2008

Computational Monitoring and Steering Using Network-Optimized Visualization and Ajax Web Server

Mengxia Zhu
Southern Illinois University Carbondale, mzhu@cs.siu.edu

Qishi Wu
University of Memphis

Nageswara S.V. Rao
Oak Ridge National Laboratory

Published in Zhu, M., Wu, Q., & Rao, N. S. V. (2008). Computational monitoring and steering using network-optimized visualization and Ajax web server. IEEE International Symposium on Parallel and Distributed Processing, 2008. IPDPS 2008, 1-12. doi: 10.1109/IPDPS.2008.4536260

Computational Monitoring and Steering Using Network-Optimized Visualization and Ajax Web Server

Mengxia Zhu
Dept of Computer Science
Southern Illinois University
Carbondale, IL 62901
mzhu@cs.siu.edu

Qishi Wu
Dept of Computer Science
University of Memphis
Memphis, TN 38152
qishiwu@memphis.edu

Nageswara S.V. Rao
Computer Science & Math Division
Oak Ridge National Laboratory
Oak Ridge, TN 37830
raons@ornl.gov

Abstract

We describe a system for computational monitoring and steering of an on-going computation or visualization on a remote host such as workstation or supercomputer. Unlike the conventional "launch-and-leave" batch computations, this system enables: (i) continuous monitoring of variables of an on-going remote computation using visualization tools, and (ii) interactive specification of chosen computational parameters to steer the computation. The visualization and control streams are supported over wide-area networks using transport protocols based on stochastic approximation methods to provide stable throughput. Using performance models for transport channels and visualization modules, we develop a visualization pipeline configuration solution that minimizes end-to-end delay over wide-area connections. The user interface utilizes Asynchronous JavaScript and XML (Ajax) technologies to provide an interactive environment that can be accessed by multiple remote users using web browsers. We present experimental results on a geographically distributed deployment to illustrate the effectiveness of the proposed system.

1 Introduction

The computing power of supercomputers continues to increase thereby enabling large-scale computations of unprecedented scale. Such computations indeed have become an indispensable research tool for a number of scientific applications in disciplines as diverse as biology, chemistry, meteorology, and astrophysics [25]. These applications including TSI [1] and combustion research [2] often generate vast amounts of simulation data in the range of terabytes to petabytes, which must be stored, transferred, visualized, and analyzed by geographically distributed teams of scientists. In several cases, large-scale computations are scheduled in a "batch mode" on supercomputers and their outputs are examined at the end typically using visualization and other analysis tools. While being effective in some cases, this paradigm potentially leads to runaway computations whose parameters either strayed away from the region of interest or did not show adequate movement in the direction of interest. Such instances represent very ineffective utilization of the valuable computing and human resources. This can be avoided if the progress of computation is monitored on-line, and its parameters are dynamically adjusted to steer the execution.

In many simulations, dynamic parameter specification through visual feedback can identify appropriate regions and can aid the discovery process. In particular, with such a capability, unsuccessful computations can be saved by steering the stray simulations on the fly. Achieving this capability over geographically distributed resources requires an integration of technologies in various fields including high performance and distributed computing, high speed networking, high performance storage systems, and large-scale visualization. In this paper, we propose Remote Intelligent Computational Steering using Ajax technology (RICSA) for online visualization and steering that optimizes the performance of visualization pipelines over wide-area networks. This system integrates three key technology components: (a) network-optimized visualization pipeline using a dynamic programming method, (b) stable transport channels for visualization and computation control streams using stochastic approximation methods, and (c) user interface based on Asychronous JavaScript using XML (Ajax) to provide convenient and wide user access.

A general remote visualization system consists of a remote server acting as a data source, a local rendering/display terminal acting as a client, zero or more intermediate hosts, and a network connecting them all together. The performance of such systems critically relies on how efficiently the visualization pipeline is partitioned

978-1-4244-1694-3/08/$25.00 ©2008 IEEE

and mapped onto the network nodes. Many commercial or noncommercial software products for remote visualization [3, 4, 16] employ a predetermined partition of visualization pipelines and send fixed-type data streams such as raw data, geometric primitives, or framebuffer (FB) to remote client nodes. While such schemes are common, they are not always optimal for high performance visualizations, particularly over wide-area connections. There have been several efforts to design architectures that assign visualization modules across network nodes efficiently. Brodlie *et al.* [20] extended existing dataflow visualization systems to Grid environments. Stegmaier *et al.* [28] provided a generic solution for hardware-accelerated remote visualization that is independent of application and architecture. Bethel *et al.* [18] designed a new architecture that utilizes high-speed WANs and network data caches for data staging and transmission. Luke and Hansen [23] presented a flexible remote visualization framework capable of multiple partition scenarios, which is tested and evaluated on a local network. Bowman *et al.* [19] proposed a framework to predict the processing times of visualization modules using analytic models, which can be used to obtain a suitable mapping of the visualization pipeline. RAVE [5] is a "resource-aware" system that can determine if rendering should be done locally or remotely for satisfactory interactivity. These systems typically require the use of third-party packages with focus on a specific aspect of performance improvement while the proposed RICSA system provides a lightweight implementation and an effective solution to the remote visualization problem from a global optimization view.

There have been various research efforts on the system design and implementation for computational steering systems. However, existing systems including SCIRun [27], CUMULVS [6], VIPER [24], and RealityGrid [7] generally require a high learning curve for all users. Besides, various packages such as Globus, SOAP, PVM [8] and AVS [9] need to be installed at the user sites to realize their full benefits. These factors often place undue burden on users, who are typical scientists, to spend significant effort in setting up and learning a new system. Furthermore, some of these technologies are platform specific and are not widely supported on diverse user platforms. The proposed RICSA system supports a user interface using Ajax web technologies that offer improved productivity and user experience that can be accessed by a web browser available on most user platforms. Partial screen updates and asynchronous communications are two essential features of Ajax that make it suitable for computational monitoring and steering applications. Using Ajax, only user interface elements that contain new information are updated with data received from a server such as next update of a monitored computation. Such a non-interrupted data-driven model replaces the traditional "click, wait, and refresh" page-driven model. With RICSA, any user with an Internet connection can use a web browser to visualize a computation rendered by a remote system, and also steer the computation from a platform anywhere on the Internet.

Control channels with stable dynamics are needed to support computational steering and interactive visualization operations, which require connections that guarantee sufficient, albeit small, bandwidth and low jitter. Inadequate bandwidths often result in poor responsiveness and high jitter may destabilize the control. Based on extensive traffic measurements and analysis, we utilize a new class of transport protocols based on stochastic approximation methods to achieve stable throughput for control channels.

In this paper, we address both analytical and implementation aspects of RICSA. We describe the framework and analytically formulate the problem of minimizing the end-to-end delay of RICSA by considering both transmission and computation times. This analytical model enables us to analyze the algorithmic complexity and optimality of mapping the visualization pipeline onto the network. Based on performance measurements on both visualization modules and transport channels, we derive the optimal decomposition and mapping scheme using the dynamic programming method for end-to-end delay minimization.

The rest of the paper is organized as follows. In Section 2, we present the system architecture. In Section 3, we describe the transport protocol that achieves throughput stabilization for control channels. In Section 4, we construct an analytical model for visualization pipeline partitioning and network mapping, and present the optimization method using dynamic programming. Implementation details and experimental results are provided in Section 5. We conclude our work and discuss future research directions in Section 6.

2 System Framework

As shown in Fig. 1, RICSA consists of five virtual component nodes, Ajax client, Ajax front end, central management (CM), simulation/data source (DS), and computing service (CS), which are connected together over a network to form a visualization loop. In general, a simulation/data source node either contains pre-generated datasets or a simulator that runs on a single host, a cluster, or a supercomputer. The simulation data is continuously produced and periodically cached on a local storage device, which serves as a data source.

A computational steering is initiated at a Ajax client node by sending a request specifying the simulator type, variable names, visualization method, and viewing parameters etc. to the Ajax front end, which forwards the request to a designated CM node. The CM node creates a connection and forwards the request to the simulation node, which starts the execution of the simulation code upon receiving

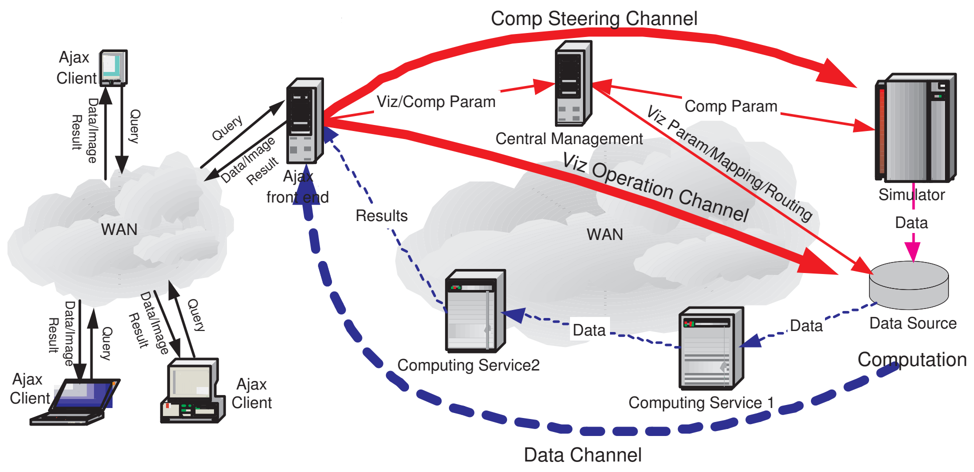


Figure 1. RICSA architecture and components.

simulation parameters. The CM node determines the best system configuration to accomplish the visualization tasks for the newly generated dataset. Specifically, based on the global knowledge of system resource distributions and simulation dataset properties, the CM node strategically partitions the visualization pipeline into groups and selects an appropriate set of CS nodes to execute the visualization modules. The computation for pipeline partitioning and network mapping results in a visualization routing table (VRT), which is delivered sequentially over the loop to establish the network routing path. Ajax front end will then save the received images as fixed-size files that are to be delivered to the browser through the object exchange mechanism of XMLHttpRequest.

A visualization and steering loop comprises two types of channel segments: (i) control channel from the Ajax front end node to the simulation/data source node for computational steering or visualization operations, and (ii) data channel from the computation node back to the front end node, as represented by the solid and dotted lines in Fig. 1, respectively. The front end node uses control channels to transmit computational steering parameters to the simulator and visualization operation parameters to the data source. Note that the control and data channels have very different transport performance requirements. In general, the transmission of control parameters of several KBytes or MBytes needs fairly small bandwidth but with smooth transport dynamics, while on the data channel, the throughput is usually of the most concern for large data transfer.

3 Transport Stabilization of Control Channel

We integrate the transport stabilization method described in [26] into the proposed RICSA system to provide stable channels for smooth control message transfer. In this transport method, Rao *et al.* consider a general window-based transport structure shown in Fig. 2 that utilizes UDP for application-level transport. This model sends $W_c(t)$ UDP datagrams periodically with an interval (sleep time) $T_s(t)$. The source rate $r_S(t)$ of a sender is primarily determined by: $r_S(t) = \frac{W_c(t)}{T_s(t)+T_c(t)}$, where $T_c(t)$ is the time spent on continuously sending a full congestion window of UDP datagrams. The goodput rate, which is the data receiving rate at the receiver ignoring the duplicates, is denoted by $g_R(t)$ in response to the sending rate $r_S(t)$.

The goal of transport stabilization is to adjust $r_S(t)$ to ensure $g_R(t) = g^*$ in some sense, where g^* is the specified target goodput level. The rate control is based on the Robbins-Monro stochastic approximation method [22]. At time step t_{n+1}, the new sleep or idle time is computed as follows:

$$T_s(t_{n+1}) = \frac{1.0}{\frac{1.0}{T_s(t_n)} - \frac{a/W_c}{n^\alpha} * (g(t_n) - g^*)} \qquad (1)$$

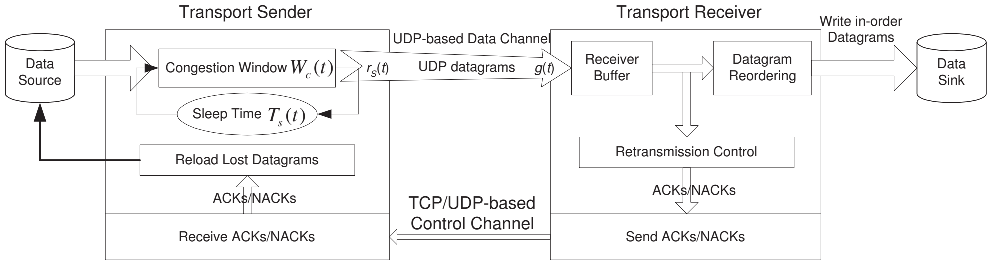


Figure 2. A general transport control structure.

where $g(t_n)$ is the goodput measurement at time step t_n at the sender side. Coefficients a and α are carefully chosen so that the source rate specified by Eq. 1 eventually converges to the target rate. Under the Robbins-Monro conditions on the coefficients, this protocol is analytically shown to asymptotically stabilize at g^* under random losses [26]. This method exhibits very robust stabilization performance over a variety of network connections.

4 Optimal Visualization Pipeline Configuration

This section presents the technical solutions to system optimization for achieving minimum end-to-end delay in remote interactive operations.

4.1 Visualization pipeline

Large volumes of simulation data generated in scientific applications need to be appropriately retrieved and mapped onto a 2D display device to be "visualized" by human operators. This visualization process involves several steps that form the so-called visualization pipeline or visualization network [21]. Fig. 3 shows a high-level general abstraction of a visualization and steering pipeline along with the data and control flow.

In many scientific applications, the raw data usually takes a multivariate format and is organized in structures such as CDF, HDF, and NetCDF [10, 11, 12]. The filtering module extracts the information of interest from the raw data and performs necessary preprocessing to improve processing efficiency and save communication resources as well. The transformation module typically uses a surface fitting technique (such as isosurface extraction) to derive 3D geometries (such as polygons). The rendering module converts the transformed geometric data to pixel-based images.

During a running simulation, an end user may control the visualization and steer the computation that occurs in various processing modules along the pipeline as shown in Fig. 3. Such control or steering commands are delivered through the stable channels designed in Section 3.

4.2 Analytical model

We present a mathematical model in Fig. 4 for the general pipeline shown in Fig. 3. Here, the visualization pipeline consists of $n+1$ sequential modules, $M_1, M_2, \ldots, M_{u-1}, M_u, \ldots, M_{v-1}, \ldots\ldots, M_w, \ldots, M_{x-1}, M_x, \ldots, M_{n+1}$, 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 the 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 of transmitting a large message of size m given by $m/b_{i,j}$. The transport network is represented by a graph $G = (V, E), |V| = k+1$, where V denotes the set of nodes (vertices) and E denotes the set of links (edges). The transport network 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 transport 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 transport path P. The data flow into a group is

[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.

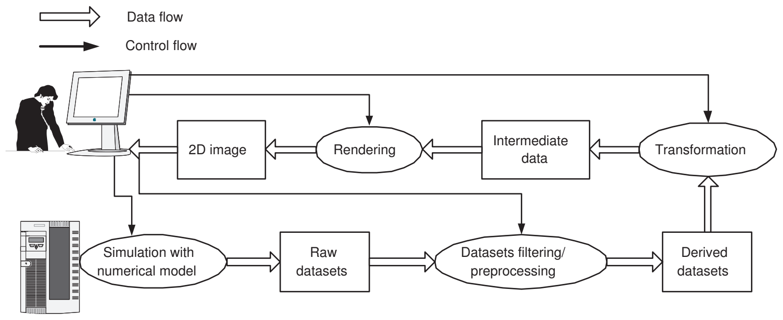


Figure 3. A general visualization steering pipeline: visualization modules, data flow, and control flow.

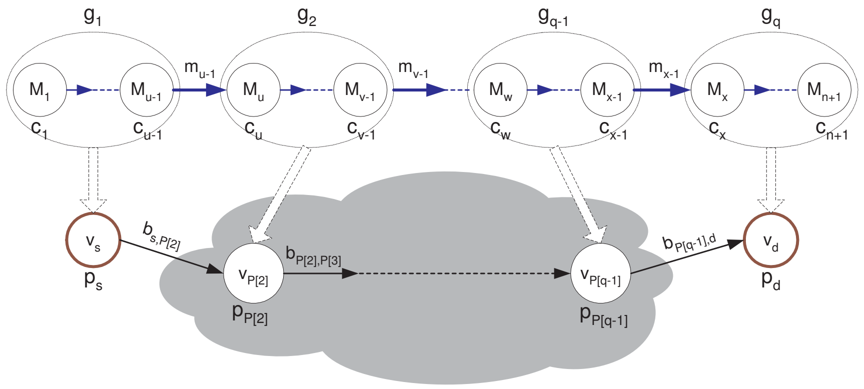


Figure 4. Mathematical model for pipeline partitioning and network mapping.

the one produced by the last module in the upstream group; for example in Fig. 4, 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 control messages is typically on 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 interactivity. The need for higher interactivity is equivalent to minimizing the *end-to-end delay* given by:

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

Thus, our goal is to minimize the total time incurred on the forward links from the source node to the destination node to achieve the fastest response for each simulation dataset. Note that in Eq. 2, 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 the simplest client-server setup.

4.3 Bandwidth measurement for transport time estimation

We present a linear regression model to estimate the bandwidth of a transport path using active traffic measurement based on [29]. Due to complex traffic distribution over wide-area networks and the non-linear nature of transport protocol dynamics (in particular TCP), the throughput achieved in actual message transfers is typically different from both the link and available bandwidths, and typically contains a random component. We consider the *effective path bandwidth* (EPB) as the throughput achieved by a flow using a given transport module under certain cross traffic conditions. The notion of effective path bandwidth is specific to the transport protocol employed by the transport daemon. The active measurement technique we apply here is to estimate the effective path bandwidth and minimum delay for each virtual link. Note that a virtual link in the overlay network of transport daemons may correspond to a multi-hop data path in wide-area networks, which usually consists of multiple underlying physical links from different networks.

There are three main types of delays involved in the message transmission over computer networks, namely, link propagation delay d_p imposed at the physical layer level, equipment-associated delay d_q mostly incurred by processing and buffering at the hosts and routers, and bandwidth-constrained delay d_{BW}. The delay d_q often experiences a high level of randomness in the presence of time-varying cross traffic and host loads. Also, since the transport protocol reacts to the competing traffic on the links, the delay d_{BW} may also exhibits randomness particularly over congested wide-area connections. We use Eq. 3 to measure the end-to-end delay in transmitting a message of size r on a path P with l physical links:

$$d(P,r) = d_{BW}(P,r) + \sum_{i=1}^{l} \left(d_{p,i}(P) + d_{q,i}(P,r)\right) \quad (3)$$

Due to the large size of data transfer in high-performance visualization applications, only the first term of Eq. 3 is significant and therefore the delay $d(P,r)$ of transmitting a message of size r along path P can be approximated by a linear model: $d(P,r) \approx r/EPB(P)$. 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 the end-to-end delays, on which we apply a linear regression to estimate the EPB.

4.4 Visualization module performance estimation

The time of performing a visualization task depends on various factors such as the available system resources, data size, visualization method, and user-specified parameters. The dynamic and input-dependent feature of some factors poses a great challenge on the performance estimation. For example, the time of extracting isosurfaces from a dataset is closely related to the number of extracted triangles that cannot be predicted before the user selects an isovalue. In addition, the intrinsic feature of a visualization technique also plays an important role, thus the performance estimation for isosurface extraction could be very different from the one for streamline generation. In this paper, we design a different performance estimation method using both analytical model and statistical measurements for each of the common visualization techniques. With reasonable preprocessing overheads, our models provide quick and accurate run-time estimates of processing times. Due to the limited paper space, here we restrict our discussions to three popular techniques: isosurface extraction, ray casting, and streamline.

4.4.1 Isosurface extraction

Traditionally, to speed up the search process, one typically traverses an octree to identify data blocks containing isosurfaces. In this case, the extraction is performed at the block level. To be general, the time to extract isosurfaces from a dataset is determined by the number of blocks containing isosurfaces, n_{blocks}, the number of cells in a block, S_{block}, and the average time of extracting isosurfaces from a block, t_{block}, which depends on S_{block}. We define the performance model for isosurface extraction as:

$$t_{extraction}(n_{blocks}, S_{block}) = n_{blocks} \times t_{block}(S_{block}). \quad (4)$$

In this model, n_{blocks} and S_{block} depend directly on the data partitioning method, which is usually known beforehand. However, since t_{block} is also controlled by the isovalue selected by the user at run time, it is difficult to provide an exact expression relating t_{block} to the other parameters. We employ a statistical method to predict the isosurface extraction time t_{block}. A set of testing datasets are sampled from various applications. With different block sizes, we first run the isosurface extraction algorithm on these datasets with a large number of possible isovalues, and for each of 15 cases including the one with no isosurface, mark down the frequency of the related cells found inside a block as well as the time spent on each case, $T_{Case}(i)$ where $i \in [0, 14]$. We then average the numbers collected for each case and use it as the case probability, $P_{Case}(i)$. At run time, we can estimate the average time spent on a block using the following equation:

$$t_{block}(S_{block}) = S_{block} \times \sum_{i=0}^{14} \left(T_{Case}(i) \times P_{Case}(i)\right), \quad (5)$$

which is constant for blocks with the same S_{block}.

For isosurface extraction, we also need to estimate the rendering cost that is controlled by the number of extracted triangles, $n_{triangles}$, and the number of triangles the graphics card can render per second. Since $n_{triangles}$ can be computed from S_{block}, $P_{Case}(i)$, and the number of triangles extracted from a cell with case i, $n_{triangle}(i)$, the performance model for rendering isosurfaces is defined as:

$$\begin{aligned} &t_{rendering}(n_{blocks}, S_{block}) \\ &= n_{blocks} \times S_{block} \times \textstyle\sum_{i=0}^{14} \left(n_{triangle}(i) \times P_{Case}(i)\right). \end{aligned} \quad (6)$$

4.4.2 Ray casting

Similar to isosurface extraction, to be general, we assume ray casting is also performed in the block level. The performance estimation for ray casting is much harder than the one for isosurface extraction because of unlimited possibilities of underlying transfer functions. The time spent on casting rays through a block is controlled by the number of rays, n_{rays}, the average number of samples per ray, $n_{samples}$, and the computing time spent on each sample, t_{sample}. Therefore, our performance model for ray casting is defined as:

$$t_{raycasting} = n_{blocks} \times n_{rays} \times n_{samples} \times t_{sample}, \quad (7)$$

where n_{blocks} is the number of nonempty blocks. Because of the unpredictable transfer function, we simplify our estimation by not considering early ray termination inside a block, aiming to provide the quantitative measurement of the computing power supported by available computing facilities. Now n_{rays} and $n_{samples}$ only depend on the viewing vector, thus is constant for a given view if orthographic projection is used. t_{sample} can be considered as constant and can be easily computed by running ray casting algorithm on a test dataset for each machine. Such estimation would be more accurate if each non-empty block is semi-transparent.

4.4.3 Streamline

Compared with isosurface extraction and ray casting, the performance estimation for the streamline algorithm is much simpler. The time needed for generating streamlines is dominated by the number of seed points, n_{seeds}, and the number of advection steps for each streamline, n_{steps}. Hence, its performance model is defined as

$$t_{streamline}(n_{seeds}, n_{steps}) = n_{seeds} \times n_{steps} \times T_{advection}, \quad (8)$$

where $T_{advection}$ is the time required to perform one advection, which is computed by running the streamline algorithm on a test data set and recording the time spent for each advection. For each computing machine, we can find an average $T_{advection}$ that will be used for run-time performance estimation.

4.5 Optimizing visualization pipeline using dynamic programming

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,

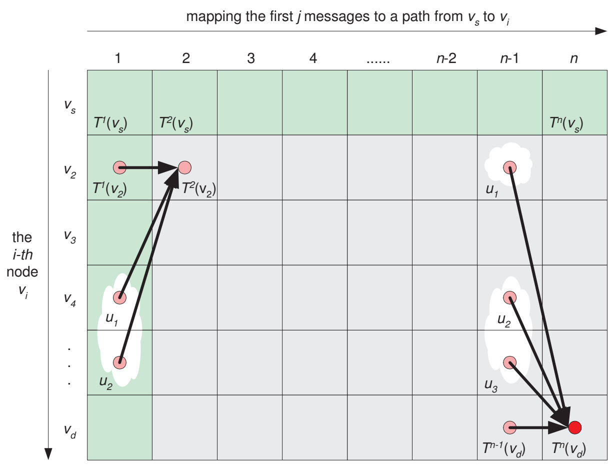


Figure 5. Construction of 2D matrix in dynamic programming.

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 $T^n(v_d)$ [30], for $j = 2, \ldots, n$, $v_i \in V$:

$$\begin{aligned} &T^j(v_i) \\ &= \min \left(\begin{array}{l} T^{j-1}(v_i) + \frac{c_{j+1} m_j}{p_{v_i}}, \\ \min_{u \in adj(v_i)} \left(T^{j-1}(u) + \frac{c_{j+1} m_j}{p_{v_i}} + \frac{m_j}{b_{u,v_i}} \right) \end{array} \right) \end{aligned} \quad (9)$$

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

$$T^1(v_i) = \begin{cases} \frac{c_2 m_1}{p_{v_i}} + \frac{m_1}{b_{v_s,v_i}}, & \forall e_{v_s,v_i} \in E \\ \infty, & otherwise, \end{cases} \quad (10)$$

as shown on the first column in the 2D matrix in Fig. 5.

In Eq. 9, 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 nodes of node v_i is enclosed in the shaded area in Fig. 5. 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.

It is worth pointing out that some additional constraints may arise in practical applications. For example, some nodes are only capable of executing certain visualization modules. Such constraints can be conveniently handled by imposing feasibility checks at each step of the dynamic programming recursions: the scenario with failed feasibility check is simply discarded. This algorithm uses data transport and processing times of various subtasks as input parameters. We developed separate cost models to reliably estimate processing times of volume visualization algorithms, including isosurface extraction and raycasting, as well as to predict network transport times. These time estimation methods are not discussed here due to space limit.

5 Implementation and Experimental Results

We implemented a proof-of-concept system for RICSA in Java, C++, and Fortran on Linux operating system using Google Web Toolkit (GWT) [17] for the Ajax web development at user ends. In this section, we describe the implementation details and present experimental results in Internet deployments.

5.1 Graphical user interface

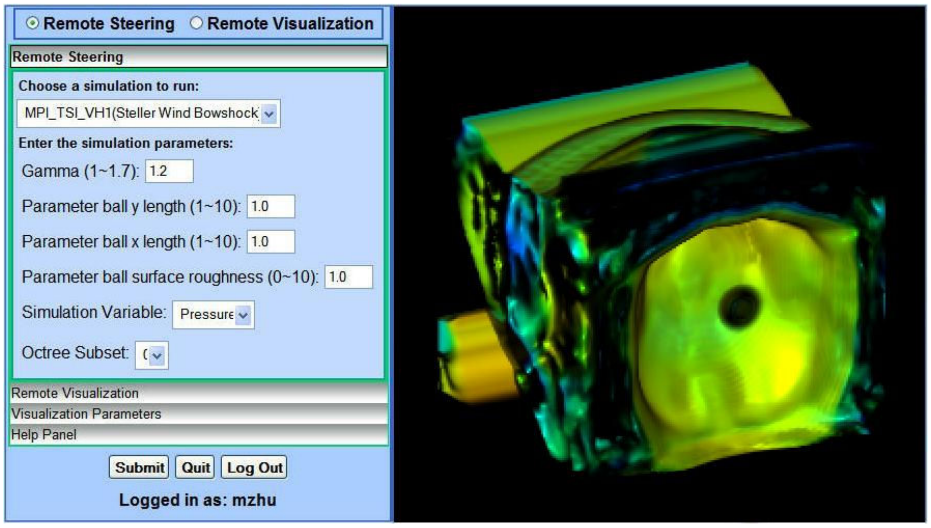
Fig. 6 displays a screenshot of the graphical user interface of RICSA developed using GWT. The Sod shock tube simulation, a classical hydrodynamics problem, is running on a Linux cluster of eight nodes for parallel computation and visualization. Each newly generated simulation dataset traverses through a linear visualization pipeline over the wide-area network with minimum end-to-end delay to achieve fast user interactivity. Marching cubes algorithm is used to extract isosurfaces from the raw dataset. By interacting with the web components in a browser, a user can choose from a list of available simulation codes to run an appropriate computation, specify computation control parameters, and select visualization operation parameters such as the variable of interest, one of the eight octree subsets or entire dataset, visualization technique, and zoom factor and rotation angle. Direct mouse interactions with the image will also trigger image rotation actions. While the simulation is running, the user can dynamically steer computational parameters based on visual feedback. When a new image arrives at the client side, only the image component in the browser is updated and the rest of the web components remain unchanged. Such a data-driven model from Ajax technology makes web applications more responsive. In addition to real-time simulation programs, RICSA can also support remote visualization for archival datasets with a different set of user input components.

```
RICSA_StartupSimulationServer();
RICSA_WaitAcceptConnection();
do RICSA_ReceiveHandleMessage();
while(Message Not SimulationReq)

Begin by reading input deck...;
Check arrays are large enough for desired number of physical zones;
Open history file and write out a description of the run;
Initialize variables for new problem;
Restart from old dump file to save time if necessary;
Increment dump filename;

//The following is the  main computational loop
do
{
        sweepx;
        sweepy;
        sweepz;
        RICSA_PushDataToVizNode();
        RICSA_ReceiveHandleMessage();
        if (Message is NewSimulationParameters)
                RICSA_UpdateSimulationParameters();
}while(Cycle Not EndCycle)
```

Figure 7. Visualization and network API function calls are inserted into Virginia Hydrodynamics simulation program.

5.2 Universal steering framework for various simulation programs

RICSA is designed as a universal framework to support various simulation programs possibly written in different programming languages. It requires a minimum amount of effort to modify the original simulation programs for integration with RICSA. We achieved this goal by developing several generic C++ visualization/network API functions and packaging them in a shared library. These API function calls are inserted at certain points in the simulation code written in various programming languages to set up socket communications, transfer datasets, or intercept steering commands from the client. Such an implementation structure facilitates modular programming and improves code portability. The pseudo code in Fig. 7 demonstrates how six essential API functions from RICSA are called at the appropriate locations in the computational loops of Virginia Hydrodynamics (VH1) simulation code written in Fortran [14].


[✓ Remote Steering] Remote Visualization

Remote Steering

Choose a simulation to run:

MPI_TSI_VH1(Steller Wind Bowshock

Enter the simulation parameters:

Gamma (1~1.7): 1.2

Parameter ball y length (1~10): 1.0

Parameter ball x length (1~10): 1.0

Parameter ball surface roughness (0~10): 1.0

Simulation Variable: Pressure

Octree Subset: (

Logged in as: mzhu

Figure 6. A web snapshot of the pressure animation of stellar wind bowshock on a cluster.

5.3 Remote visualization experimental results

We deployed the RICSA system on six Internet nodes located at Oak Ridge National Laboratory (ORNL), Louisiana State University (LSU), University of Tennessee at Knoxville (UT), North Carolina State University (NCState), Ohio State University (OSU), and Georgia Institute of Technology (GaTech), respectively. Among them, the nodes at UT and NCState are clusters with high-performance parallel computing capabilities, while the rest of the nodes are PC Linux hosts with common hardware and software configurations. We run the Ajax client and front end at ORNL, CM node at LSU, two DS nodes at OSU and GaTech, and two CS nodes at UT and NCState. The network configuration of the distributed visualization experiment is shown in Fig. 8.

We wish to visualize at ORNL (client) three pre-generated experimental datasets, namely, Jet data of 16 MBytes , Rage data of 64 MBytes, and Visible Woman data of 108 MBytes[2], which are duplicated at OSU and GaTech. Note that the data objects, network connections, and host computing resources determine whether or not to use the MPI-based visualization modules installed on the clusters at either UT or NCState. For streaming applications, each new dataset is treated as a pre-generated dataset, which consequently undergoes the same remote visualization process. In general, the simulation does not proceed until the image from the last time step is delivered to the end user; otherwise, several datasets may have been gen-

[2]Due to limited available system resources, the visible woman dataset is downsampled from its original size by 8 times.

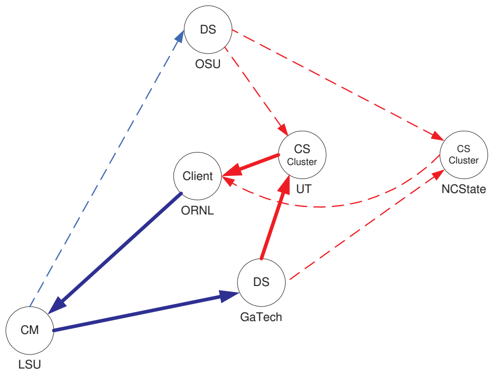


Figure 8. Network configuration of distributed visualization experiment.

erated under the previous parameter setting after the steering request is submitted, hence undermining the purpose of on-line steering. The entire steering task is essentially composed of a series of data generation processes and subsequent remote visualization processes. The time needed to generate a simulation dataset is mainly determined by the problem size and capacity of the simulation node.

5.3.1 Performance comparisons among RICSA loops

We wish to demonstrate that the optimal visualization loop chosen by RICSA outperforms all other alternatives in terms of end-to-end delay. We first perform a statistical analysis on transport measurements to estimate network bandwidths and on visualization measurements to estimate processing times. Using these estimates in the dynamic programming equations in Eqs 9 and 10, we compute the visualization routing table for each dataset. The optimal visualization pipeline GaTech-UT-ORNL is shown using solid lines in Fig. 8, wherein GaTech is used as a data storage node and UT is used as a computing service node. Together with the control path ORNL-LSU-GaTech that carries the control messages, this solution forms the closed optimal loop ORNL-LSU-GaTech-UT-ORNL.

According to the experiment network configuration shown in Fig. 8, there are three other possible visualization loops using intermediate MPI-based visualization modules and two conventional PC-PC (client/server) visualization loops. For comparison purposes, we partitioned the same visualization pipeline in a similar way and mapped it onto each of these loops. Particularly, in the PC-PC experiments, since neither the GaTech host nor the OSU host is equipped with a graphics card, we performed isosurface extraction on these two hosts acting as both a data source and a computing service node, and isosurface rendering on the ORNL host acting as both a client and a computing service node. The average measurements of the end-to-end delay experienced by all these visualization loops using the isosurface extraction technique with an identical set of parameters are shown in Fig. 9 for a visual comparison.

The differences in these end-to-end delay measurements are mainly caused by the disparities in the computing power of the selected nodes (including the rendering capability of the client node in the PC-PC cases) and the bandwidths of the corresponding network links connecting them. The performance comparisons clearly show that the optimal visualization loop ORNL-LSU-GaTech-UT-ORNL computed by our algorithm provided substantial performance enhancements over other pipeline configurations. We observed that the optimal loop achieved more than three times speedup over a default server/client mode when visualizing a dataset of about 100 MBytes. Such performance improvements are expected to increase more significantly and rapidly when the sizes of datasets continue to grow.

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. 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.

5.3.2 Performance comparisons between RICSA and ParaView

RICSA is implemented using a message-driven programming model and a state machine-based methodology that enable self-adaptive pipeline configurations on intermediate nodes. In addition, we developed a framework to efficiently compute an optimal configuration using dynamic programming based on reliable underlying cost models. The complexity of the dynamic programming process is in polynomial time, which guarantees high efficiency and scalability for complex visualization pipelines and large network sizes.

The optimization algorithm that minimizes the end-to-end delay of a visualization pipeline can be leveraged by existing remote visualization systems. To show that our system has a lightweight implementation with relatively small message-based control overhead, we ran ParaView on the same optimal network configuration of the visualization pipeline as determined by the optimization algorithm for the same visualization job on the identical datasets.

Specifically, our experiments involved running *pvdataserver* on the DS node at GaTech, *pvrenderserver* (executed by mpirun using the same four processing nodes utilized by RICSA) on the cluster-based CS node at UT, and *pvclient* at ORNL. Note that the CM node at LSU was not involved because ParaView does not yet employ such additional nodes. The end-to-end delay measurements averaged over multiple runs using the same configuration for both ParaView and RICSA are illustrated together in Fig. 10.

We observed that for the same tasks, RICSA achieved comparable performances with ParaView. The performance differences may have been caused by higher processing and communication overhead incurred by visualization and network transfer functions used in ParaView. Instead of using third-party visualization packages, we developed and implemented our own lightweight visualization modules for RICSA. It is, however, not our intention to compare the efficiency of our visualization modules with those of other existing systems. The main difference of these two systems is that in ParaView, the mapping from the visualization pipeline to the network nodes is manually performed as an initial setup, while in our system, the initial configuration is automatically computed using dynamic programming by the CM node and the mapping scheme is adaptively re-configured during runtime in response to drastic network or host condition changes [3].

[3]In our tests, a new visualization routing table was computed for each subsequent interactive operation but the system re-configuration was not triggered due to the stable network and host conditions during the experiment.

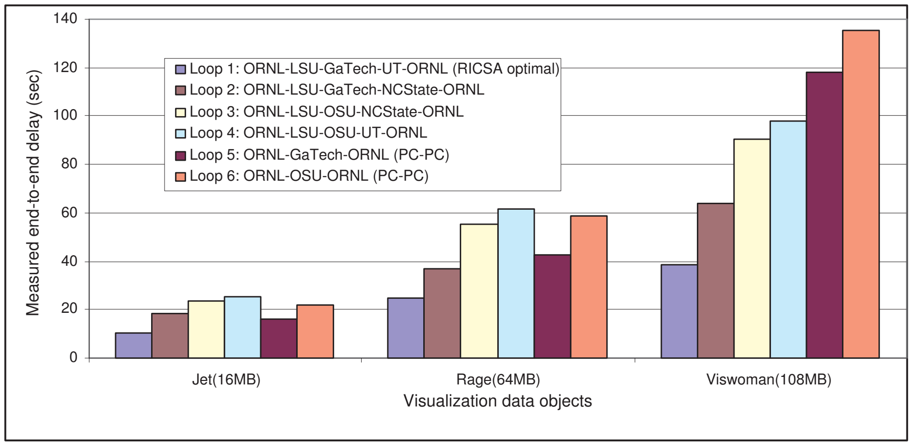


Figure 9. Performance comparisons between different visualization loops: Loop 1 along ORNL-LSU-GaTech-UT-ORNL (optimal), Loop 2 along ORNL-LSU-GaTech-NCState-ORNL, Loop 3 along ORNL-LSU-OSU-NCState-ORNL, Loop 4 along ORNL-LSU-OSU-UT-ORNL, Loop 5 along ORNL-GaTech-ORNL (PC-PC), and Loop 6 along ORNL-OSU-ORNL (PC-PC).

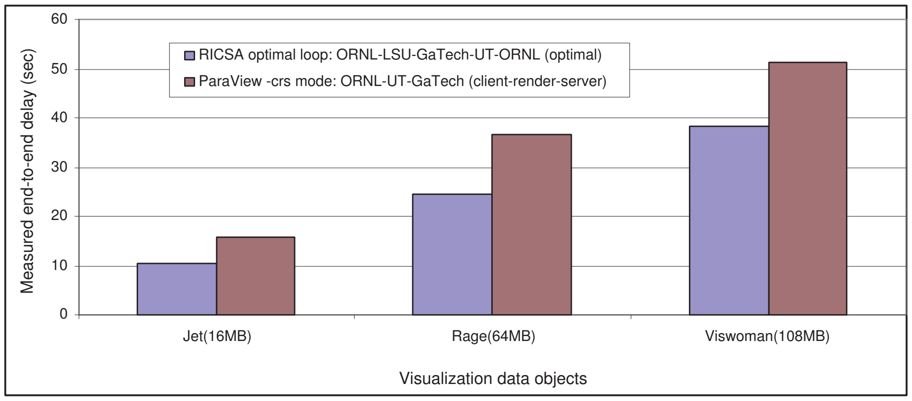


Figure 10. Performance comparisons between different visualization loops: RICSA optimal loop along ORNL-LSU-GaTech-UT-ORNL, ParaView -crs mode along ORNL-UT-GaTech.

6 Conclusion and Future Work

We proposed a distributed system for visualizing, monitoring and steering simulations over wide-area networks. We presented a new transport protocol for stable control channels and also a mathematical model for mapping a visualization pipeline to networks. We proposed a dynamic programming-based approach to compute an optimal visualization pipeline configuration that minimizes the end-to-end delay of a remote visualization and steering system.

Compared with existing systems, RICSA is lightweight without requiring the installation of any third-party packages. It is highly accessible since users can access the system using a web browser with Web 2.0 technology supported. However, in order to utilize network resources for optimal network performance, efficient and accurate performance estimation daemons need to be deployed to dynamically monitor and measure network and host conditions. These daemons must work seamlessly with the main system to provide accurate performance estimations, which could impose a great challenge on implementation.

It would be of our future interest to study various formulations of this class of optimization problems from the viewpoint of computational criteria and practical implementations. We plan to integrate RICSA with various large-scale simulation programs from different disciplines such as biology, chemistry, and physics. Collaborative visualization and steering will be supported to enable team work within a group of geographically distributed users. In addition to the Internet, we will deploy the system over dedicated networks, such as DOE UltraScience Net [15], for experimental testing especially for large datasets. New transport methods will be incorporated to overcome the limitations of default TCP or UDP in terms of throughput, stability, and dynamics in our remote visualization and steering system at a later stage.

Acknowledgments

This research is partially sponsored by National Science Foundation under Grant No. CNS-0721980 and Oak Ridge National Laboratory, U.S. Department of Energy, under Contract No. PO 4000056349 with University of Memphis.

References

[1] Terascale Supernova Initiative. http://www.phy.ornl.gov/tsi.

[2] Terascale High-Fidelity Simulations of Turbulent Combustion with Detailed Chemistry. http://scidac.psc.edu.

[3] ParaView. http://www.paraview.org.

[4] ASPECT. http://www.aspect-sdm.org.

[5] RAVE. http://www.wesc.ac.uk/projectsite/rave/index.html.

[6] CUMULVS. http://www.csm.ornl.gov/cs/cumulvs.html.

[7] RealityGrid. http://www.sve.man.ac.uk/Research/AtoZ/RealityGrid.

[8] PVM. http://www.csm.ornl.gov/pvm.

[9] AVS. http://www.avs.com.

[10] CDF: Common Data Format. http://nssdc.gsfc.nasa.gov/cdf.

[11] HDF: Hierarchical Data Format. http://hdf.ncsa.uiuc.edu.

[12] NetCDF: Network Common Data Form. http://my.unidata.ucar.edu/content/software/netcdf.

[13] Network Weather Service. http://nws.cs.ucsb.edu.

[14] VH1. http://wonka.physics.ncsu.edu/pub/VH-1.

[15] DOE UltraScienceNet. http://www.csm.ornl.gov/ultranet.

[16] Computational engineering international. http://www.ceintl.com/products/ensight.html.

[17] Gwt. http://code.google.com/webtoolkit.

[18] 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.

[19] I. Bowman, J. Shalf, K. 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.

[20] K. Brodlie, D. Duce, J. Gallop, M. Sagar, J. Walton, and J. Wood. Visualization in grid computing environments. In *Proc. of IEEE Visualization*, pages 155–162, 2004.

[21] A. Kaufman. *Trends in visualization and volume graphics, Scientific Visualization Adv ances and Challenges*. IEEE Computer Society Press, 1994.

[22] H. J. Kushner and D. S. Clark. *Stochastic Approximation Methods for Constrained and Unconstrained Systems*. Springer-Verlag, 1978.

[23] E. Luke and C. Hansen. Semotus visum: a flexible remote visualization framework. In *Proc. of IEEE Visualization*, pages 61–68, 2002.

[24] S. R. M. Oberhuber and A. Bode. Tuning parallel programs with computational steering and controlled execution. In *In Proceedings of the Thirty-First Hawaii International Conference on System Sciences*, volume 7, pages 157–166, 1998.

[25] A. Mezzacappa. Scidac 2005: Scientific discovery through advanced computing. *Journal of Physics: Conference Series*, 16, 2005.

[26] N. S. V. Rao, Q. Wu, and S. S. Iyengar. On throughput stabilization of network transport. *IEEE Communications Letters*, 8(1):66–68, 2004.

[27] D. W. S.G. Parker and C. Johnson. The scirun computational steering software system. In *Modern Software Tools in Scientific Computing*, pages 1–44, 1997.

[28] S. Stegmaier, M. Magallon, and T. Ertl. A generic solution for hardware-accelerated remote visualization. In *Proc. of Data Visualisation*, pages 87–95, 2002.

[29] 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.

[30] 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*, pages 402–405, Hong Kong, China, December 18-20, 2004.