International Symposium on Electronic Imaging , January 2005,
San Jose, California

On optimal mapping of visualization pipeline onto linear arrangement of network nodes

Mengxia Zhu[a], Qishi Wu[b], Nageswara S. V. Rao[b], S. Sitharama Iyengar[a*]

[a] Department of Computer Science, Louisiana State University, Baton Rouge, LA, USA 70803;

[b] Computer Sci. and Math. Division, Oak Ridge National Laboratory, Oak Ridge, TN, USA 37831

ABSTRACT

This paper discusses algorithmic and implementation issues of optimally mapping a visualization pipeline onto a linear arrangement of wide-area network nodes to minimize the total delay. The first network node typically is a data source, the last node could be a display device ranging from a personal computer to a powerwall, and each intermediate node could be a workstation or computational cluster. This mapping scheme appropriately distributes the filtering, geometry generation, rendering, and display modules of the visualization pipeline to the linear arrangement of network nodes to make efficient use of the computing resources at end nodes and also the network bandwidth between them. A regression based network daemon is developed to measure the available bandwidth on a transport link. We present an analytical formulation of this problem by taking into account the computational power of nodes, the bandwidths between them, and the sizes of messages exchanged between visualization modules. We propose a polynomial-time optimal algorithm that uses the dynamic programming method to compute the mapping with the minimum total delay. An OpenGL-based remote visualization system is implemented and deployed at three geographically distributed nodes for preliminary experiments.

Keywords: Remote visualization, visualization pipeline, bandwidth measurement, network mapping

1. INTRODUCTION

A remote visualization system can potentially enable an end user equipped with a simple display device and network access to visualize large volumes of scientific data that resides at powerful remote visualization servers. Such large data sets are difficult to be generated or stored at end user nodes, which typically have limited computing and storage resources. A simple version of a remote visualization system consists of a remote data source acting as a server, a local rendering/display terminal acting as a client, zero or more intermediate hosts, and a network connecting them all together. The overall performance of such a remote visualization system critically depends on how efficiently its visualization pipeline is mapped onto the network nodes. In particular, this mapping determines the execution times of various modules and the transmission times of data exchanged between them, thereby deciding the total time of the visualization pipeline. In this paper, we address both analytical and implementation aspects of realizing an optimal mapping of a visualization pipeline when the network nodes form a linear arrangement.

Many existing remote visualization systems [11,12,13] employ a predetermined partition of the visualization pipeline and typically send fixed-type data streams such as raw volume data, filtered data, geometric objects, or frame buffer (FB) to remote end nodes for visualization. In general, the task of a client node can be quite varied depending on the data sent by its predecessor module in the pipeline: it can range from geometry generation from raw data, to rendering OpenGL commands, to displaying frame buffers. In some commercial client-server visualization systems, the server typically generates the geometry (usually in a vendor-specific format) and sends it to the client, where it is rendered and displayed. While in others, OpenGL commands are sent from the server to the client that in turn "interprets" them. While such schemes are common, they are not always optimal for high performance visualizations that typically deal with large data sizes and complex data structures. Particularly over wide-area connections, this problem is further compounded by

* Further author information: (Send correspondence to Nageswara S. V. Rao)
Mengxia Zhu: Email: zhum@ornl.gov; Telephone: 1 865 576-7210
Qishi Wu: Email: wuqn@ornl.gov; Telephone: 1 865 576-5603
Nageswara S. V. Rao: Email: raons@ornl.gov; Telephone: 1 865 574-7517
S. Sitharama Iyengar: Email: iyengar@csc.lsu.edu; Telephone: 1 225 578-1252

limited network bandwidths and time-varying dynamics of network conditions. In certain data sets, the geometry is much simpler than the data itself, in which case it is appropriate to ship the geometry across the network. On the other hand, for fractal data sets, the geometry is much more complicated than the data, and hence it is more efficient to ship the data rather than the geometry. Such variations may occur in data sets generated even in a single domain such as supernova computations [17]. In general, the network bandwidths and exchange data sizes together with processing times must be taken into account to optimally map a visualization pipeline onto network nodes.

Considerable research efforts have been underway in designing flexible remote visualization systems that distribute visualization subtasks across network nodes. Bowman *et al* [8] proposed a performance prediction framework to predict processing times of visualization modules using linear models and network bandwidth using Network Weather Service [18]; these predictions are used to obtain a suitable mapping of the visualization pipeline. Luke *et al* [9] constructed Semotus Visum, a flexible remote visualization framework capable of multiple partition scenarios. They run their tests using different partition schemes on a local network and the corresponding performances are measured [9]. ARTE [14] implements an adaptive delivery of 3D model by keeping track of the available client, server and network resources. The original 3D data is converted to selected modalities that are best suited for rendering and transmission [14]. In these works, the mapping of visualization modules to network nodes is accomplished by empirical testing and manual configuration.

We analytically formulate a problem of optimizing the total delay of a visualization pipeline by considering the times for transmission, computation and rendering. This formulation enables us to analyze the algorithmic aspects and optimality of mapping the visualization pipeline onto a given linear arrangement of wide-area network nodes. We consider a simple case where each node is capable of executing any visualization module with a special role played by the end nodes. The first network node is a data source, and can also perform other tasks such as filtering, geometry generation, or rendering. The last node is capable of displaying the frame buffer but can also perform other visualization subtasks; it could be equipped with a simple terminal or a more sophisticated device such as a powerwall or a tiled display. The intermediate nodes are capable of performing all visualization subtasks, and they could be workstations, computational clusters, or custom rendering engines.

This characterization is a simplified abstraction of real remote visualization systems wherein the nodes often have specialized functionalities such as high-performance storage, cluster-based rendering and tiled displays. Despite this simplification, this model highlights the inherent computational aspects of realizing an optimal mapping. An optimal mapping must make efficient use of the computing resources at the nodes and also the network bandwidths between them to allocate the modules such as filtering, geometry generation, rendering, and display, to various nodes. We present a polynomial-time optimal solution using dynamic programming to compute a mapping with the minimum total delay considering processing times and connection bandwidths. The computational complexity of the solution is $O(n \times k)$, where $n+1$ is the number of visualization modules and $k+1$ is the number of network nodes in a linear arrangement. We implemented an OpenGL-based remote visualization system and deployed it at three geographically distributed nodes. Some preliminary experiments are run on this client/server-based remote visualization system.

The rest of the paper is organized as follows. In Section 2, we describe a simple visualization pipeline and a remote visualization framework, which form the basis for our analytical model. In Section 3, we first describe a bandwidth measurement method based on [5], and then present our partition schemes using dynamic programming-based algorithms for total delay minimization. Implementation details and test results are provided in Section 4. We finally conclude our work in Section 5.

2. REMOTE VISUALIZATION SYSTEM

2.1. Visualization pipeline

The large volumes of data generated in scientific or medical applications have 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* [1]. Fig. 1 shows a simple visualization pipeline along with the flow of data produced at each pipeline module. In many scientific applications, the raw data usually takes a multivariate format and is organized in structures such as CDF, NetCDF, and HDF [2,3,4]. The *filtering* module extracts the information of interest from the data source in order to improve processing efficiency and save communication resources as well. The decision on rendering techniques to be employed is first made in the transformation module,

which typically uses a surface fitting technique (such as isosurface extraction) to derive 3D geometries (such as polygons), or uses transfer functions to perform color and opacity classifications for each voxel based on its attribute. Rendering module converts the transformed geometric or volumetric data in 3D view coordinates to a pixel-based image in 2D screen coordinates. In most of the existing graphics systems, the visual properties of a raster image such as color, opacity, and depth is stored and carried in a frame buffer for final display on a display device.

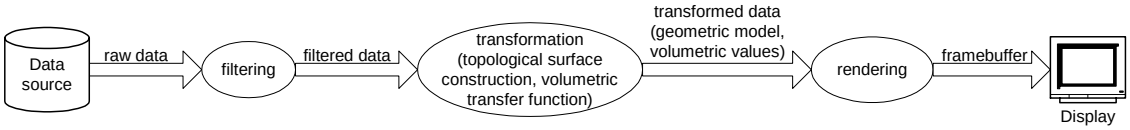

Figure 1. A general visualization pipeline and its data flow of remote visualization system.

In the stand-alone mode, all these visualization modules reside on the same computing node. It is worthwhile pointing out that Fig. 1 is a diagrammatic description of highly abstracted visualization components; for instance, the rendering module itself might involve several stages of vertex transformations in the OpenGL environment. The visualization pipelines of real applications may significantly vary due to the disparate implementation procedures and the use of different visualization techniques.

2.2. Framework for remote visualization system design

The block diagram in Fig. 2 illustrates a baseline framework for visualization systems that employ optimal pipeline partitioning and automatic network mapping. Since many scientific applications generate terabyte or even petabyte data, which makes it very difficult to run computation-intensive visualization modules on a single desktop computer. Hence we first perform a parallel computing to explore the parallelism in the modules and the computational capability provided by the nodes with high-performance computing (HPC) resources, such as clusters. The connection bandwidths over the underlying transport network are estimated using active traffic measurements. Based on the bandwidth measurements, sizes of data to be processed, and computational complexities of visualization modules, a decomposition and mapping of the pipeline is then performed. The visualization pipeline is decomposed into groups, which are then mapped one-to-one to the computing nodes distributed in the transport network.

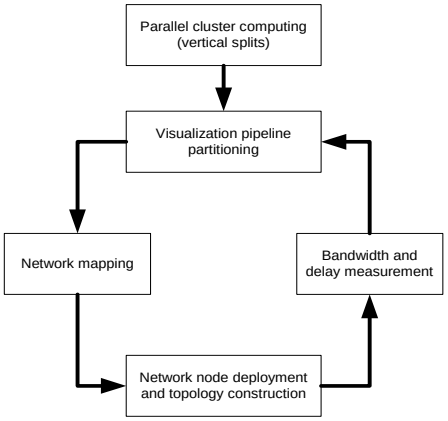

Figure 2. Architecture for remote visualization system.

The design procedure forms a closed loop as shown in Fig. 2 because the execution of each design block depends on the input from another. In a practical implementation, we generally start with computing node deployment and network topology construction. For example, the site of data source is *a priori* known, and the location of a remote client is determined whenever it initiates a ‘visualization” connection with the server. The information collected on networking and computing resources can be used to select optional intermediate nodes with specific visualization and/or computing facilities. In this paper, we consider a simple case where all intermediate nodes and their order along the visualization

pipeline are known in advance so that a surjective mapping is required. Such knowledge can be obtained from the information about the available network nodes and their capabilities. Despite the simplification, this case highlights some of the salient features of mapping the visualization pipeline, while avoiding the combinatorial complexity of choosing a suitable "ordered path" embedded in the network.

2.3. Analytical model

We now describe an analytical model for the visualization pipeline decomposition and its network mapping. The *visualization pipeline* consists of $n+1$ sequential modules, which are denoted by $M_1, M_2, \ldots, M_{u-1}, M_u, \ldots, M_{v-1}, \ldots, M_w, \ldots, M_{x-1}, M_x, \ldots, M_{n+1}$ as shown in Fig. 3. 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 its upstream module M_{j-1} and generates data of size m_j, which is then sent over a network link to its downstream module M_{j+1} for further processing. A *linear arrangement* consists of computing nodes $N_1, N_2, \ldots, N_k, N_{k+1}$, which are geographically distributed over the network. Node $N_i, i = 1, 2, \ldots, k$ with a normalized computing power† p_i is connected to its downstream neighbor node N_{i+1} with a *forward link* L_i of bandwidth b_i and minimum link delay d_i. The minimum link delay is mostly contributed by the link propagation and queuing delay, and is much smaller than the bandwidth-constrained delay in most cases.

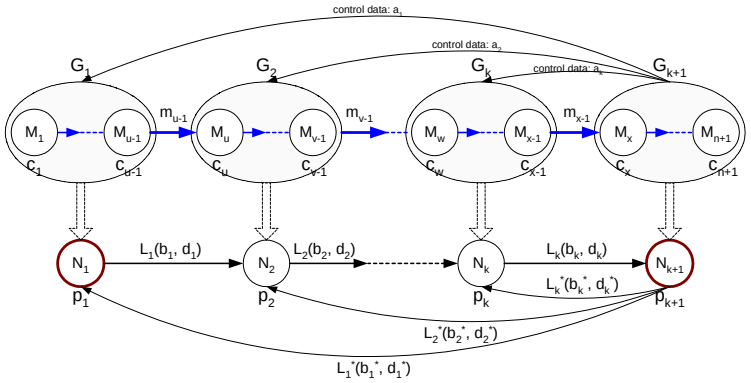

Figure 3. Pipeline partitioning and network mapping.

We decompose the visualization pipeline into *visualization groups* denoted by $G_1, G_2, \ldots, G_k, G_{k+1}$. The data flow between two adjacent groups is the one produced by the last module in the upstream group; for example in Fig. 3, we have $m(G_1) = m_{u-1}, m(G_2) = m_{v-1}, \ldots, m(G_k) = m_{x-1}$. The client residing in the destination group G_{k+1} sends control data $a_i, i = 1, 2, \ldots, k$, such as simulation parameters, filter types, visualization modes, and view parameters, to preceding visualization groups $G_i, i = 1, 2, \ldots, k$ through *backward* links $L_i^*(b_i^*, d_i^*), i = 1, 2, \ldots, k$ for interactive operations. For forward links which support large amounts of visualization data, we usually ignore the minimum link delay $d_i, i = 1, 2, \ldots, k$ occurred on the link L_i. Furthermore, the control data generally ranges from several bytes to several thousand bytes, and is often much smaller than the visualization data and therefore its transport time is also assumed to be negligible. Under these conditions, the objective of the *decomposition and mapping problem* of the visualization pipeline is to minimize the *total time* given by:

$$T = T_{computing} + T_{transport} = \sum_{i=1}^{k+1} T_{G_i} + \sum_{i=1}^{k} T_{L_i} = \sum_{i=1}^{k+1} \left(\frac{1}{p_i} \sum_{j \in G_i \text{ and } j \geq 2} \left(c_j m_{j-1} \right) \right) + \sum_{i=1}^{k} \left(\frac{m(G_i)}{b_i} \right), \quad (1)$$

†For simplicity, we use a normalized quantity to indicate a node's computing power without detailing its memory size, processing speed, and computing capabilities in different ways such as numeric and graphics.

where we also assume that the transport time between modules within one group on the same computing node is negligible.

Note that in Fig. 3, we assume that the set of computing nodes in the underlying transport network are preselected along with their order in the visualization pipeline. However, in a general model, except for the data source and the remote client, the number and order of the intermediate nodes are not always predetermined for an optimal mapping of the visualization pipeline. Indeed as shown in Fig. 2, the computer node deployment (both location and order) and visualization pipeline decomposition often strongly interact with each other during the entire system design process. The general model shown in Fig. 2 has a more complicated structure and therefore requires a more sophisticated solution. It is interesting to note that there may not be any intermediate nodes in some scenarios, which results in the simplest topology for remote visualization, i.e. one server and one client.

3. MAPPING FOR REMOTE VISUALIZATION SYSTEM

In this section, we present a linear regression model to estimate link bandwidth using active traffic measurement based on [5], and then propose an approach based on dynamic programming to solve the visualization pipeline decomposition and mapping problem.

3.1. Bandwidth measurement

The link bandwidth is the fastest rate at which data can be generated and sent along the link, while the available link bandwidth is the spare bandwidth of the link "left over" after the cross traffic. Due to complex traffic distributions over wide-area networks, and the non-linear nature of transport protocol dynamics (in particular TCP), the throughput achieved in actual message transfers is often different from both the link and available bandwidths, and typically contains a random component. The *effective path bandwidth* is defined as the throughput achieved by a flow using the given transport method under certain cross traffic conditions. The notion of effective bandwidth is specific to the employed transport protocol and is related to both link and available bandwidth perhaps in a complicated way. The active measurement technique we apply here is to estimate the effective path bandwidth based on [5]. Note that a link here may correspond to a multi-hop physical path over a wide-area network.

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} determined by the available bandwidth and data size. Due to the time-varying network cross traffic, the delay d_q often experiences a high level of randomness. Also, since the transport protocol reacts to the competing traffic on the links, the delay d_{BW} may also exhibit randomness particularly over congested wide-area connections. We have the following expression [5] for the end-to-end delay or message delay in transmitting a message of size *r* on a path *P* with *l* hops:

$$d(P,r) = d_{BW}(P,r) + \sum_{i=1}^{l} (d_{p,i}(P) + d_{q,i}(P,r)), \tag{2}$$

In this paper we consider large message sizes so that only the first term of the above equation is significant. Let *EBW(P)* denote the effective bandwidth of path *P* such that the message delay $d(P,r)$ can be approximated by the linear model:

$$d(P,r) \approx \frac{1}{EBW(P)} r, \tag{3}$$

The active measurement technique is employed to estimate the effective bandwidth of a link. A measurement node generates a set of test messages of various sizes, sends them over an outgoing link through a transport channel such as a TCP flow, and measures the corresponding end-to-end delays. This process is repeated several times for each message size and the average delay is calculated. Once the average end-to-end delays for messages of different sizes are determined on an outgoing link, we apply a linear regression to fit the measured points [5]. The message delay measurements between two transport daemons deployed at Louisiana State University (LSU) and Oak Ridge National Laboratory (ORNL) as well as its corresponding linear regression estimate are illustrated in Fig. 4. The measurement of the end-to-end delay for each message size is carried out three times. From this figure, we estimate that the effective path

bandwidth of this virtual link to be about 1.0 Mbps. We emphasize that the measurements are collected using the same transport method that will be used by the visualization module. If measurements are collected by tools such as Iperf, or NWS [18], they must be appropriately translated into the effective bandwidth seen by the pipeline modules.

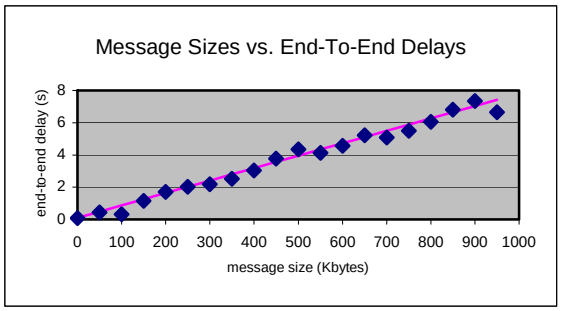

Figure 4. End-to-end message transmission delay measurements between LSU and ORNL.

3.2. Decomposition and mapping

The visualization pipeline needs to be decomposed into several groups and assigned to the preselected computer nodes so that the total delay is minimized. Intuitively, a more general version of this problem is quite similar to the classical graph clustering problem, which is NP-complete in most cases. Approximate solutions may be found by the branch-and-bound technique but in exponential time [10]. However, since our visualization system is linearly deployed on a set of computer nodes arranged in a predetermined order, we are able to solve this problem in polynomial time.

We present an algorithm to minimize both the transport and computation time as stated in Eq(1). As discussed earlier, there are *n* messages of sizes m_j flowing between the visualization modules of complexity c_j, $j = 1,2,\ldots,n$, *k* network links with bandwidths $b_i, i = 1,2,\ldots,k$, and *k*+1 processing nodes each with processing speed $p_s, s = 1,2,\ldots,k+1$. $T(i, j)$ denotes the minimal total delay with the first *j* messages (namely the first *j*+1 visualization modules) mapped onto the first *i* network links.

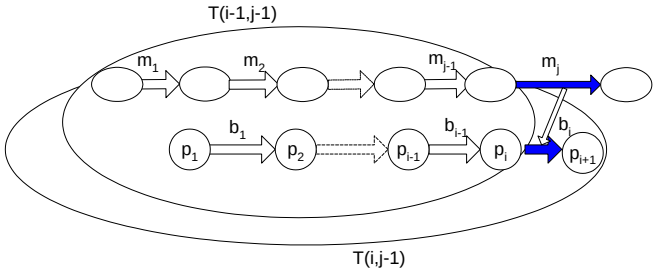

Figure 5. Dynamic programming for minimal transport delay time.

Then $T(i, j)$ can be computed recursively based on the following dynamic programming equations:

$$\underset{i=1 to k,\ j=1 to n,\ i \le j}{T(i, j)} = \min \left\{ \begin{array}{l} T(i-1, j-1) + m_j / bi + c_{j+1} m_j / p_{i+1} \\ T(i, j-1) + c_{j+1} m_j / p_{i+1} \end{array} \right\}, \qquad (4)$$

where the base conditions are computed as $T(c,c) = \sum_{i=1}^{c} m_i / b_i + \sum_{i=1}^{c} c_{i+1} m_i / p_{i+1}$ on the diagonal line and

$T(0,c) = \sum_{w=1}^{c} c_{w+1} m_w / p_1$ in the first row, $c = 1, 2, \ldots, k$. The complexity of this algorithm is in the order of $O(n \times k)$.

In Equation (4), $T(i, j)$ takes the minimal of two scenarios as illustrated in Fig. 5. In scenario 1, we map the last message m_j to the last link b_i. The transport time on this mapped link together with the computing time of the last module on the last node is then added to $T(i-1, j-1)$, which is the sub-problem of size *i*-1 and *j*-1. In scenario 2, we do not map the last message m_j on any link, which means that the last two modules are both executed on the last node, and then the problem directly reduces to the sub-problem of size *i* and *j*-1 with the addition of the computing time of the last module on the last node. Fig. 6 shows the dynamics of 2D matrix construction for computing $T(i, j)$. The entries of the diagonal line and first row are entered beforehand. In order to fill up the whole upper triangle, the calculation sweeps across the matrix from left to right and from top to bottom. For example, consider the back tracing path of $T(3,4)$. We first visit $T(2,3)$ and $T(3,3)$, of which $T(3,3)$ is already known from the base case and $T(2,3)$ is calculated from $T(2,2)$ and $T(1,2)$, which is in turn calculated from $T(1,1)$ and $T(0,1)$. Once the base cases are reached, the tracing process will stop and bounce back to its starting point. The complexity of this algorithm is $O(n \times k)$. Note that an additional matrix is needed to record the mapping scheme for the winner of each comparison along with the computation. On the diagonal line with an equal number of links and messages, we simply map all messages one-to-one onto links in a linear order. For each comparison step, the mapping scheme matrix either inherits the mapping scheme from that of $T(i, j-1)$ by adding module M_{j+1} to the last group, or appends a separate group with message *j* to the mapping scheme of $T(i-1, j-1)$.

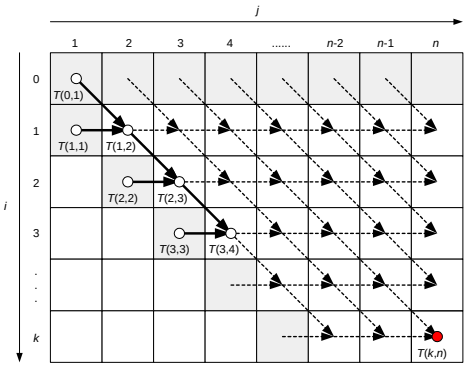

Figure 6. Construction of a 2D matrix of *T*(*i*,*j*) for dynamic programming.

4. IMPLEMENTATION AND CASE STUDY

Our remote visualization system is deployed at three nodes located at North Carolina State University (NCSU), ORNL, and LSU as shown in Fig. 7. Our parallel isosurface extraction computation is implemented on Orbitty cluster at NCSU, which consists of 23 computer nodes (total of 92 CPUs each at 2.4GHz). Linux workstations with 3GHz CPU are used as hosts at ORNL and LSU. The current data channel and control channel are built upon TCP. New transport protocol can easily be plugged in replace of the default TCP in our system.

Our cluster-based remote visualization system provides the following functionalities:

- Scalar glyphs: A dot representation is used to represent a scalar value at each position. The scales and the colors of the dots reflect the scalar values.
- Isosurfaces: Surfaces of a scalar field with the same value are produced. The surfaces are also colored according to the different isovalues.
- Vector glyphs: Arrows are used to represent the vector at each data point. The direction, magnitude, and color of an arrow are set by the corresponding vector value.

- Volume rendering: Fastvox 1.0 [15], an OpenGL-based API, is embedded into our system to perform the volume rendering using the ray casting algorithm.
- Dynamic monitor and steering: The functions provided above are post-processing procedures. This function enables client to monitor the data dynamically generated at the server end on the fly. Client end can steer the data generation by sending back control feedback parameters in the course.

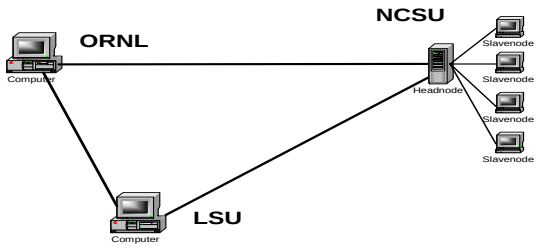

Figure 7. Remote visualization system deployments.

The user interface of our remote visualization system and a 2D pressure map generated by a hydrodynamics simulation of a supernova evolution [17] are displayed in Fig. 8.

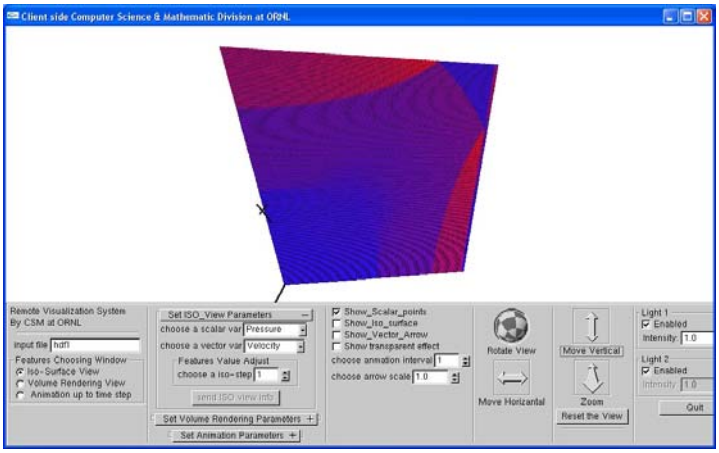

Figure 8. User interface at client end.

In this initial implementation, our system runs in a simple client/server mode without considering intermediate nodes; note, however, that partition still needs to be optimized albeit among only two network nodes since the exact data sizes exchanged between them decides the total delay. It can be viewed as a special case of linear arrangement of nodes, namely only two nodes are preselected. However, the system is capable of accepting intermediate nodes with complex network topologies using the algorithms described in Sect. 3.2. In the current deployment, the server designates proper visualization modules to the clients and calculates the estimated delay time based on the information on the visualization entity size and bandwidth estimates. The server can either send raw data, geometry data, or FB to the client. The decision is made automatically by the server to minimize the total delay. Several experiments are conducted between two hosts, one located at LSU as a server and ORNL as a client.

Since the server and client at these two locations have approximately the same computational capacity, the mapping scheme that incurs the least transport time leads to the minimal total delay. Correspondingly, the link with the minimal message size is mapped onto the network link. Table 1 illustrates the estimated transport time with different sizes of messages transmitted between the server and client. The measured bandwidth EBW and the minimum link delay denoted by *d* are dynamically measured by our network daemons deployed at LSU and ORNL. Only the one-way bandwidth and minimum delay from server to client are measured. The message size for raw data, 3D geometry and FB are estimated at

the server. The estimated transport delay represented as T_{delay} is computed as: $T_{delay} = d + \frac{Msg\_size \times 8}{EBW}$. Note that the computing times are not explicitly included in the table because they are negligible compared to the transport times in these cases.

Table 1. Partition test.

	Dimension	Estimated Bandwidth	Minimum Delay	Raw size / delay	Geometry size / delay	FB size / delay
Cube1	10x6x8	0.284Mbps	0.032sec	8 K / 0.257sec	1K / 0.032sec	1.8M/50.73sec
Cube2	50x20x39	0.300Mbps	0.034sec	610K / 16.3sec	16K / 0.46sec	1.8M/48.03sec
Cube3	150x210x139	0.277Mbps	0.033sec	71.6M / 34.4min	2.4M / 69.34sec	1.8M/52.01sec
Hand	256x256x80	0.239Mbps	0.033sec	81.9M / 45.69min	NA	1.8M/60.28sec

Case 1: The cube 1 data has a very small size of geometry and raw data. The transfer of the FB, which is much larger than either raw data or geometry data size, incurs transport delay of about 51 seconds. In this case, the server can choose to send either raw data or geometry data to client ends in less than one second.

Case 2: The cube 2 data has a larger raw data size than cube 1. The delay for sending raw data is about 16 seconds, which is not desirable for interactive operations. Since its small size of geometry data only needs less than one second of transport time, the server obviously chooses to send the geometry data instead of the raw data.

Case 3: The cube 3 data has a raw data size of 71.6 Mbytes. It would take more than half an hour for raw data transmission, which is simply not unbearable for practical use. The calculation shows that the geometry data size is similar to the FB size and either of them takes about one minute to transmit. In general, with comparable data size, sending geometry data is always preferable to sending the FB because the regeneration of FB introduces additional traffic upon the changes on the view parameters at client ends. Alternatively, server can reduce the rendering resolution level to achieve interactivity at the expense of accuracy.

Case 4: A CT scanned hand data provided by Fastvox has a raw data size of 81.9 Mbytes, which would incur unbearable transport time. Since the volume rendering is employed, we only need to decide whether to send raw data or FB to the client in this case.

In all cases above, the minimum link delay is at least an order smaller than the total delay of the visualization pipeline. In dealing with large data sets, the rendering module may be located on the server where the raw data is generated. Such a server can exploit the data locality, disk I/O bandwidth, memory, and parallel processing for accelerating the visualization pipeline. The size of frame buffer generated by the rendering process is mainly decided by the display screen width, screen height no matter how big the dataset is. A typical display window with 1000x600 pixels has a size of 1.8Mbytes, which is not a very big burden for most transport links. In case of lower bandwidth links, the display window can be scaled down to reduce the frame buffer size, and in addition, a lower resolution level can be chosen.

5. CONCLUSION AND FUTURE PLAN

In this paper, we proposed a framework and a mathematical model for an automatic mapping of a visualization pipeline onto a linear arrangement of computer nodes. Our objective is to minimize the total delay of the visualization pipeline by considering both message transport and module computations. Dynamic programming methods are proposed to compute an optimal mapping of the visualization modules assuming that we know which intermediate nodes to use and in what order to use them. Selection of suitable subset of intermediate hosts and their order of deployment from a larger set of diverse nodes on a wide-area network is an obvious and more realistic extension of our model. Nodes with high bandwidth links and high computational capacity serve as good candidates to be selected as intermediate nodes. This problem is, however, much more complex, and is akin to the NP-complete graph partitioning problems. It would be of future interest to study various formulations of this class of problems form the viewpoints of computational complexity and practical implementation.

Since the control data has a much smaller volume than the visualization data, we ignored its transport time in our

analytical formulation. However, for prompt response and accurate steering in most real-time remote collaboration applications, the control channel always imposes higher stability requirements on transport performance than the data channel. It would of future interest to explicitly include the terms corresponding to the control channels into the total delay of the pipeline. Such formulation appears to be more complex than the case studied here, and it would be interesting to see if the dynamic programming method can be extended to this case.

Our implementation takes a simple client and server form. In the future, we plan to strengthen our system to include intermediate hosts and deploy our system over dedicated networks, such as DOE UltraSceince Net [19], for experimental testing. As the datasets sizes reach terabytes, the transport delays within the pipeline could be prohibitively high even over dedicated high throughput networks. One obvious solution is to speed up the transport process. But most existing remote visualization systems (including ours) use the default TCP for both data and control message transmission. Newer transport protocols based on stochastic approximation methods for throughput stabilization and maximization [7,16] have been developed to overcome the limitations of default TCP or UDP in terms of throughput, stability and dynamics. We plan to incorporate these new transport methods in our remote visualization system at a later stage.

ACKNOWLEDGMENTS

Authors thank Professor John Blondin of North Carolina State University for providing us the code for computing the hydrodynamics of supernova and also for providing us an access to their computational and networking facilities. This research is sponsored by the High Performance Networking Program of the Office of Science, U.S. Department of Energy, under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC, the Defense Advanced Projects Research Agency under MIPR No.~K153, and by National Science Foundation under Grants No. ANI-0229969 and No. ANI-335185.

REFERENCES

1. A. Kaufman, “Trends in visualization and volume graphics”, *Scientific Visualization Advances and Challenges*, IEEE Computer Society Press, 1994.
2. http://hdf.ncsa.uiuc.edu/
3. http://nssdc.gsfc.nasa.gov/cdf/cdf_home.html
4. http://my.unidata.ucar.edu/content/software/netcdf/index.html
5. N. S.V. Rao, Y.C. Bang, S. Radhakrishnan, Q.Wu, S.S. Iyengar, and H. Cho, “NetLets: Measurement-based routing daemons for low end-to-end delays over networks”, in *Computer Communications*, **26**, no. 8, pp. 834-844, 2003.
6. G. Humphreys, M. Houston, Y. Ng, R. Frank, S. Ahern, P. Kirchner, and J.T. Klosowski, “A stream-Processing framework for interactive rendering on clusters”, in *ACM Transactions on Graphics*, **21**, pp.693-702, July 2002.
7. Q. Wu, “Control of transport dynamics in overlay networks”, *Ph.D. dissertation*, Dept of Computer Science, Louisiana State University, 2003.
8. I. Bowman, J. Shalf, and K. Ma, “Performance modeling for grid-based visualization”, submitted to Parallel Graphics and Visualization 2004.
9. E.J. Luke and C.D. Hansen, “Semotus Visum: a flexible remote visualization framework”, in *IEEE Visualization 2002, Proc.Visualization02,* pp.61-68, 2002.
10. P. Fränti, O. Virmajoki and T. Kaukoranta, "Branch-and-bound technique for solving optimal clustering", in *Int. Conf. on Pattern Recognition (ICPR'02)*, Québec, Canada, **2**, pp.232-235, August 2002.
11. http://www.ceintl.com/products/ensight.html
12. http://www.paraview.org/HTML/Index.html
13. http://www.aspect-sdm.org/
14. I.M. Boier-Martin, “Adaptive graphics”, in *IEEE Computer Graphics and Applications*, **2**, no.1, pp.6-10, Jan-Feb 2003.
15. http://www.digitalmedics.de/html/fastvox.html
16. Q. Wu, N.S.V. Rao, “A class of reliable UDP-based transport protocols based on stochastic approximation,” manuscript submitted.
17. Terascale Supernova Initiative, http://www.phy.ornl.gov/tsi
18. Network Weather Service, http://nws.cs.ucsb.edu
19. DOE UltraScienceNet, http://www.csm.ornl.gov/ultranet