Optimal pipeline decomposition and adaptive network mapping to support distributed remote visualization

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

[a]*Department of Computer Science, Southern Illinois University, Carbondale, IL 62901, USA*
[b]*Department of Computer Science, University of Memphis, Memphis, TN 38152, USA*
[c]*Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA*
[d]*Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803, USA*

Abstract

This paper discusses algorithmic and implementation aspects of a distributed remote visualization system that optimally decomposes and adaptively maps the visualization pipeline to a wide-area shared or dedicated network. The first node of the system typically generates or stores raw data sets, and a remote client resides on the last node equipped with a display device ranging from a personal desktop to a powerwall. Intermediate nodes include workstations, clusters, or rendering engines, which can be located anywhere on the network. We employ a regression method to estimate the effective bandwidth of a transport path. Based on link measurements, node characteristics, and module properties, we strategically organize visualization pipeline modules into groups and dynamically assign the groups to various network nodes to achieve minimal total delay or maximal frame rate. We propose polynomial-time algorithms using the dynamic programming method to compute optimal solutions for the problems of pipeline decomposition and network mapping under different constraint conditions. The proposed remote visualization system is implemented and deployed at several geographically distributed nodes for experimental testing. The proposed decomposition and mapping scheme is generic and can be applied to other distributed applications whose computing components form a linear arrangement.
© 2007 Elsevier Inc. All rights reserved.

Keywords: Remote visualization; Distributed computing; Visualization pipeline; Bandwidth measurement; Dynamic programming; 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 residing or generated at remote sites. A typical remote visualization system consists of a remote data source (DS) 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 performance of such systems critically relies on how efficiently the visualization pipeline is *decomposed* or *partitioned* and mapped to the network nodes. In this paper, we address both analytical and implementation aspects of realizing an optimal visualization pipeline decomposition and adaptive network mapping under various circumstances.

Many existing remote visualization systems [1,4,11] employ a predetermined partition of the visualization pipeline and typically 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 that typically deal with large data sets with complex structures. Particularly over wide-area connections, this problem is further compounded by limited network bandwidths and time-varying dynamics of competing network traffic. A considerable amount of research efforts have been made to implement flexible remote visualization systems that distribute visualization modules across network nodes. Bowman et al. [2] proposed a framework to predict the processing times of visualization modules using linear models and the network bandwidth using Network Weather Service

[10]; these predictions are used to obtain a suitable mapping of the visualization pipeline. Luke et al. [8] constructed Semotus Visum, a flexible remote visualization framework capable of multiple partition scenarios, which is tested and evaluated on a local network. In these works, the mapping of visualization modules to network nodes is accomplished by empirical testing and manual configuration. Furthermore, most of these visualization systems are limited to a traditional client and server mode with no intermediate nodes involved.

In this paper, we formulate a problem of optimizing the total delay or frame rate of a visualization pipeline by considering the times for data transmission, computation, and rendering. The resultant computational model enables us to analyze the algorithmic aspects such as complexity and optimality of mapping the visualization pipeline onto a given network. The first node serves as a DS with the capability of performing other tasks such as filtering, geometry generation, or rendering. The last node displays final images on the screen but also contains other visualization components; it could be equipped with a simple terminal or a more sophisticated device such as a powerwall or a tiled display. The intermediate nodes could be workstations, computational clusters, or custom rendering engines that are specially utilized to perform computation-intensive visualization subtasks. We present several polynomial-time solutions with increasingly stringent conditions based on dynamic programming to compute optimal mappings with minimum total delay or maximum frame rate. The time complexities of our methods range from $O(n \times k)$ to $O(n \times |E|)$ under different conditions, where $n + 1$ is the number of visualization modules in a pipeline, $k + 1$ is the number of network nodes arranged linearly in a predetermined order, and $|E|$ is the number of edges in an arbitrary wide-area shared or dedicated network.

The rest of the paper is organized as follows. In Section 2, we describe a generic visualization pipeline and a framework for the design of the distributed remote visualization system, which together form a basis for our analytical model. In Section 3, we describe a bandwidth measurement method based on [15], and then present the optimal decomposition and mapping schemes using dynamic programming-based algorithms for total delay minimization or frame rate maximization under various constraint conditions. The implementation details and test results are provided in Section 4. We finally conclude our work in Section 5.

2. Visualization pipeline and analytical model

2.1. Visualization pipeline

Large volumes of data generated in scientific or medical 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* [12].

Fig. 1 shows a high-level abstraction of a visualization pipeline along with the flow of data produced at each module in the pipeline. In many scientific applications, the raw data usually takes a multivariate format and is organized in structures such as CDF, NetCDF, and HDF [3,7,9]. The *filtering* module extracts the information of interest from the raw data and performs necessary preprocessing to improve the 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), or uses transfer functions to perform color and opacity classifications for each voxel based on its attribute. Rendering module converts the transformed geometric or composite 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 and opacity are stored and carried in a FB for final display on an end device.

It is worth pointing out that the visualization pipelines in real applications may significantly vary due to the disparate implementation procedures and the use of different visualization techniques; for instance, the rendering module itself might involve several stages of vertex transformations in the OpenGL environment.

2.2. Framework for design of distributed remote visualization system

The block diagram in Fig. 2 illustrates a baseline framework for the design of a distributed remote visualization system that employs pipeline *partitioning* or *decomposition* and automatic *mapping* to network nodes. Since many scientific applications generate terabyte or even petabyte data sets, it is currently very difficult to execute all visualization modules on a single desktop computer. Hence we first exploit the parallelism within 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 computing node capabilities, bandwidth estimates, sizes of data sets to be processed, and computational complexities of visualization modules, the visualization pipeline is decomposed into *groups*. These groups are then mapped one-to-one to the computing nodes distributed over a transport network.

In a practical implementation, we generally start with the computing node deployment and network topology construction. For example, the site of DS is *a priori* known, and the location of a remote client is determined whenever a "visualization" connection to the server is initiated. The information collected on networking and computing resources can be used to select optional intermediate nodes with specific visualization and/or computing capabilities. In this paper, we consider several problems of pipeline decomposition and network mapping with different optimization goals under various constraint conditions, as tabulated in Fig. 3.

2.3. Analytical model

We now describe an analytical model for the general problem of visualization pipeline decomposition and
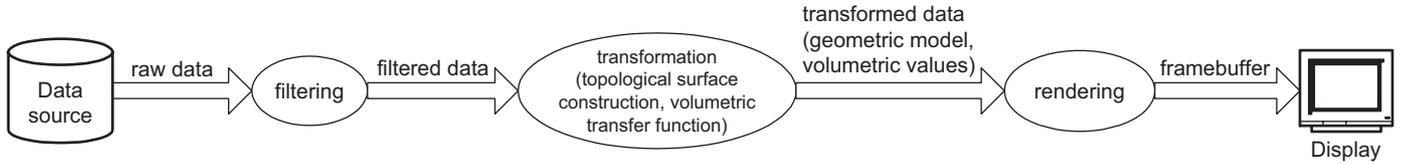
Fig. 1. A general visualization pipeline.
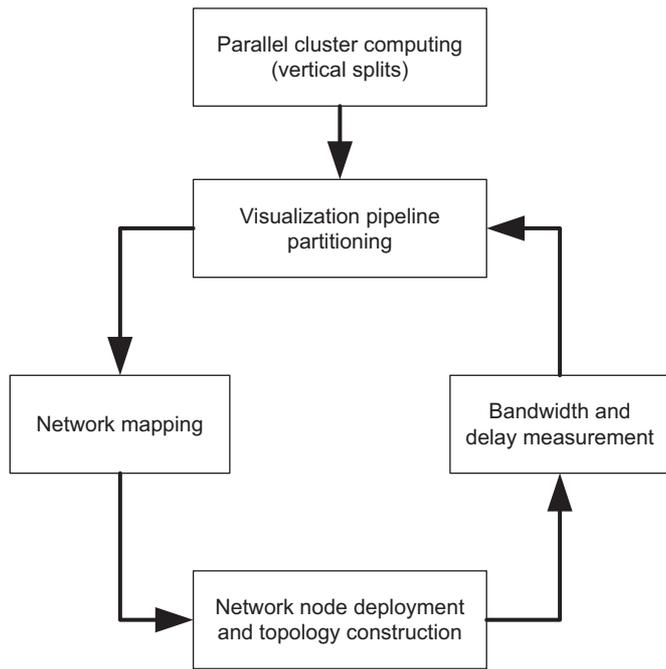
Fig. 2. Framework for design of remote visualization system.

its network mapping. As shown in Fig. 4, the visualization pipeline consists of $n + 1$ sequential modules, $M_1, M_2, \ldots, M_{u-1}, M_u, \ldots, M_{v-1}, \ldots, M_w, \ldots, M_{x-1}, M_x, \ldots, M_{n+1}$, where M_1 is a DS. 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 $v_1, v_2, \ldots, v_k, v_{k+1}$. Node v_i, $i = 1, 2, \ldots, k, k+1$ has a normalized computing power p_i [1] and is connected to its neighbor node v_j, $j = 1, 2, \ldots, k, k+1, j \neq i$ 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 $g_1, g_2, \ldots, g_{q-1}, g_q$, which are mapped one-to-one onto the q nodes on transport path P. The data exchanged between two adjacent groups is 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 simulation 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 in the order of bytes or kilobytes, which is often much smaller than the size of visualization data, its transport time is assumed to be negligible.

Minimal total delay: An important requirement in many collaborative visualization applications is the interactivity of the system. For remote visualizations, we account for this quantity by minimizing the total delay:

$$T_{\text{total}}(\textit{Path } P \textit{ of } q \textit{ nodes}) = T_{\text{computing}} + T_{\text{transport}}$$

$$= \sum_{i=1}^{q} T_{g_i} + \sum_{i=1}^{q-1} T_{L_{P[i], P[i+1]}}$$

$$= \sum_{i=1}^{q} \left(\frac{1}{p_{P[i]}} \sum_{j \in g_i, j \geqslant 2} \left(c_j m_{j-1} \right) \right)$$

$$+ \sum_{i=1}^{q-1} \left(\frac{m(g_i)}{b_{P[i], P[i+1]}} \right). \quad (1)$$

Here, 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. Note that in Eq. (1), we assume that the transport time between modules within a single group on the same computing node is negligible.

Maximal frame rate: We also consider another goal of maximizing the frame rate by minimizing the time incurred on a bottleneck link/node for applications with streaming data,

---

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

Problem No.	Mapping scheme	Module grouping in visualization pipeline	Node deployment in computer network	Optimization goals	Solutions
1	Bijective mapping (one-to-one onto)	No	Linear arrangement of a predetermined number of nodes	N/A	Trivial
2	Surjective mapping (onto)	Yes		Minimal total delay	Dynamic programming
3	Injective mapping (one-to-one)	No	Arbitrary network with undetermined number and order of nodes	Minimal total delay	Dynamic programming
4	General mapping for delay minimization	Yes			
5	General mapping for rate maximization	Yes		Maximal frame rate	

Fig. 3. Problem category of pipeline decomposition and network mapping.
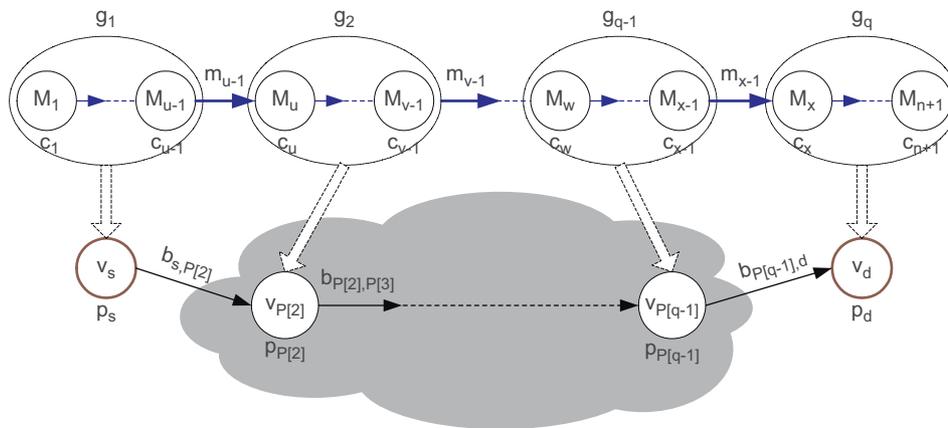
Fig. 4. Mathematical model for pipeline partitioning and network mapping.

given as in Eq. (2):

$$
\begin{aligned}
&T_{\text{bottleneck}}(Path\ P\ of\ q\ nodes) \\
&= \max_{\substack{Path\ P\ of\ q\ nodes \\ i=1,2,\ldots,q-1}} \left\{ \begin{array}{l} T_{\text{computing}}(g_i), \\ T_{\text{transport}}(L_{P[i],P[i+1]}), \\ T_{\text{computing}}(g_q) \end{array} \right\} \\
&= \max_{\substack{Path\ P\ of\ q\ nodes \\ i=1,2,\ldots,q-1}} \left\{ \begin{array}{l} \dfrac{1}{p_{P[i]}} \sum\limits_{j \in g_i \text{ and } j \geqslant 2} (c_j m_{j-1}), \\ \dfrac{m(g_i)}{b_{P[i],P[i+1]}}, \\ \dfrac{1}{p_{P[q]}} \sum\limits_{j \in g_q \text{ and } j \geqslant 2} (c_j m_{j-1}) \end{array} \right\}.
\end{aligned}
$$

(2)

3. Design of distributed remote visualization system

In this section, we first present a linear regression model to estimate the connection bandwidth and minimum delay using active traffic measurement, and propose solutions based on dynamic programming for the visualization pipeline partitioning and mapping problems under various conditions.

3.1. Bandwidth measurement

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 often 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 given cross traffic conditions. The notion of EPB is specific to the transport protocol employed by the transport daemon. The active measurement technique we apply here estimates the EPB and minimum delay for connection. Note that a connection may consist of multiple underlying physical links from possibly 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 exhibit randomness particularly over congested wide-area connections. We use Eq. (3) to estimate the end-to-end delay in transmitting a
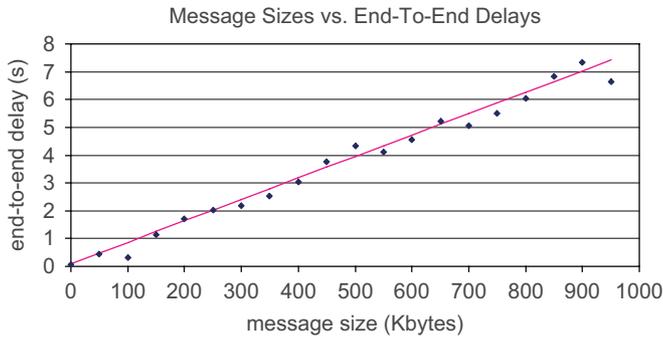
Fig. 5. End-to-end message transmission delay measurements between LSU and ORNL.

message of size r on a path P with l links:

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

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 the connection (for example, using TCP), and measures the end-to-end delays, on which we apply a linear regression to estimate the EPB. The delay measurements between Louisiana State University (LSU) and Oak Ridge National Laboratory (ORNL) as well as the corresponding linear regression are illustrated in Fig. 5, from which we estimate the EPB on this path to be about 1.0 Mbps. We emphasize that the same transport method used for collecting measurements will be used by the visualization modules. If measurements are collected by tools such as Iperf, or NWS [10], they must be appropriately translated into the effective bandwidth seen by the visualization pipeline. If the bandwidth measurements do not use the same transport module or the same transport parameters as the visualization modules, it is not sufficient to simply "plug-in" their estimates in Eqs. (1) and (2).

3.2. Pipeline partition and network mapping

We propose solutions based on dynamic programming to solve the visualization pipeline partition and mapping problems with the minimal total delay or maximal frame rate. Intuitively, a more general version of these problems is quite similar to the classical graph clustering problem, which is NP-complete and may be solved approximately, for example, using the branch-and-bound technique [6]. However, by exploiting the linear arrangement of visualization modules, we are able to achieve optimal solutions to these problems in polynomial time.

3.2.1. Minimal total delay

We consider Problems 2, 3, and 4 in Fig. 3 for total delay minimization under different stringent levels of constraint

conditions. Problem 2 decomposes the modules of the visualization pipeline into a fixed number of groups that are then mapped *onto* a pre-selected linear arrangement of network nodes. Problem 3 maps all the modules in the visualization pipeline *one-to-one* to different nodes in an arbitrary network, i.e. each node executes a single module. Problem 4 addresses a general case where the pipeline can be decomposed into any number of groups and the available computing nodes form an arbitrary network.

Surjective mapping with linearly arranged network nodes: In this scenario, all $k + 1$ nodes in the computer network are pre-selected and linearly arranged for mapping. We aim to decompose $n + 1$ modules into $k + 1$ groups and map them onto $k + 1$ nodes to achieve the minimal total delay of the visualization pipeline. There are n messages flowing between $n + 1$ visualization modules with sizes $m_j, j = 1, 2, \ldots, n$, and k network links with bandwidths $b_i, i = 1, 2, \ldots, k$ connecting $k + 1$ computing nodes, each of which has processing speed $p_s, s = 1, 2, \ldots, k + 1$.

Let $T(i, j)$ denote the minimal total delay with the first j messages (namely the first $j + 1$ visualization modules) mapped onto the first i network links. The dynamic programming method uses the following recursion to compute $T(i, j)$:

$$T(i, j)_{i=1 \text{ to } k, j=2 \text{ to } n, i \leqslant j}$$

$$= \min \begin{cases} T(i-1, j-1) + m_j/b_i + c_{j+1}m_j/p_{i+1}, \\ T(i, j-1) + c_{j+1}m_j/p_{i+1}, \end{cases}$$

$$(4)$$

where the base conditions are computed as $T(0, t) = \sum_{i=1}^{t} c_{i+1}m_i/p_1$, $t = 1, 2, \ldots, n$ in the first row and $T(t, t) = \sum_{i=1}^{t} m_i/b_i + \sum_{i=1}^{t} c_{i+1}m_i/p_{i+1}$, $t = 1, 2, \ldots, k$ on the diagonal line. Note that the first module is assumed to be a DS that does not introduce computing time. The complexity of this algorithm is in the order of $O(n \times k)$.

We now establish the correctness of Eq. (4). At each step of the recursion, $T(i, j)$ takes the minimal of two subcases as illustrated in Fig. 6. In the first subcase, the last message m_j is mapped to the last link b_i so that the transport time on this mapped link together with the computing time of the last module on the last node is added to $T(i-1, j-1)$, which is the subproblem of size $i - 1$ and $j - 1$. In the second subcase, 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 subproblem of size i and $j - 1$ with the addition of the computing time of the last module on the last node.

Fig. 7 shows the dynamics of 2D matrix construction for computing $T(i, j)$. The entries of the diagonal line and first row are entered beforehand as the base conditions. In order to fill up the whole upper triangle, the calculation sweeps across the matrix from left to right first and from top to bottom afterwards. For example, let us 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
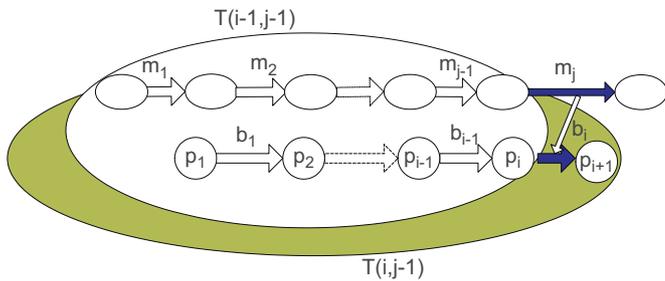
Fig. 6. Surjective mapping for minimal delay time.

Fig. 8. Illustration of n-hops shortest path algorithm.

Fig. 7. Construction of a 2D matrix of $T(i, j)$.

$T(2, 2)$ 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. 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 module M_{j+1} to the mapping scheme of $T(i - 1, j - 1)$. Note that an additional matrix is needed to record the mapping scheme for the winner of each comparison along with the computation.

Injective mapping with arbitrary network: In this scenario, all $n + 1$ visualization modules are individually mapped to a set of $n + 1$ nodes in an arbitrary network of $k + 1$ nodes with the first module mapped to v_s, and the last module mapped to v_d. That is, we consider a mapping path P of $q = n + 1$ nodes and n hops. The mapping of intermediate modules to network nodes needs to be decided such that the total delay time is minimized. Since no grouping of visualization modules is allowed, the pipeline partition is actually predetermined.

With appropriate adjustments made on the evaluation of the objective function, this problem turns into an n-hops shortest

path problem. Let $T^l(v_i)$ denote the minimal delay of a path with l hops from the source node v_s to the node v_i under consideration. We have the following recursion leading to $T^n(v_d)$:

$$\underset{l=1 \text{ to } n, v_i \in V}{T^l(v_i)} = \left\{ \min_{u \in adj(v_i)} \left(T^{l-1}(u) + c_{l+1} m_l / p_{v_i} + m_l / b_{u,v_i} \right) \right\}. \quad (5)$$

As Fig. 8 shows, this recursion follows from the observation that the minimal delay to v_i with l hops is the minimum of the delays to its neighbor with $l - 1$ hops plus the cost incurred by that link. The base conditions are computed as

$$\underset{v_i \in V \text{ and } v_i \neq v_s}{T^1(v_i)} = \begin{cases} c_2 m_1 / p_{v_i} + m_1 / b_{v_s, v_i} & \forall e_{v_s, v_i} \in E, \\ \infty & \text{otherwise.} \end{cases}$$

The complexity of this algorithm is $O(n \times |E|)$.

A general mapping scheme: Now we consider a general mapping scheme that removes the previous constraints imposed on the visualization pipeline and computer network. The objective of pipeline partition and network mapping is to minimize the total delay by decomposing the visualization pipeline into any number of groups and mapping them to an arbitrary network. This scenario imposes the least restrictions and is the most general case for our problem. 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 the computer network. Then, we have the following recursion leading to $T^n(v_d)$:

$$\underset{j=2 \text{ to } n, v_i \in V}{T^j(v_i)} = \min \begin{cases} T^{j-1}(v_i) + c_{j+1} m_j / p_{v_i}, \\ \min_{u \in adj(v_i)} \left(T^{j-1}(u) + c_{j+1} m_j / p_{v_i} + m_j / b_{u,v_i} \right), \end{cases}$$

$$(6)$$
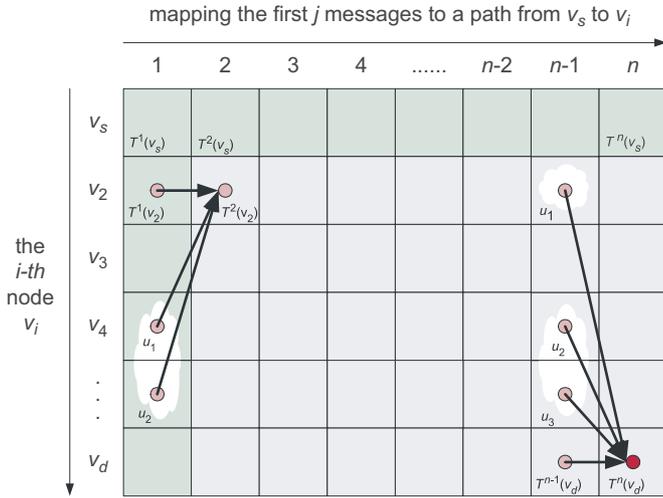
Fig. 9. Construction of 2D matrix in dynamic programming for the most general case.

with the base conditions computed as

$$T^1(v_i) \atop v_i \in V \text{ and } v_i \neq v_s} = \begin{cases} c_2 m_1/p_{v_i} + m_1/b_{v_s,v_i} & \forall e_{v_s,v_i} \in E, \\ \infty & \text{otherwise} \end{cases}$$

on the first column and $T^t(v_s) = \sum_{i=1}^{t} \left(c_{i+1}m_i/p_{v_s}\right)$, $t = 1, 2, \ldots, n$ on the first row in the 2D matrix of dynamic programming as shown in Fig. 9.

In Eq. (6), at each step of the recursion, $T^j(v_i)$ takes the minimal of two subcases. In the first subcase, 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 subproblem of node v_i of size $j-1$. This subcase is represented by the direct inheritance link from its left neighbor element in the 2D matrix. In the second subcase, 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 a cloudy area in Fig. 9. 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 further compared with the one calculated in the first subcase. 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 complexity of this algorithm is $O(n \times |E|)$.

3.2.2. Maximal frame rate

In visualization applications producing streaming data such as animations with a number of time steps, visualization data of the same modality is continuously generated, manipulated, and rendered in a pipelined manner. The maximal frame rate that a pipelining system can achieve is limited by the slowest (bottleneck) transport link or computing node along the pipeline. Due to the similarity of the recursive processes in achieving

optimization goals for surjective and injective mapping schemes, only the most general case is discussed here.

We adapt the above dynamic programming method to this problem by introducing necessary modifications. Let $1/T^j(v_i)$ denote the maximal frame rate with the first j messages (namely the first $j+1$ visualization modules) mapped to a path from source node v_s and node v_i in an arbitrary computer network. Let $S^j(v_i)$ represent the sum of the message sizes of all modules on node v_i with the first j messages mapped from node v_s to v_i. We have the following recursion leading to $T^n(v_d)$:

$$\begin{aligned} &T^j(v_i) \\ &{}_{j=2 \text{ to } n, v_i \in V} \\ &= \min \begin{cases} \max\left(T^{j-1}(v_i), \left(S^{j-1}(v_i) + c_{j+1}m_j\right)/p_{v_i}\right), \\ \min_{u \in adj(v_i)} \left(\max\left(T^{j-1}(u), c_{j+1}m_j/p_{v_i}, m_j/b_{u,v_i}\right)\right), \end{cases} \end{aligned} \tag{7}$$

with the base conditions computed as

$$T^1(v_i) \atop v_i \in V \text{ and } v_i \neq v_s} = \begin{cases} \max\left(c_2 m_1/p_{v_i}, m_1/b_{v_s,v_i}\right) & \forall e_{v_s,v_i} \in E \\ \infty & \text{otherwise} \end{cases}$$

and $T^t(v_s) = \sum_{i=1}^{t} \left(c_{i+1}m_i/p_{v_s}\right)$, $t = 1, 2, \ldots, n$.

Every transport link or computer node is a potential bottleneck and needs to be checked. At each step of the recursion, the bottleneck times for all possible schemes are computed and the scheme with the minimal bottleneck time is chosen as the optimal result for maximal frame rate. One inherent problem for the maximal frame rate algorithm is that since our algorithm does not exclude loops in network path selection, loops may compromise the optimality of frame rate for streaming data due to the non-exclusive computing node possession by different modules at the same time. If loops appear, optimality cannot be guaranteed for the maximal frame rate. However, such situations rarely occur.

It is worth pointing out that some additional constraints may arise in practical applications. For example, the maximal number of modules on a specific node may be limited, or 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 described in Eqs. (4)–(7). The scenario with failed feasibility check is simply discarded. Note that a solution in this situation is not guaranteed, especially when all scenarios fail in the feasibility checks. This feasibility checking feature makes our system versatile in meeting diverse practical requirements.

4. Implementation and case study

Our distributed remote visualization system is implemented in C++ on the Linux platform using GTK+ for client GUI design. We selected three Internet hosts, boba.sinrg.cs.utk.edu at University of Tennessee at Knoxville (UT), ozy4.csm.ornl.gov

at ORNL, and robot.rrl.lsu.edu at LSU for system deployment, as shown in Fig. 10. The hosts ozy4 at ORNL and robot at LSU are Linux workstations with common system configurations including a 3 GHz CPU. The host boba deployed at UT is a cluster, on which we use four nodes for parallel visualization computations in our experiment. Our system provides various visualization functionalities such as isosurface extraction, volume rendering, linear integral convolution (LIC), and streamline. Data transmission is currently carried out via TCP/IP sockets.

4.1. Experimental tests

Our system consists of four virtual component nodes, namely the client, central management (CM), DS, and computing service (CS), which are connected together by network links to form a closed loop. A visualization loop always starts at a client that initiates a particular visualization task by sending a request (containing data file, variable name, visualization method, view parameters, etc.) to a designated CM node. CM determines the best system configuration to accomplish the visualization task.

Based on the global knowledge of data source and system resource distributions within its service area, CM strategically partitions the pipeline of the selected visualization method 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, which is delivered sequentially over the rest of the loop
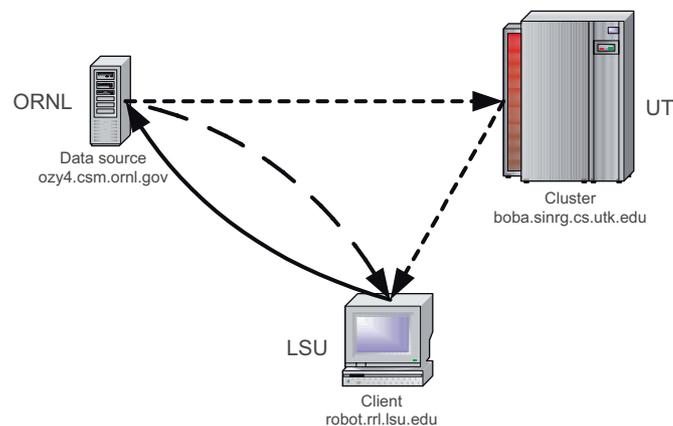
to establish the communication channel for the requested visualization task. Each CS node along the path receives intermediate results from upstream node, and executes the visualization modules designated by the routing table. Computation results are forwarded to the downstream node for further processing. This loop ends at the client node for final image display.

Several experiments are conducted with the ozy4 host at ORNL as the DS node, the boba cluster at UT as the CS node, and the robot host at LSU as both the client and the CM node. We established three different routing tables for isosurface visualization of two raw data sets as shown in Fig. 11. Each column indicates corresponding computing node for that computing module.

In the first case, the raw data set was sent out by the ozy4 host directly to the client end. The robot host at LSU received the raw data and performed local isosurface extraction and rendering. Significant end-to-end delays were observed for this case due to large data set transmission on a low-bandwidth link. In the second case, the ozy4 host performed isosurface extraction and off-screen rendering, and final image was shipped to the robot host for final display. End-to-end delays were reduced approximately by half of the first case. In the third case, the ozy4 host pushed raw data to the boba cluster for parallel isosurface extraction and off-screen rendering using four computing nodes. The resulted image was transmitted from the boba cluster to the robot host. Due to the speedup gained by the parallel processing, end-to-end delays were only about one-third of the second case.

In dealing with large data sets, the final image is preferably sent to the client especially when the available bandwidth is limited. The size of image generated by the rendering process is mainly decided by the width and height of the display screen regardless of the data set volume. In case of very low bandwidth links, the display window may need to be suitably scaled down to reduce the image size, and in addition, a lower level of image resolution may be chosen.

As the third case shows, adding intermediate parallel processing nodes can potentially accelerate the visualization pipeline. However, this may not be true if the benefit of time savings on parallel processing is offset by the overhead incurred for extra communications. Our system is self-adaptive so that an optimal visualization network configuration is always automatically selected by the dynamic programming-based core algorithm in the beginning of the visualization task.

Even during the same visualization session, any drastic condition changes might trigger re-computing the routing table

Fig. 10. Remote visualization system deployment.

Raw data	Send data	Isosurface extraction	Rendering	Display	Interactive response (sec)
Hand (5.24MB)	ozy4 @ ORNL	robot @ LSU			27.07
Engine (7.21MB)					41.53
Hand (5.24MB)	ozy4 @ ORNL			robot @ LSU	15.21
Engine (7.21MB)					27.37
Hand (5.24MB)	ozy4 @ ORNL	boba @ UT		robot @ LSU	5.84
Engine (7.21MB)					8.15

Fig. 11. Experimental results.

and therefore adaptively reconstructing the routing path. Such condition changes take place in the following circumstances: (i) a selected network link is more or less congested due to the increase or decrease in cross traffic, i.e., the effective BW changes; (ii) a new CPU-intensive process is concurrently executed on one selected node, i.e., the computing time changes; (iii) the user modifies isovalue, or viewing angles, or source data sets, which could affect both the transport and computing times; (iv) new system resources are added to the network, e.g., a new cluster is installed, a dedicated channel is brought up, one of the hosts is upgraded, or duplicate data sets are found at a closer site. All these network and host conditions are estimated and sent to the CM node for system reconfiguration on a periodical basis or at the time of event occurrence.

5. Conclusion and future plan

In this paper, we proposed a framework and a mathematical model for automatically mapping a visualization pipeline to a computer network. Our objectives are to minimize the total delay or maximize the frame rate of the visualization system. Dynamic programming-based approaches are proposed to compute optimal schemes for grouping and mapping the visualization modules onto a computer network, which could either be the Internet or a dedicated network.

It would be of our future interest to study various formulations of this class of problems from the viewpoint of computational criteria and practical implementation. The routing table is manually generated in our current implementation. In the future, we plan to strengthen our implementation to support automatic routing table construction after network condition monitoring daemon is embedded, and also deploy our system over dedicated networks, such as DOE UltraScience Net [5], for experimental testing.

As the data set 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 [13,14] 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

References

[1] Aspect ⟨http://www.aspect-sdm.org/⟩.
[2] I. Bowman, J. Shalf, K. Ma, 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.
[3] Common data format ⟨http://nssdc.gsfc.nasa.gov/cdf⟩.
[4] Computational engineering international ⟨http://www.ceintl.com/products/ensight.html⟩.
[5] Doe ultrasciencenet ⟨http://www.csm.ornl.gov/ultranet⟩.
[6] P. Fränti, V.O., K.T., Branch and bound technique for solving optimal clustering, in: International Conference on Pattern Recognition, 2002, pp. 232–235.
[7] Hierarchical data format ⟨http://hdf.ncsa.uiuc.edu⟩.
[8] E. Luke, C. Hansen, Semotus visum: a flexible remote visualization framework, in: Proceedings of IEEE Visualization, 2002, pp. 61–68.
[9] Network common data form ⟨http://my.unidata.ucar.edu/content/software/netcdf⟩.
[10] Network weather service ⟨http://nws.cs.ucsb.edu⟩.
[11] Paraview ⟨http://www.paraview.org/HTML/Index.html⟩.
[12] W. Schroeder, K. Martin, B. Lorensen, The Visualization Toolkit, second ed., Prentice-Hall PTR, 1998.
[13] Q. Wu, Control of transport dynamics in overlay networks, Ph.D. Thesis, Department of Computer Science, Louisiana State University, 2003.
[14] Q. Wu, N. Rao, A class of reliable UDP-based transport protocols based on stochastic approximation, in: Proceedings of the 24th IEEE INFOCOM, Miami, FL, March 13–17, 2005.
[15] Q. Wu, N. Rao, S. Iyengar, On transport daemons for small collaborative applications over wide-area networks, in: Proceedings of the 24th IEEE International Performance Computing and Communications Conference, Phoenix, Arizona, April 7–9, 2005.

Mengxia Zhu received the Ph.D. degree in computer science from Louisiana State University in 2005. She worked as a Ph.D. student research associate in the Computer Science and Mathematics Division at Oak Ridge National Laboratory on her Ph.D. dissertation from 2003 to 2005. Dr. Zhu is currently an assistant professor in Computer Science Department at Southern Illinois University at Carbondale. Her research interests include remote visualization, bioinformatics, and distributed sensor networks.
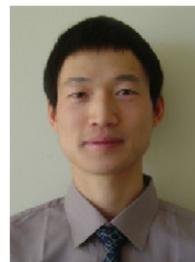
Qishi Wu received the B.S. degree in remote sensing and GIS from Zhejiang University, People's Republic of China in 1995, the M.S. degree in geomatics from Purdue University in 2000, and the Ph.D. degree in computer science from Louisiana State University in 2003. He was a research fellow in the Computer Science and Mathematics Division at Oak Ridge National Laboratory during 2003–2006. He is currently an Assistant Professor with the Department of Computer Science at University of Memphis. His research interests include computer networks, remote visualization, distributed sensor networks, high performance computing, algorithms, and artificial intelligence.

Nageswara S.V. Rao is currently UT-Battelle Corporate Fellow at Oak Ridge National Laboratory where he joined in 1993. He received M.E. from the School of Automation, Indian Institute of Science, Bangalore in 1984, and the Ph.D. in computer science from Louisiana State University in 1988. He was an assistant professor of computer science at Old Dominion University during 1988–1993. His research interests include network transport dynamics, statistical approaches to transport control,

information and sensor fusion, robot navigation, and fault diagnosis. He is an associate editor for Pattern Recognition Letters, International Journal of Distributed Sensor Networks, and Information Fusion.

S.S. Iyengar is the Chairman and Roy Paul Daniels Chaired Professor of Computer Science at Louisiana State University, Baton Rouge, and is also the Satish Dhawan Chaired Professor at the Indian Institute of Science, Bangalore. His publications include six textbooks, five edited books and over 380 research papers. His research interests include high-performance algorithms, data structures, sensor fusion, data mining, and intelligent systems. He is a world class expert in computational aspects of sensor networks.

He is a Fellow of IEEE, Fellow of ACM, Fellow of AAAS and Fellow of SDPS. He is a recipient of IEEE awards, best paper awards, the Distinguished Alumnus award of the Indian Institute of Science, Bangalore and other awards. He has served as the editor of several IEEE journals and is the founding editor-in-chief of the International Journal of Distributed Sensor Networks. His research has been funded by National Science Foundation (NSF), Defense Advanced Research Projects Agency (DARPA), Multi-University Research Initiative (MURI Program), Office of Naval Research (ONR), Department of Energy/Oak Ridge National Laboratory (DOE/ORNL), Naval Research Laboratory (NRL), National Aeronautics and Space Administration (NASA), US Army Research Office (URO) and various state agencies and companies. He has served on US National Science Foundation and National Institute of Health Panels to review proposals in various aspects of computational science and has been involved as and external evaluator (ABET-accreditation) for several computer science and engineering departments.

Dr. Iyengar had 39 doctoral students under his supervision and the legacy of these students can be seen in prestigious Laboratories (JPL, Oak Ridge National Lab, Los Alamos National Lab, Naval Research Lab) and universities round the world. He has been the program chairman of various international conferences.