-level performance parameters of the visualization pipeline and match them to both the computing and network resources in an adaptive and optimal manner. Our system exploits the detailed information of the visualization pipeline and computing nodes and network connections to dynamically optimize the end-to-end response time. This system can be integrated into a visualization middleware to transparently achieve network and host level performance optimization.

III. Optimal Visualization Pipeline

In Section III-A, we first describe a general pipeline of a visualization system, which will form a basis for this paper. After defining an analytical model for various components in the pipeline in Section III-B, we present our overall design and an optimization method based on dynamic programming in Section III-C.

A. General Visualization Pipeline

Visualization task can be decomposed into a number of subtasks, each of which is carried out by a different module of the visualization pipeline [28]. In many scientific applications, raw data is organized in formats such as CDF [29], HDF [30] and NetCDF [31]. In general, filtering modules perform the necessary preprocessing to the data for more efficient subsequent processing. The subsequent modules then convert the filtered data into graphical primitives which will be delivered for final display.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

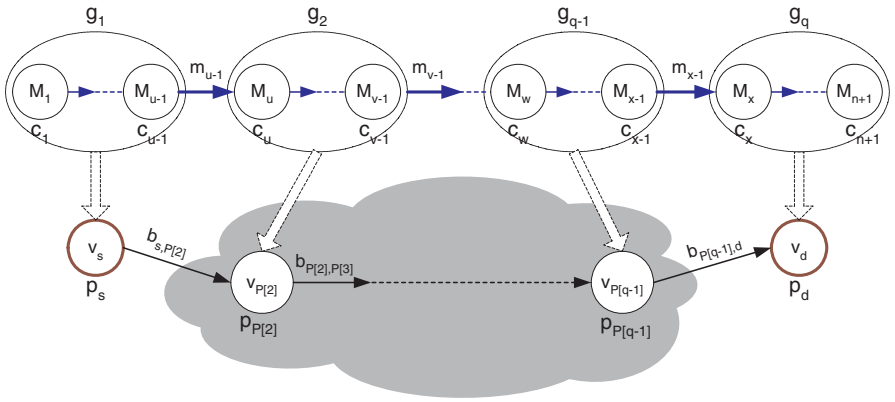


Fig. 1. Mathematical model for pipeline decomposition and network mapping.

B. Analytical Model

We now describe a mathematical model in Fig. 1 for a general visualization pipeline. The visualization pipeline consists of a sequence of $n+1$ modules, M_1, M_2, …, M_{u-1}, M_u, …, M_{v-1}, ……, M_w, …, M_{x-1}, M_x, …, M_{n+1}, where M_1 is a data source. Module $M_j, j = 2,\ldots,n+1$, performs a computational task of complexity c_j on data of size m_{j-1} received from module M_{j-1} and generates data of size m_j, which is then sent over a virtual network link to module M_{j+1} for further processing. An underlying transport network consists of $k+1$ geographically distributed computing nodes denoted by $v_1, v_2, \ldots, v_{k+1}$. Node v_i has a normalized computing power p_i[1] and is connected to its neighbor node $v_j, j \neq i$ with a network link $L_{i,j}$ of bandwidth $b_{i,j}$ and minimum link delay $d_{i,j}$. The minimum link delay is mostly contributed by the link propagation and queuing delay, and is in general much smaller than the bandwidth-constrained delay $m/b_{i,j}$ of transmitting a large message of size m. The communication network is represented by a graph $G = (V,E), |V| = k+1$, where V denotes the set of nodes and E denotes the set of virtual links. The network G may or may not be a complete graph, depending on whether the node deployment environment is the Internet or a dedicated network.

We consider a path P of q nodes from a source node v_s to a destination node v_d in the network, where $q \in [2, min(k+1, n+1)]$ and path P consists of nodes $v_{P[1]} = v_s, v_{P[2]}, \ldots, v_{P[q-1]}, v_{P[q]} = v_d$. The visualization pipeline is decomposed into q *visualization groups* denoted by $g_1, g_2, \ldots, g_q$, which are mapped one-to-one onto q nodes of path P. The data flow into a group is the one produced by the last

[1]For simplicity, we use a normalized quantity to reflect a node's overall computing power without specifying in detail its memory size, processor speed, and presence of co-processors; such details may result in different performances for both numeric and visualization computations.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

module in the upstream group; in Fig. 1, we have $m(g_1) = m_{u-1}, m(g_2) = m_{v-1}, \ldots, m(g_{q-1}) = m_{x-1}$. The client residing on the last node v_d sends control messages such as visualization parameters, filter types, visualization modes, and view parameters to one or more preceding visualization groups to support interactive operations. However, since the size of a control message is typically of the order of bytes or kilobytes, which is considerably smaller than the visualization data, we assume its transport time to be negligible.

A very important requirement in many applications of remote visualization is the high-level interactivity, which is characterized by the *end-to-end delay* given by:

$$\begin{aligned} & T_{total}(Path\ P\ of\ q\ nodes) = T_{computing} + T_{transport} \\ & = \sum_{i=1}^{q} T_{g_i} + \sum_{i=1}^{q-1} T_{L_{P[i],P[i+1]}} \\ & = \sum_{i=1}^{q} \left(\frac{1}{p_{P[i]}} \sum_{j \in g_i, j \geq 2} \left(c_j m_{j-1} \right) \right) + \sum_{i=1}^{q-1} \left(\frac{m(g_i)}{b_{P[i],P[i+1]}} \right). \end{aligned} \tag{1}$$

Our goal is to minimize the end-to-end delay, which is the time incurred on the forward links, from the source node to the destination node, to achieve the fastest response. Note that in Eq. 1, we assume the transport time between modules within each group on the same computing node to be negligible. When the number of groups $q = 2$, the system is reduced to a simple client-server setup.

C. Optimal Configuration

Based on a pipeline decomposition scheme shown in Fig. 1, we categorize the visualization modules into four types of virtual nodes: *client*, *central management* (CM), *data source* (DS), and *computing service* (CS). Each virtual node may correspond to a PC, supercomputer, cluster, rendering engine, display device, or storage system running one or more specific modules.

These nodes are connected over a communication network, typically the Internet, to form a closed visualization *control loop* as illustrated in Fig. 2. A visualization loop starts at a client that initiates a particular visualization task by sending a request containing dataset of interest, list of variable names, visualization method, and view parameters to a designated CM node. CM then determines the best pipeline configuration to accomplish the visualization task. Based on a global knowledge of the entire system as well as the available datasets, CM strategically decomposes the visualization pipeline into groups and assigns them to an appropriate CS nodes for the execution of visualization modules. The resultant pipeline decomposition and network mapping is represented as a *Visualization Routing Table* (VRT), which is delivered sequentially along the rest of the loop to establish the visualization pipeline.

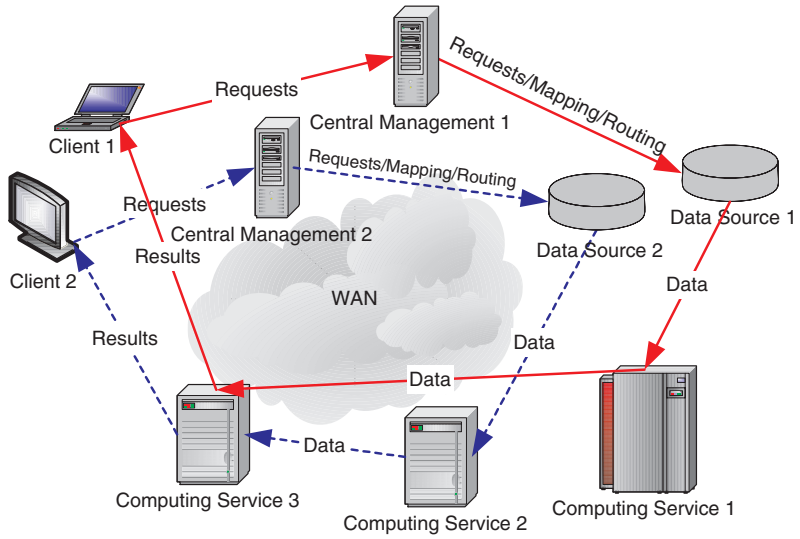


Fig. 2. DRIVE architecture: constituent elements and visualization control loops.

Since there are many possible combinations of decompositions and mappings, for the highest interactivity, it is necessary to search for the optimal combination that produces minimal end-to-end delay. We now present a dynamic programming method to achieve this goal. Let $T^j(v_i)$ denote the minimal total delay with the first j messages (namely, the first $j+1$ visualization modules) mapped to a path from the source node v_s to node v_i under consideration in G. Then, we have the following recursion leading to $T^n(v_d)$ [32], for $j = 2,\ldots,n$, $v_i \in V$:

$$T^j(v_i) = \min \left(\begin{array}{l} T^{j-1}(v_i) + \frac{c_{j+1} m_j}{p_{v_i}}, \\ \min_{u \in adj(v_i)} \left(T^{j-1}(u) + \frac{c_{j+1} m_j}{p_{v_i}} + \frac{m_j}{b_{u,v_i}} \right) \end{array} \right) \tag{2}$$

with the base conditions computed as, for $v_i \in V$, $v_i \neq v_s$:

$$T^1(v_i) = \begin{cases} \frac{c_2 m_1}{p_{v_i}} + \frac{m_1}{b_{v_s,v_i}}, & \forall e_{v_s,v_i} \in E \\ \infty, & otherwise, \end{cases} \tag{3}$$

as shown on the first column and on the first row in the 2D matrix in Fig. 3.

In Eq. 2, at each step of the recursion, $T^j(v_i)$ takes the minimum of delays of two sub-cases. In the first sub-case, we do not map the last message m_j to any network link; instead we directly place the last module M_{j+1} at node v_i itself. Therefore we only need to add the computing time of M_{j+1} on node v_i to $T^{j-1}(v_i)$, which is a sub-problem of node v_i of size $j-1$. This sub-case is represented by the direct inheritance link from its left neighbor element in the 2D matrix. In the second sub-case, the last message m_j is mapped to one of the incident network links from its neighbor nodes to node v_i. The set of neighbor

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

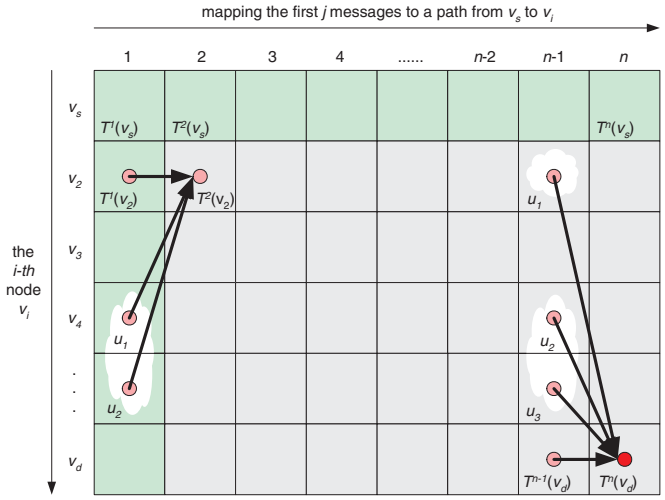


Fig. 3. Construction of 2D matrix in dynamic programming.

nodes of node v_i is enclosed in the shaded area in Fig. 3. We calculate the total delay for each mapping of an incident link of node v_i and choose the one with the minimum delay, which is then compared with the first sub-case. For each comparison step, the mapping scheme of $T^j(v_i)$ is obtained as follows: we either directly inherit the mapping scheme of $T^{j-1}(v_i)$ by simply adding module M_{j+1} to the last group, or create a separate group for module M_{j+1} and append it to the mapping scheme $T^{j-1}(u)$ of the neighbor nodes $u \in adj(v_i)$ of node v_i. The computational complexity of this core algorithm is $O(n \times |E|)$, which guarantees that our system scales well as the network size increases.

IV. Cost Models

We present in this section the cost models for both visualization computing and network transport modules to estimate the processing and communication times.

A. Processing Time Estimation

Optimization often plays an important role in the performance of a visualization technique. For instance, volume rendering could use hardware accelerations of various kinds, while software volume rendering may leverage sophisticated space leaping methods. Similarly, iso-surface extraction methods are also able to leverage a number of advanced data structures to expedite searching process. The performance gained by employing such choices could be very significant at times. For our study of remote visualization with interactive operations, however, these acceleration methods require non-trivial pre-processing and the resultant storage overheads, in addition to already high costs of large datasets. We consider the traditional

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

coarse-grained mechanism, which widely used by both parallel and out-of-core communities. We partition spatial volumes into blocks of equal size and perform space culling on a block basis. Accordingly, we estimate processing times using a block-by-block procedure for volume visualization, including isosurface extraction and volume rendering.

Herein, we model the overall computing time incurred by an entire volume T_v as:

$$T_v = \sum_{i=1}^{N} T_{b_i}, \tag{4}$$

where, N is the number of nonempty blocks in the volume,and T_{b_i} is the processing time for the i-th volume block b_i. Each block processing time T_{b_i} is determined by a variety of factors, mainly including: (i) the overall host processing power, which is further affected by the dynamic system overhead due to the sharing of system resources among concurrent jobs; and (ii) the nonempty voxels, which depend on selected iso-values or transfer functions. We model T_{b_i} as an independent random variable with a distribution (μ_i, σ_i^2), which is closely related to the block size, as observed in our experiments. Thus, in order to obtain an accurate estimate of T_{b_i} with a controllable small variation σ_i^2, in practice, we conduct a sufficiently large number of tests for a given block size so that the accumulated variance $\sum_{i=1}^{N} \sigma_i^2$ of T_v is controlled within an acceptable range. As long as the processing time for that block size is accurately estimated, the estimation accuracy of T_v in our method is not affected by the size of volume blocks. In the following, we discuss our methods to compute T_{b_i} for isosurface extraction and volume rendering.

1) Marching cubes: The exact number of voxel cells intersecting an isosurface, from now on referred to as *surface voxels*, is not known *a priori*, nor is the spatial distribution of surface voxels throughout the dataset. Therefore, it is not straightforward to accurately estimate the processing time of a volume using the marching cubes algorithm.

However, we found out that it is feasible to make such estimation with a small amount of additional meta data. Obviously, the processing time of a block should highly depend on the number of extracted triangles. While a cell can generate triangles in different ways, previous researchers have identified the topologically unique cases out of all possibilities [33].

We designed a simple experiment to examine the probabilities, p_i, in which a surface voxel generates i (=1,2,3,4) triangles, respectively. By computing a straight-forward weighted average as $\sum_{i=1}^{4}(i \times p_i)$, we obtain the mathematical expectation of the number of triangles generated by a surface voxel. We call the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

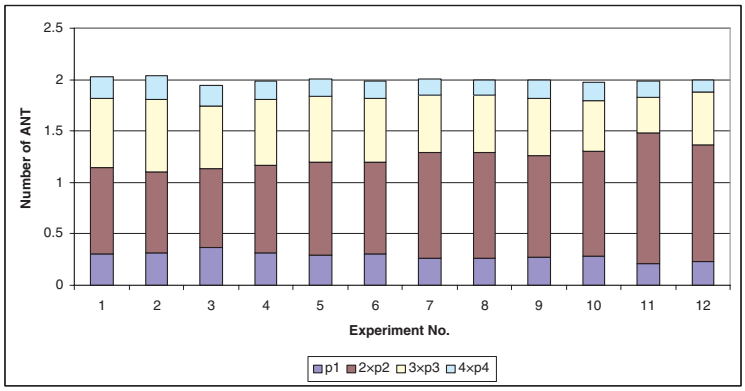


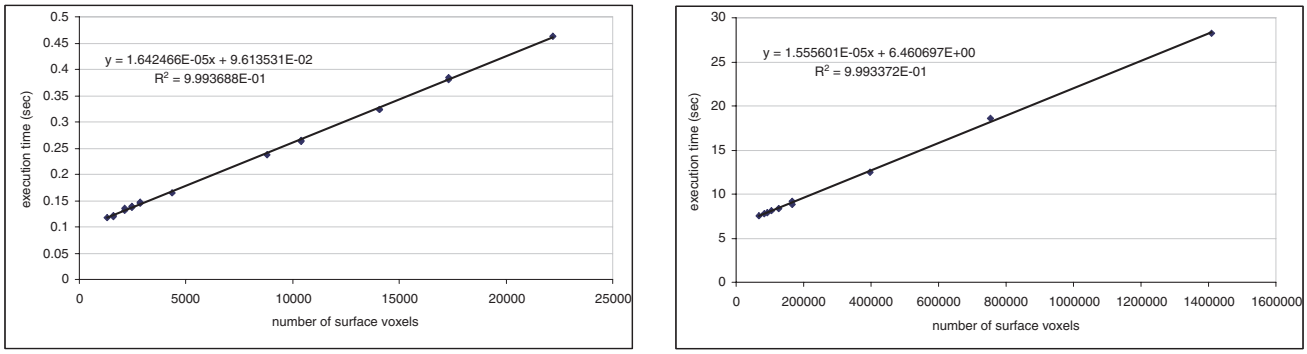
Fig. 4. The *Average-case Number of Triangles*(ANT) distribution obtained from four datasets, each tested with three randomly selected isovalues.

result of this weighted average *Average-case Number of Triangles* (ANT). Since different datasets as well as different iso-values chosen for the same dataset could both affect the value of ANT, we experimented with four different datasets (NegHip, Diesel Injection, UNC Head, MRI Skull) from scientific, engineering, and medical applications. For each dataset, three different isovalues are randomly chosen. We show the results of twelve (4×3) scenarios in Fig. 4, where each individual bar represents a separate test. The colored sections of each bar, in the bottom-up order, illustrate the values of $(i \times p_i)$'s for a surface voxel that generates $i = 1,2,3,4$ triangles, respectively. Although the exact values of $(i \times p_i)$'s vary on a small scale, the values of ANT remain relatively constant. The observed mean of ANT values is 1.99 with a standard deviation of 0.02.

Based on our observation on the consistency in the values of ANT, we construct a linear cost model to estimate T_{b_i}'s for the marching cubes algorithm: $T_{mc} = a \times n_{sv} + b$, i.e. the constant a (the time to generate two triangles) multiplied by the number of surface voxels n_{sv}, and then combined with an additional constant overhead b. To consider the effects of block sizes on the overall performance of CPU caches, a and b, especially b, should vary for different block sizes.

We tested the linear cost model hypothesis using 64 cubed and 256 cubed volume blocks from four datasets described above. The linear regression results shown in Fig. 5 indicate that our linear model agrees with practical scenarios in both cases with a statistical significance of $\chi^2 > 0.99$.

In summary, our procedure to estimate the work load of marching cubes algorithm is the following. First, for any block size and targeted computers chosen by a user, we move a small number of data blocks of that size striped from actual datasets to the targeted machines. Second, to obtain each data point in



(a) $64 \times 64 \times 64$ block size (b) $256 \times 256 \times 256$ block size

Fig. 5. The linear performance model of marching cubes: different data points correspond to various iso-values chosen from the range between 30 and 150, assuming 8-bit values on the voxels.

Fig 5, we repeatedly run a large number of tests on one block and then take the average of those tests as T_{b_i}. Such process is repeated for each block with different iso-values. Third, we compute the linear regression model and store the coefficients a and b. Note the above three steps are performed on each targeted machine separately. Finally, we compute a discrete n_{sv} lookup table (NLUT) for a set of isovalues, densely sampled in the range of all possible isovalues for a targeted dataset. Assuming the coherence in n_{sv} between similar isovalues, we interpolate the n_{sv} value from NLUT at runtime. As shown later in our results section, we consistently achieve a relative prediction error of less than 5.0% in actual runs.

2) Raycasting: For raycasting [34], a common acceleration technique is early ray termination, which is a very simplistic yet very effective method on a single processor. Unfortunately, as shown by previous researchers [35], early ray termination does not scale well in parallel implementations. To achieve scalability, visibility culling is usually performed at block level with pre-computed information such as *Plenoptic Opacity Function* [35]. The transfer functions for volume rendering are typically selected to produce semi-transparent volumes, in which case, early ray termination within voxel blocks may not lead to significant speedups. Therefore, without significantly compromising performances, we rely solely on block-based visibility culling. This way, all blocks are subject to the same computation cost, i.e. $T_{b_i} = c$ where c is a constant. The estimation of c can be done by running the raycasting algorithm on a large number of non-empty blocks and choosing the average time spent on a block. In our experiments on a PC equipped with 2.4 GHz CPU and 2 GBytes memory, we ran the raycasting algorithm on a dataset with 512 non-empty blocks of 64^3 voxels. The time to render each block is 0.387 seconds on average with a relative standard deviation of 4.7%.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

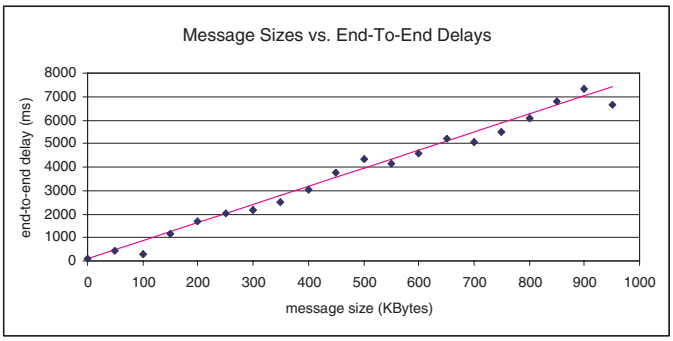


Fig. 6. End-to-end delay measurements between ORNL and LSU.

B. Bandwidth Estimation

Due to the complex traffic distributions over wide-area networks and the non-linear nature of transport protocols (particularly, TCP), the throughput as perceived by visualization modules are typically different from the link capacities. We define *effective path bandwidth* (EPB) as the network transport throughput observed on the virtual link connecting the visualization modules on two nodes. Obviously, EPB is heavily influenced by conditions of cross traffic (i.e. concurrent traffic sharing network resources), and in addition the transport protocol employed. Note that a virtual link connecting any two nodes in G may correspond to a multi-hop data path in wide-area networks, which usually consists of multiple underlying physical links. To estimate EPB, we approximate the end-to-end delay in transmitting a message of size r on a path P as

$$d(P,r) = r/EPB(P). \tag{5}$$

The active measurement technique generates a set of test messages of various sizes, sends them to a destination node through a transport channel such as a TCP flow, and measures end-to-end delays. We then fit a linear regression model to the obtained size-delay data points, whose slope corresponds to EPB. For validation, we measured the actual delays between Louisiana State University (LSU) and Oak Ridge National Laboratory (ORNL). We show both the measured results and the corresponding linear model in Fig. 6. Here, each data point is an average of three separate measurements. From that linear model, we estimate the EPB on this path to be about 1.0 Mb/s.

For high fidelity results, the active measurement operation uses the same transport method as that used by the visualization modules. We note here that there exist publicly available network tools such as Iperf, or NWS [36] that can also be used for estimating EPB. Our method for bandwidth estimation provides: (i)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

performance guarantee on the estimation [37], (ii) an automatic re-estimation triggered by drastic changes in network conditions and user interactions, and (iii) utilization of the same transport modules for both bandwidth estimation and data transmission.

V. Message-Based Control Flow

Many previous remote visualization systems are based on a client-server model, which works very well as long as end nodes have all the needed capabilities. Introducing intermediate nodes into remote visualization systems could lead to enriched functionality and better system throughput. However, the traditional client-server model lacks the capability to exploit the flexiblity and resources at the intermediate nodes. However, this is a complex problem since different intermediate nodes may have been utilized at different times depending on the nature of the visualization task.

To support dynamic configuration of a different visualization pipeline over networked nodes, we adopt a message-based control flow. Each node in our system acts as an independent state machine. An operation is always triggered by a message and the resultant outputs are sent as messages as well. On each node, three threads are executed in endless loops for receiving messages (RecvThread), processing data (ProcThread), and sending results (SendThread). While RecvThread and SendThread are the same among all nodes, the nature of ProcThread determines the type of a node, namely client, CM, DS and CS as described in Section III-C. The overall control flow for our entire system is summarized in Fig. 7.

A visualization task is initiated by a client sending a request to a designated CM, after which the system is driven completely by user interaction (on the client) and control and data messages (for all other nodes including CM, DS, CS). System control messages are used to initiate/terminate a visualization session or report a service failure or establish/close visualization routing paths. Visualization control messages are used to deliver visualization-specific information such as choice of visualization method, viewport resolution, viewing parameters, feature value (i.e. iso-value), and appearance definitions. Data messages are used to transmit raw data or visualization results such as geometry, intermediate volume rendered imagery results and final framebuffer. As shown in Fig. 7, the ProcThreads in different nodes perform their own specific actions in processng the incoming messages.

One critical task in our system design and implementation is the dynamic computation and setup of a VRT by the CM node in response to a specific visualization request or sensed change in network and computing environment. Upon receiving a new visualization request or modifications of visualization

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

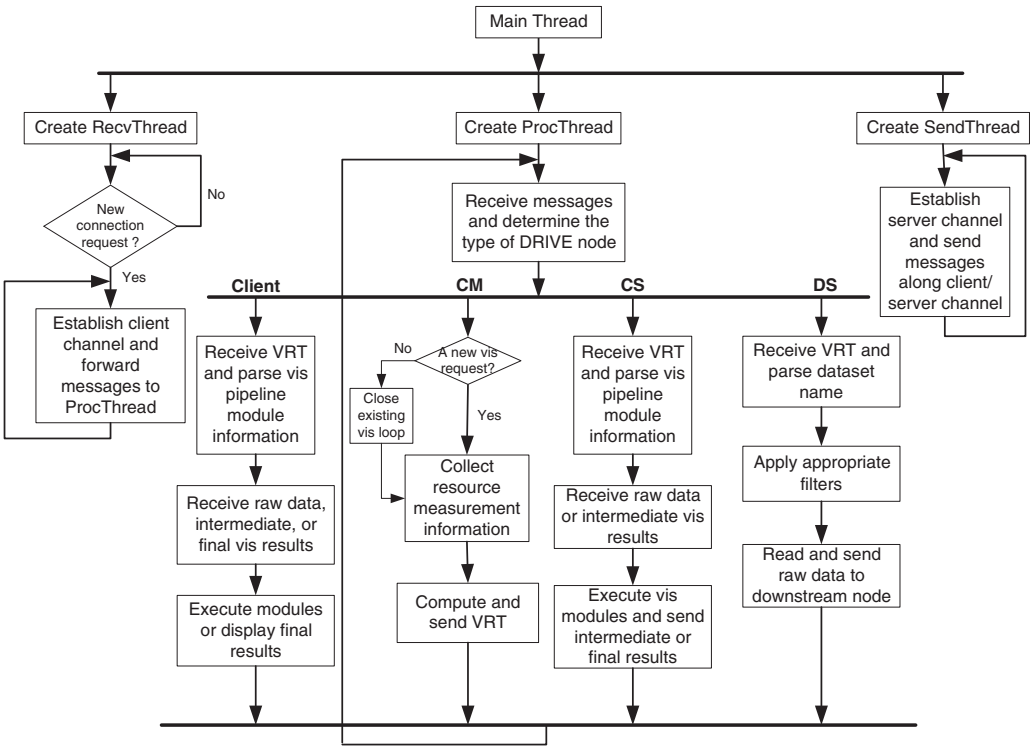


Fig. 7. DRIVE control flow diagram.

parameters (such as data source or visualization method), CM node employs the dynamic programming method to create a new routing table, and compares it with an existing one if any. Routing decisions are then made as per the following conditions: (i) if there is no existing routing path, CM initiates a new path for the pipeline and sends the routing table to appropriate nodes; (ii) if the existing routing path is different from the computed one, CM closes the existing path and establishes a new one by sending the routing table to appropriate nodes; and (iii) if the routing path exists and is the same as the computed one but has different assignments of computing modules, CM updates the module assignments. Upon receiving a routing table, an intermediate node (DS or CS) simply creates a connection (if no connection exists) and forwards the routing table to its immediate downstream node.

Our system uses the latest VRT for all later communications unless a new visualization request arrives, a different visualization method is selected, or there are drastic changes in network traffic or node load conditions.

VI. Implementation and Experimental Results

Our prototype system, DRIVE, is implemented in C++ on Linux operating system using GTK+ for the client GUI. In this section, we describe implementation details and present experimental results in

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

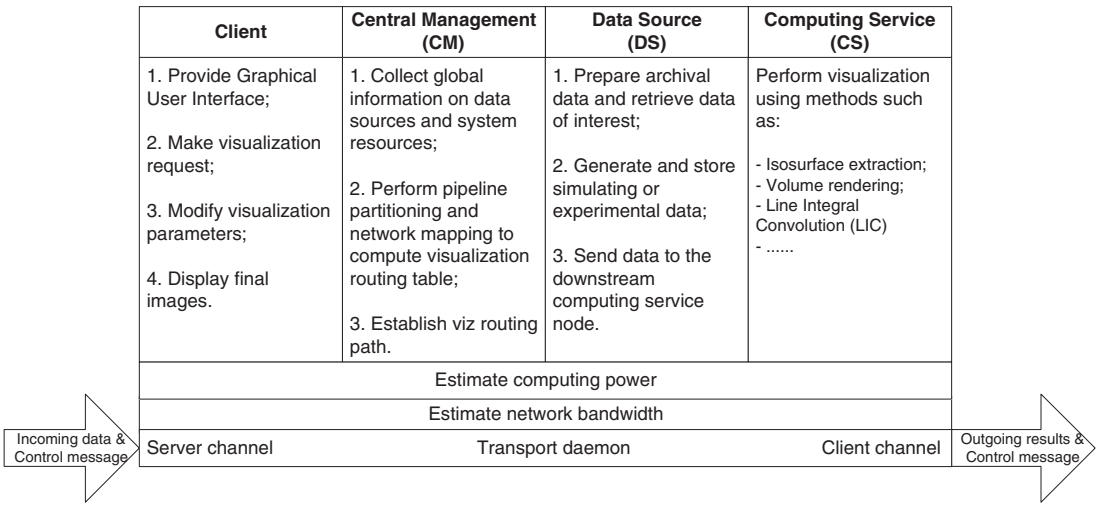


Fig. 8. DRIVE function diagram.

Internet deployments.

A. *DRIVE Functional Diagram*

The functional diagram of four DRIVE elements is shown in Fig. 8. Each DRIVE node implements two types of communication channels: server channel for receiving and client channel for sending. In general, the incoming data is transmitted via the server channel from its upstream node, while the outgoing results are transmitted via the client channel to its downstream node. Control messages can be carried in both directions to report service failures, establish and close visualization routing paths.

The information on resource availability is collected by two measurement units, one estimating the network bandwidth and the other estimating the processing power. The collected information is sent to the CM periodically for calculating the optimal system configuration. This information update may also be triggered by the observation of drastic changes in the current measurements. Particularly, to account for the time-varying network utilizations and CPU occupations, the CM always issues an active inquiry message to all participating hosts for immediate update on the resource information upon the arrival of each new visualization request.

A client node usually resides on a host equipped with a display device ranging from a personal desktop to a powerwall. It displays the final images and enables end users to interact with the visualization application. The main function of CM node is to use the global information collected on data sources and system resources to compute and establish the optimal visualization pipeline for a specific visualization task.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

A DS node stores pre-computed archival datasets. For online computational/experimental monitoring and steering, it might be a simulation running on a computing host or an experiment in progress on a user facility. The retrieved data is sent to the downstream CS node along the routing path for further processing. A CS node can be located anywhere in the network and can range from a workstation to a cluster to a custom rendering engine. They receive data from upstream nodes, perform specific visualization computations, and output final or intermediate visualization results to downstream nodes.

B. System Deployment

For testing purposes, we deployed the DRIVE system on a number of Internet nodes distributed across the United States as shown in Fig. 9. These nodes consist of supercomputers, PC clusters, storage systems, and PC Linux workstations [2]. To demonstrate the capability of supporting multiple clients simultaneously, we selected three representative client nodes located at North Carolina State University (NCSU, east coast), University of California at Davis (UCD, west coast), and Louisiana State University (LSU, southern US), respectively. Three CM nodes are set up at Pittsburgh, Los Angeles, and Kansas City, respectively, gathering and storing time-varying information on data repository, node deployment, and cost models. Since the actual computing tasks are performed on CS nodes, the selection of a different CM node usually has a negligible impact on the overall system performance. However, to minimize communication cost, a client typically selects CM node with a reliable and fast connection.

We collected a wide range of datasets generated by various scientific, medical, and engineering applications including Terascale Supernova Initiative (TSI) project [38], Visible Human Project [39], Jet Shockwave Simulation of the Kelvin-Helmholz instability, and Rayleigh-Taylor hydrodynamic instability simulations. The sizes of these datasets range from dozens of MBytes to hundreds of GBytes. In particular, the TSI datasets generated on ORNL Cray supercomputer and archived on the storage system at ORNL are about 300 GBytes, containing 128 time steps of 864^3 volumes. The other smaller datasets are duplicated on the storage systems deployed at The Ohio State University (OSU) and Georgia Institute of Technology (GIT). In order to handle large-scale datasets, we utilized four PC clusters: hawk cluster at ORNL, boba cluster at University of Tennessee at Knoxville (UTK), orbitty cluster at NCSU, and bale cluster at Ohio Supercomputing Center (OSC).

[2]These workstations are either PlanetLab nodes or Linux boxes deployed at various collaborative sites.

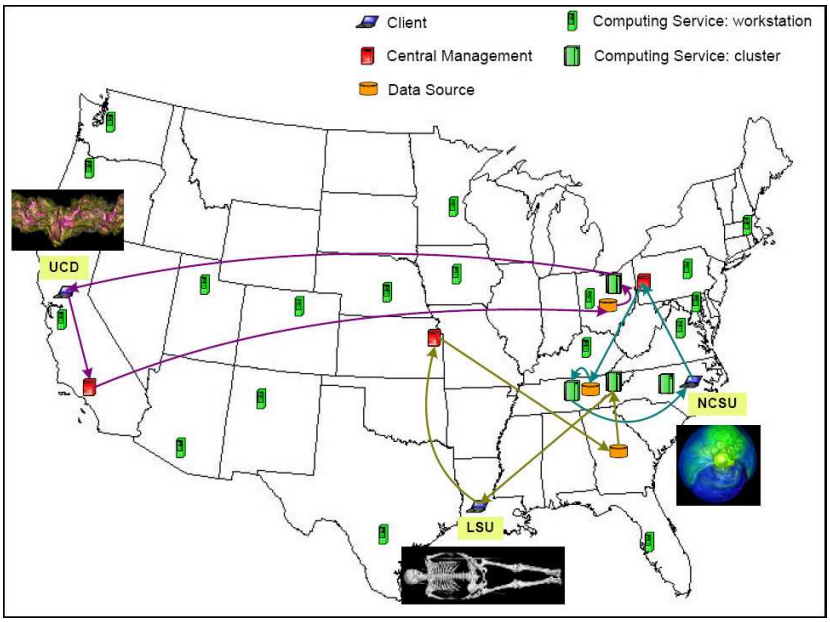


Fig. 9. System deployment and initial DRIVE configurations for three concurrent sessions of distributed visualization.

C. Initial Setup

As illustrated in Fig. 9, each of the three clients issues a particular remote visualization request and in response the corresponding CM computes an initial optimal pipeline configuration. Note that these three concurrent visualization loops determined by the optimization algorithm happen to be disjoint from each other. This is likely the case if the nodes and links in the environment have comparable computing and networking capabilities so that the visualization and transport subtasks tend to spread out for load balancing. We would also like to point out that some parts of the visualization loops might overlap if running simultaneously, for example, when a particular node or link has extremely higher computing or bandwidth than others. In other words, these resources might be shared among several concurrent visualization loops.

(A) *Astrophysics Datasets:* In the visualization task initiated by a scientific application at NCSU, an astrophysicist uses the workstation dali.physics.ncsu.edu to visualize the TSI datasets located on the ORNL storage system using the raycasting technique. The DRIVE system selects the nearby hawk cluster at ORNL for computing, which delivers the final image to the client at NCSU over a wide-area connection in each time step.

(B) *Medical Application:* In the visualization task initiated by a medical application at LSU, a physician uses the workstation robot.rrl.lsu.edu to diagnose the visible woman MRI image rendered by the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

isosurface extraction technique. The DRIVE system retrieves the dataset from the storage system at GIT and uses the boba cluster at UTK for both geometry extraction and rendering.

(C) *Engineering Application:* In the visualization task initiated at UCD, an engineer uses the workstation jalapeno.cs.ucdavis.edu to investigate the rage simulation using the raycasting technique. The DRIVE system retrieves the dataset from the storage system at OSU and selects the nearby bale cluster for computing.

D. Self-Adaptation

We illustrate the self-adaptation capability of the DRIVE system, by changing visualization techniques and parameters, and monitoring the variations in computing load and network traffic on each node[3]. Especially, when the monitoring modules deployed on each node detect significant variations in network traffic or computing load, they send the updated measurements of system conditions to the CM. Based on the new measurements, the CM executes the dynamic programming-based optimization algorithm immediately to calculate a new optimal visualization path that adapts to the current system conditions. The routing decision made by the CM using the new visualization path is described in detail in . The events that cause system reconfigurations, and the resultant performance estimators (described in Section IV) and measurements of total delay along the DRIVE loop are tabulated in Fig. 10.

(A) *Astrophysics Datasets:* Event A.1 in Fig. 10 corresponds to the initial DRIVE configuration for the TSI visualization request issued by the client at NCSU. A snapshot of the DRIVE client graphical user interface (GUI) visualizing TSI datasets using raycasting technique is shown in Fig. 12. In Event A.2, the client switches the visualization method to isosurface extraction with an isovalue of 218. The system utilizes the same configuration of computing resources, including 8 nodes for parallel computing on the hawk cluster. In Event A.3, the client selects a new isovalue of 21, which results in a significantly reduced number of triangles. In response, the DRIVE system decides to transmit the geometry data directly to the remote client workstation instead of rendering the geometries on the hawk cluster. In Event A.4, there is a rapid increase of computing load on the hawk cluster. As a result, a slight change of isovalue forces the system to shift the isosurface extraction task to the orbitty cluster at NCSU using 32 processor nodes. The extracted geometry data is then sent via a

[3]The variations in transport and computing times are either due to external events such as cross traffic on the Internet and concurrent workload on clusters or carefully designed experiments.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

Client	Event #	Activity	Visualization method	Dataset	Location of central management	Location of data source	Computing service node 1: computing	Computing service node 2: rendering	End-to-end delay perf. estimation (seconds)	End-to-end delay perf. measurement (seconds)
dali @ NCSU	A.1	Initial setup	Raycasting	TSI simulation	Pittsburgh, PA	ORNL storage	hawk cluster @ ORNL (use 8 nodes)		1.95	2.27
	A.2	Visualization method change	Isosurface extraction: iso = 218	TSI simulation	Pittsburgh, PA	ORNL storage	hawk cluster @ ORNL (use 8 nodes)		12.28	12.93
	A.3	iso value change	Isosurface extraction: iso = 21	TSI simulation	Pittsburgh, PA	ORNL storage	hawk cluster @ ORNL: (use 8 nodes)	dali client workstation @ NCSU	5.73	6.03
	A.4	Computing time and iso value change	Isosurface extraction: iso = 48	TSI simulation	Pittsburgh, PA	ORNL storage	orbitty cluster @ NCSU (use 32 nodes)	dali client workstation @ NCSU	9.23	9.63
robot @ LSU	B.1	Initial setup	Isosurface extraction: iso = 57	Visible woman MRI data	Kansas City, MO	GIT storage	boba cluster @ UTK (use 16 nodes)		24.40	25.38
	B.2	Data source change	Isosurface extraction: iso = 170	Brain CT data	Kansas City, MO	GIT storage	boba cluster @ UTK (use 16 nodes)	PlannetLab workstation @ Florida	4.31	4.60
	B.3	iso value and bandwidth change	Isosurface extraction: iso = 165	Brain CT data	Kansas City, MO	OSU storage	bale cluster @ OSC (use 16 nodes)	PlannetLab workstation @ Houston	1.68	1.89
	B.4	Data source and computing time change	Isosurface extraction: iso = 40	Hand CT data	Kansas City, MO	GIT storage	robot client workstation @ LSU		3.07	3.43
jalapeno @ UCD	C.1	Initial setup	Raycasting	Rage	Los Angeles, CA	OSU storage	bale cluster @ OSC (use 16 nodes)		2.32	2.58
	C.2	Data source change	Raycasting	Jet shockwave simulation	Los Angeles, CA	GIT storage	boba cluster @ UTK (use 16 nodes)		9.05	9.44
	C.3	Computing time, bandwidth, visualization method change	Isosurface extraction: iso = 248	Jet shockwave simulation	Los Angeles, CA	OSU storage	PlannetLab workstation @ Utah	jalapeno client workstation @ UCD	17.11	17.76

Fig. 10. Adaptive reconfiguration of DRIVE visualization loops in response to system variations and client interactions.

fast LAN connection to the client for rendering.

(B) *Medical Application:* Event B.1 depicts the initial DRIVE configuration for the visualization of the visible woman MRI data using isosurface extraction technique with an isovalue of 57. In Event B.2, the client switches to the visualization of a brain CT dataset with an isovalue of 170. Since the size of the geometry data extracted with this isovalue is comparable to the frame buffer size and the client workstation is heavily loaded with several other concurrent graphics applications, the system transmits the geometry data to an intermediate PlanetLab node planetlab2.csee.usf.edu deployed at University of South Florida, which renders and sends the final image to the client for display. In Event B.3, due to the drastic performance decrease on the boba cluster, the system chooses a duplicate of the brain data on the OSU storage system and uses the bale cluster for isosurface extraction. The previous rendering node in Florida is replaced by a new PlanetLab node planetlab2.hstn.internet2.planet-lab.org at Houston to take advantage of the higher transport bandwidth of its connection to the selected bale cluster. In Event B.4, the hand CT data on the GIT storage system is selected for visualization using isosurface extraction technique. Due to the small data size, the system sends the entire raw data directly to the client to avoid the unnecessary computing overhead (data partitioning and image

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

gathering) incurred for parallel processing on a cluster.

(C) *Engineering Application:* Event C.1 corresponds to the initial network setup for the client at UCD that requests to visualize the rage dataset. In Event C.2, the client switches to visualize the jet shockwave simulation data using the raycasting technique. The dataset is then retrieved from GIT storage and sent to the boba cluster for parallel rendering using 16 nodes. In Event C.3, the visualization method is switched to isosurface extraction. Since at that moment, there is a substantial performance drop on the clusters and only very limited bandwidth is available, the system chose the workstation planetlab2.flux.utah.edu deployed at University of Utah among the network for isosurface extraction. Due to its limited rendering capability, the extracted geometry data is sent to the client for final rendering and display.

For all the events shown in Fig. 10, the overall estimation error of transport and computing times is less than 5.0%, which demonstrates the accuracy of our performance models for both network and visualization parts. We also observed that the system overhead is typically of less than one second, which is about 7.0% of the total loop delay in Fig. 10. This overhead consists of two components: setup time and loop time. The former is the time needed to compute VRT and establish a visualization path. An example of VRT is illustrated in Fig. 11. The computing time for a VRT scales nicely with the number of nodes in the network due to the polynomial computing time for the dynamic programming-based optimization algorithm, as shown in Section III-C. The latter is the time spent in delivering control messages along the network loop for interactive visualization operations.

The setup and loop times are related to the size of VRT's and control messages. The size of a VRT depends on the number of modules in a visualization pipeline. A typical pipeline of common visualization techniques such as isosurface extraction and raycasting may produce a VRT of several hundred bytes. A control message is generally of several dozen bytes, containing user-specified parameters. Hence, we can conveniently pack a VRT or a control message in a single TCP segment or UDP datagram, which implies that the loop time is roughly the sum of the minimum end-to-end link delays along a visualization loop. Once a visualization path has been established, the system only has the loop time overhead, which is generally less than half second, if the routing table remains the same. This amount of overhead is almost negligible compared to the end-to-end delay on the order of dozens of seconds for large-scale remote visualization.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

NUMBER OF MODULES: 5

MODID	**MODID**	**MODID**	**MODID**	**MODID**
0	1	2	3	4
HOSTNAME	**HOSTNAME**	**HOSTNAME**	**HOSTNAME**	**HOSTNAME**
planetlab1.kscy.internet2.planet-lab.org	ccil.cc.gatech.edu	boba121.sinrg.cs.utk.edu	planetlab2.csee.usf.edu	robot.rrl.lsu.edu
IPADDR	**IPADDR**	**IPADDR**	**IPADDR**	**IPADDR**
198.32.154.202	130.207.117.12	160.36.140.77	131.247.2.242	130.39.224.168
PORTNO	**PORTNO**	**PORTNO**	**PORTNO**	**PORTNO**
CENTRALMANAGEMENTLISTENPORT	COMPUTINGNODELISTENPORT	COMPUTINGNODELISTENPORT	COMPUTINGNODELISTENPORT	COMPUTINGNODELISTENPORT
MODTYPE	**MODTYPE**	**MODTYPE**	**MODTYPE**	**MODTYPE**
MODULE_NOACTION	MODULE_SENDRAWDATA	MODULE_ISOEXTRACTION	MODULE_ISORENDERING	MODULE_DISPLAY
SERVICE	**SERVICE**	**SERVICE**	**SERVICE**	**SERVICE**
SERVICE_MPIDISABLED	SERVICE_MPIDISABLED	SERVICE_MPIENABLED	SERVICE_MPIDISABLED	SERVICE_MPIDISABLED

Fig. 11. An example of visualization routing table (VRT) in DRIVE.

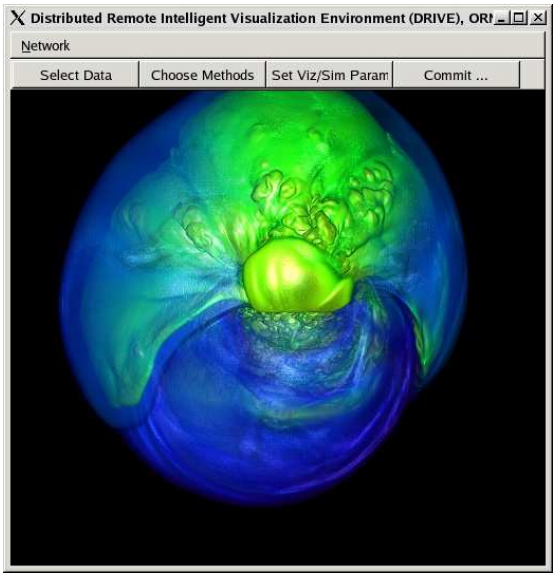


Fig. 12. DRIVE client GUI with TSI simulation dataset rendered using raycasting technique.

It is interesting to point out that the advantage of utilizing an intermediate MPI module is not very obvious for small datasets because of the overhead incurred by data distributions and communications among cluster nodes. As a matter of fact, for datasets of several or dozens of MBytes, a simple PC-PC configuration with any type of server/client mode might be sufficient to deliver reasonable performances for remote visualization (such as Event B.4). However, for large-scale scientific datasets, parallel processing modules have become an indispensable tool supporting the visualization task. Hence, it also becomes increasingly important to select an appropriate set of processing nodes available in the Internet to map the visualization pipeline for the optimal performance.

To show that our system has a relatively small control overhead, we also conducted performance comparisons between DRIVE and ParaView for the same visualization tasks. For these tasks, DRIVE consistently achieved comparable or somewhat better performance compared to ParaView. The configuration, however, had to be manually performed in ParaView setup, whereas in DRIVE the configuration is automatically

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

computed and is self-adaptive. The performance advantage observed in DRIVE may have been caused by higher processing and communication overhead incurred by visualization and network transfer functions of ParaView. However, it is not our main goal to compare the performance of our visualization modules with that of existing ones but rather to illustrate the performance of visualization pipeline mapping onto network nodes.

VII. Conclusions

Interactive remote visualization of large-scale scientific datasets and on-going computations is a challenging research and development task. In this paper, we present a general framework for remote visualization systems, which self-adapt over wide-area networks with dynamic host and network conditions. We have found that it is practical to develop performance models for estimating times of both visualization computation and network transport subtasks. In addition, using these performance estimations, we developed an efficient dynamic programming method for computing an optimal configuration of a visualization pipeline to achieve minimal end-to-end delay. By integrating a message-driven control mechanism, our system could efficiently self-adapt to dynamic scenarios. Even though we focused on volume visualizations, our framework can be readily extended to supporting any other computing tasks with a pipelined process flow.

It is of future interest to study various analytical formulations of this class of problems from the viewpoint of computational complexity. For experimental research, we plan to deploy our distributed visualization system over dedicated networks UltraScience Net [40] and CHEETAH [41] to evaluate possibilities of handling terabyte datasets using remote visualization. Our near-term focus in that regard involves incorporating latest network transport protocols for high-performance shared [42] and dedicated connections [43] with optimized remote visualization algorithms. Finally, we note that although most visualization techniques employ a linear pipeline without branches or loops, computational science tasks involving comparative visualizations, and coordinated computational monitoring and steering, require more complex configurations. We plan to expand our framework to address visualization pipelines with branches and loops.

Acknowledgments

This research is sponsored by the High Performance Networking and SciDAC Programs of the Office of Science, U.S. Department of Energy, under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

and by National Science Foundation Under Grants No. ANI-0229969 and No. ANI-0335185. The authors would like to thank all the collaborators for providing the access to their facilities and datasets used in the experiments. The authors would also like to thank three anonymous reviewers for their insightful suggestions and constructive comments that have greatly improved the technical quality and presentation of the revised paper.

REFERENCES

[1] A. Mezzacappa, "Scidac 2005: Scientific discovery through advanced computing," *Journal of Physics: Conference Series*, vol. 16, 2005. [Online]. Available: http://stacks.iop.org/1742-6596/16/i=1/a=E01
[2] G. P. Bonneau, T. Ertl, and G. M. Nielson, Eds., *Scientific Visualization: The Visual Extraction of Knowledge from Data*. Springer-Verlag, 2005.
[3] K.-L. Ma and D. M. Camp, "High performance visualization of time-varying volume data over a wide-area network," in *Proc. of Supercomputing*, 2000.
[4] J. Ding, J. Huang, M. Beck, S. Liu, T. Moore, and S. Soltesz, "Remote visualization by browsing image based databases with logistical networking," in *Proc. of Supercomputing*, 2003.
[5] W. Bethel, B. Tierney, J. Lee, D. Gunter, and S. Lau, "Using high-speed wans and network data caches to enable remote and distributed visualization," in *Proc. of Supercomputing*, 2000.
[6] A. Neeman, P. Sulatycke, and K. Ghose, "Fast remote isosurface visualization with chessboarding," in *Proc. of Parallel Graphics and Visualization*, 2004, pp. 75–82.
[7] "Vis5d+," http://vis5d.sourceforge.net/.
[8] "Paraview," http://www.paraview.org/HTML/Index.html.
[9] "Aspect," http://www.aspect-sdm.org/.
[10] "Computational engineering international," http://www.ceintl.com/products/ensight.html.
[11] Y. Livnat and C. Hansen, "View dependent isosurface extraction," in *IEEE Visualization '98*, 1998, pp. 175–180.
[12] Z. Liu, A. Finkelstein, and K. Li, "Progressive view-dependent isosurface propagation," in *Vissym'01*, 2001.
[13] S. Stegmaier, M. Magallon, and T. Ertl, "A generic solution for hardware-accelerated remote visualization," in *Proc. of Data Visualisation*, 2002, pp. 87–95.
[14] I. Foster, J. Insley, G. von Laszewski, C. Kesselman, and M. Thiebaux, "Distance visualization: Data exploration on the grid," *Computer*, vol. 32, no. 12, pp. 36–43, 1999. [Online]. Available: citeseer.ist.psu.edu/foster99distance.html
[15] I. T. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid - enabling scalable virtual organizations," *CoRR*, vol. cs.AR/0103025, 2001.
[16] I. T. Foster, "Globus toolkit version 4: Software for service-oriented systems," in *NPC*, 2005, pp. 2–13.
[17] D. Kranzlmuller, G. Kurka, P. Heinzlreiter, and J. Volkert, "Optimizations in the grid visualization kernel," in *Proceedings of International Parallel and Distributed Processing Symposium*, 2002, pp. 129–135.
[18] P. Heinzlreiter, D. Kranzlmuller, and J. Volkert, "Network transportation and optimization for grid-enabled visualization techniques," *Neural, Parallel Sci. Comput.*, vol. 12, no. 3, pp. 307–328, 2004.
[19] J. Shalf and E. W. Bethel, "The grid and future visualization system architectures," *IEEE Computer Graphics and Applications*, vol. 23, no. 2, pp. 6–9, 2003.
[20] I. J. Grimstead, N. J. Avis, and D. W. Walker, "Automatic distribution of rendering workloads in a grid enabled collaborative visualization environment," in *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*. Washington, DC, USA: IEEE Computer Society, 2004, p. 1.
[21] K. Engel, O. Sommer, and T. Ertl, "An interactive hardware accelerated remote 3d-visualization framework," in *Proc. of Data Visualisation*, 2000, pp. 167–177.
[22] K. Brodlie, D. Duce, J. Gallop, M. Sagar, J. Walton, and J. Wood, "Visualization in grid computing environments," in *Proc. of IEEE Visualization*, 2004, pp. 155–162.
[23] M. D. Beynon, A. Sussman, T. Kurc, and J. Saltz, "Optimizing execution of component-based applications using group instances," *CCGrid*, pp. 56–65, 2001.
[24] J. Ahrens, K. Brislawn, K. Martin, B. Geveci, C. C. Law, and M. Papka, "Large-scale data visualization using parallel data streaming," *IEEE Computer Graphics and Applications*, vol. 21, no. 4, pp. 34–41, 2001.
[25] E. J. Luke and C. D. Hansen, "Semotus visum: a flexible remote visualization framework," in *Proc. of IEEE Visualization*, 2002, pp. 61–68.
[26] I. M. Boier-Martin, "Adaptive graphics," *IEEE Computer Graphics and Applications*, vol. 23, no. 1, pp. 6–10, 2003.
[27] I. Bowman, J. Shalf, K.-L. Ma, and W. Bethel, "Performance modeling for 3d visualization in a heterogeneous computing environment," in *Technical Report No. 2005-3, Department of Computer Science, University of California at Davis*, 2005.
[28] W. Schroeder, K. Martin, and B. Lorensen, *The Visualization Toolkit (2nd Edition)*. Prentice Hall PTR, 1998.
[29] "Common data format," http://nssdc.gsfc.nasa.gov/cdf.
[30] "Hierarchical data format," http://hdf.ncsa.uiuc.edu.
[31] "Network common data form," http://my.unidata.ucar.edu/content/software/netcdf.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

[32] M. Zhu, Q. Wu, N. Rao, and S. Iyengar, "Adaptive visualization pipeline decomposition and mapping onto computer networks," in *Proc. of the IEEE Internatioal Conference on Image and Graphics*, Hong Kong, China, December 18-20, 2004, pp. 402–405.

[33] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *Computer Graphics*, vol. 21, no. 4, pp. 163–169, July 1987.

[34] M. Levoy, "Efficient ray tracing of volume data," *ACM Transactions on Graphics*, vol. 9, no. 3, pp. 245–261, 1990.

[35] J. Gao, J. Huang, H.-W. Shen, and J. A. Kohl, "Visibility culling using plenoptic opacity functions for large data visualization," in *Proc. of IEEE Visualization '03*, 2003, pp. 341–348.

[36] "Network weather service," http://nws.cs.ucsb.edu.

[37] Q. Wu, N. Rao, and S. Iyengar, "On transport daemons for small collaborative applications over wide-area networks," in *Proc. of the 24th IEEE International Performance Computing and Communications Conference*, Phoenix, Arizona, April 7-9, 2005.

[38] "Terascale supernova initiative," http://www.phy.ornl.gov/tsi.

[39] "Visible human project," http://www.nlm.nih.gov/research/visible/.

[40] N. S. V. Rao, W. R. Wing, , S. M. Carter, and Q. Wu, "Ultrascience net: Network testbed for large-scale science applications," *IEEE Communications Magazine*, vol. 43, no. 11, pp. s12–s17, 2005, expanded version available at www.csm.ornl.gov/ultranet.

[41] X. Zheng, M. Veeraraghavan, N. S. V. Rao, Q. Wu, and M. Zhu, "CHEETAH: Circuit-switched high-speed end-to-end transport architecture testbed," *IEEE Communications Magazine*, vol. 43, no. 11, pp. s11–s17, 2005.

[42] M. Hassan and R. Jain, *High Performance TCP/IP Networking: Concepts, Issues, and Solutions*. Prentice Hall, 2004.

[43] N. Rao, Q. Wu, S. Carter, and W. Wing, "High-speed dedicated channels and experimental results with hurricane protocol," *Annals of Telecommunications*, vol. 61, no. 1-2, pp. 21–45, 2006.