4-2010

System Design and Algorithmic Development for Computational Steering in Distributed Environments

Qishi Wu
University of Memphis

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

Yi Gu
University of Memphis

Nageswara SV Rao
Oak Ridge National Laboratory

System Design and Algorithmic Development for Computational Steering in Distributed Environments

Qishi Wu, *Member*, *IEEE*, Mengxia Zhu, *Member*, *IEEE*, Yi Gu, *Student Member*, *IEEE*, and Nageswara S.V. Rao, *Fellow*, *IEEE*

Abstract—Supporting visualization pipelines over wide-area networks is critical to enabling large-scale scientific applications that require visual feedback to interactively steer online computations. We propose a remote computational steering system that employs analytical models to estimate the cost of computing and communication components and optimizes the overall system performance in distributed environments with heterogeneous resources. We formulate and categorize the visualization pipeline configuration problems for maximum frame rate into three classes according to the constraints on node reuse or resource sharing, namely no, contiguous, and arbitrary reuse. We prove all three problems to be NP-complete and present heuristic approaches based on a dynamic programming strategy. The superior performance of the proposed solution is demonstrated with extensive simulation results in comparison with existing algorithms and is further evidenced by experimental results collected on a prototype implementation deployed over the Internet.

Index Terms—Distributed computing, computational steering, remote visualization, performance modeling.

1 Introduction

Large-scale computations enabled by the advances in high-performance computing technologies have led to new insights and discoveries in several scientific disciplines. Many computations involve continuous simulations that span a number of time steps, each generating megabyte to gigabyte data sets, resulting in a total data size of terabytes to petabytes. These data sets must be stored, transferred, visualized, and analyzed by geographically distributed teams of scientists. Large-scale computations are typically scheduled in a "batch mode" on supercomputers and their outputs are examined at the end 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 movements in a meaningful direction. Such instances represent very ineffective utilization of valuable computing and human resources. They can be avoided if the progress of computations is monitored through visual feedback, even in a restricted sense, and the parameters are dynamically steered during the execution.

A visualization process generally consists of a number of steps or modules, which form a so-called visualization pipeline, and a remote visualization system needs to support the pipeline specific to the selected visualization technique over wide-area networks (WAN). Visualization modules, data objects, and network nodes and links have their own distinct characteristics: visualization modules are of different computing complexities; data objects transmitted between modules are of different sizes; network nodes have different processing power; and transport links have different bandwidths and end-to-end delays. The performance of such a system critically depends on how efficiently the visualization pipeline is realized, that is, partitioned and mapped onto network nodes. Most existing visualization systems do not support flexible and adaptive pipeline decomposition and network mapping. Oftentimes, they employ a fixed partition/mapping scheme: in a conventional client/server mode, the server typically performs filtering and geometry extraction, and the client performs rendering and display. Such fixed schemes do not always render optimal performance when running on WAN connections. For example, the geometric data size could be significantly larger than the original data volume if the data objects have very complex structures as in the case of the core collapse in the Terascale Supernova Initiative (TSI) simulation [1]; while the opposite is true if the data objects are of regular shapes with smooth surfaces at the initial stage of the TSI simulation. Therefore, an improper partition and mapping scheme may result in prohibitively expensive data transfer between the server and the client.

We propose *Remote Intelligent Computational Steering using Ajax* (RICSA) for online visualization and steering that optimizes the end-to-end performance of visualization pipelines over wide-area networks to ensure a smooth data flow. RICSA features a Web-based user interface via Asychronous JavaScript using XML (Ajax) technology to provide convenient and wide user access. This paper

- *Q. Wu and Y. Gu are with the Department of Computer Science, University of Memphis, Memphis, TN 38152-3240. E-mail: {qishiwu, yigu}@memphis.edu.*
- *M. Zhu is with the Department of Computer Science, Southern Illinois University, Faner 2142, Mail Code 4511, Carbondale, IL 62901. E-mail: mzhu@cs.siu.edu.*
- *N.S.V. Rao is with the Oak Ridge National Laboratory, Bldg 5600, Rm D305, Oak Ridge, TN 37831. E-mail: raons@ornl.gov.*

Manuscript received 12 Feb. 2009; accepted 29 Apr. 2009; published online 18 May 2009.
Recommended for acceptance by Y. Pan.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2009-02-0064.
Digital Object Identifier no. 10.1109/TPDS.2009.81.

focuses on both algorithmic development for performance modeling and optimization and system design for practical implementation and deployment. Based on the analytical cost models for visualization modules, computing nodes, and transport links, we formulate pipeline mapping for *Maximum Frame Rate* (MFR) in distributed environments as optimization problems and categorize them into three classes based on the constraints on node reuse or resource sharing: 1) MFR with No Node Reuse or Sharing (MFR-NNRS), 2) MFR with Contiguous Node Reuse and Sharing (MFR-CNRS), where multiple contiguous modules along the pipeline may be executed on one node, and 3) MFR with Arbitrary Node Reuse and Sharing (MFR-ANRS), which imposes no restriction on node reuse. We prove these three problems to be NP-complete and propose a set of dynamic-programming (DP)-based heuristic solutions, named Efficient Linear Pipeline Configuration (ELPC) algorithms. For comparison purposes, the *Streamline* [14] algorithm adapted to linearly pipelined workflows and a *Greedy* algorithm are also implemented and tested in the same simulated networks. The Streamline algorithm is mainly designed to allocate computation and communication resources to various stages of a computing workflow with different requirements in grid environments. The simulation results illustrate the efficacy of the ELPC algorithms in comparison with Streamline and Greedy algorithms. The performance superiority of the proposed system is further evidenced by the experimental results collected on a prototype implementation of the RICSA system deployed over the Internet.

The rest of the paper is organized as follows: In Section 2, we describe existing efforts related to our work. Section 3 presents the system architecture and Section 4 describes a transport method that achieves stable and smooth data flows for control channels. In Section 5, we construct cost models for pipeline and network components, define an objective function to maximize the frame rate, and discuss various performance measurement techniques. In Section 6, we categorize the pipeline mapping problems into three classes and prove their NP-completeness. We design heuristic mapping approaches in Section 7. Simulation-based performance evaluations are conducted in Section 8 and implementation details as well as experimental results are provided in Section 9. We conclude our work and discuss future research in Section 10.

2 Related Work

Many commercial or noncommercial software products for remote visualization [2], [3], [4] employ a predetermined partition of visualization pipelines and send fixed-type data streams such as raw data, geometric primitives, or frame buffer 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 aimed at designing architectures that assign visualization modules across network nodes more efficiently. Brodlie et al. [18] 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. [16] 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 and tested it on a local network. Bowman et al. [17] proposed a framework to predict the processing time of visualization modules using analytic models, which can be used to obtain a suitable mapping of visualization pipelines.

Closely related to our pipeline mapping problems is the work in [15], where Benoit and Robert presented the mapping of computing pipelines onto different types of fully connected networks with identical processors and links, with identical links but different processors, or with different processors and links. They discussed three versions of pipeline mapping problems, i.e., one-to-one, interval, and general mappings, all of which are NP-complete in fully heterogeneous platforms. The problem that maps each task onto one node in a serial manner is similar to MFR-NNRS in [29]. However, our work differs from theirs mainly in the fact that the network we consider features an arbitrary topology with an arbitrary number of heterogeneous computing nodes and transport links.

There have been several research efforts on the design and implementation of computational steering systems. Existing systems such as SCIRun [25], CUMULVS [5], VIPER [24], and RealityGrid [6] generally require a high learning curve for end users. Besides, various packages such as Globus, SOAP, PVM [7], and AVS [8] need to be installed at user sites to realize their full benefits. These factors often place undue burden on users, who are primarily scientists, to spend significant efforts 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 provides a user interface using Ajax Web technologies to offer a rich user environment and enables any user with an Internet connection to use a Web browser to monitor a remote ongoing computation process and also steer the computation dynamics on the fly.

3 System Framework

As shown in Fig. 1, our system consists of five virtual component nodes, Ajax client (Client), Ajax frontend (Web server), central management (CM), data source (DS), and computing service (CS), which are connected together over a network to form a visualization-steering loop. In general, a DS node either contains pregenerated data sets or a simulator that runs on a host, a cluster, or a supercomputer, where simulation data are continuously produced and periodically cached on a local storage device, which serves as a data source.

A computational steering is initiated at a Client node by sending a request specifying the simulation program, variable names, visualization method, and viewing parameters to the Ajax frontend, which forwards the request to a designated CM. The CM then creates a connection and forwards the request to a remote simulator to run the preloaded simulation. Based on the global knowledge of system resource distributions and simulation data set properties, the CM strategically partitions the visualization pipeline into groups and selects an appropriate set of CS

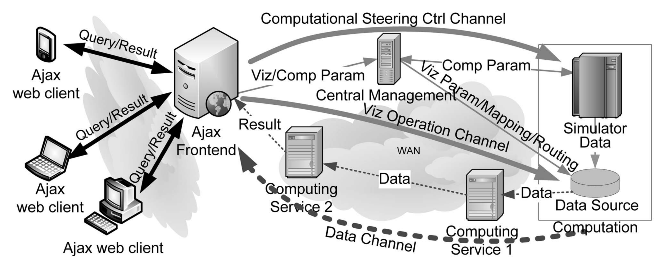


Fig. 1. System architecture and components.

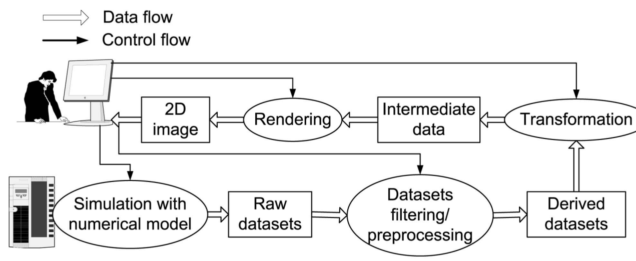


Fig. 2. A general visualization steering pipeline: visualization modules, data flow, and control flow.

nodes to execute the visualization modules. The results of pipeline partitioning and network mapping are stored in a visualization routing table and delivered sequentially over the loop to establish a network routing path. The visualization-steering loop comprises two channel segments: 1) control channel from the Ajax frontend to the simulator or data source for computational steering and visualization operations and 2) data channel from the computation back to the frontend, as represented by the solid and dotted lines in Fig. 1, respectively. The Ajax frontend receives and saves images as fixed-size files, which are transmitted to the user through the object exchange mechanism of XMLHttpRequest.

4 Transport Method for Control Channel

We integrate the transport stabilization method described in [26] into the proposed RICSA system to provide stable channels for smooth control message delivery. In this transport method, Rao et al. consider a general window-based transport structure that utilizes UDP for application-level transport. This model sends a congestion window $W_c(t)$ of 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 time is computed as:

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

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 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] and the real network implementation exhibits very robust stabilization performance over a variety of network connections.

5 Optimizing Visualization Pipeline

5.1 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]. For a small-scale standalone application where an end user accesses a local data source, the entire visualization pipeline may be executed on a single computer. However, in large-scale distributed applications with geographically distributed data sources and end users, it is a significant challenge to support complex visualization pipelines over wide-area networks with heterogeneous computing nodes and transport links. The remote visualization in next-generation scientific applications such as Terascale Supernova Initiative (TSI) [1] is a typical example. Note that a computing pipeline with only two modules reduces to the traditional client/server mode. Fig. 2 shows a high-level abstraction of a visualization and steering pipeline along with data and control flows.

The raw data in many scientific applications is stored in multivariate format such as CDF, HDF, and NetCDF [9], [10], [11]. The filtering module extracts the information of interest from the raw data and performs necessary preprocessing to improve processing efficiency and save communication cost. The transformation module typically uses a surface fitting technique such as isosurface extraction to derive 3D geometries and the rendering module converts the transformed geometric data to pixel-based images. During a simulation run, an end user may steer the computation and control the visualization that takes place in various computing modules along the pipeline. Such steering or control commands are delivered through the stable channels described in Section 4.

5.2 Cost Models of Pipeline and Network Components

We construct analytical cost models for pipeline and network components to facilitate the formulation of the objective function. The computational complexity of computing module[1] w_i is denoted as a function $f_{w_i}(\cdot)$ of the incoming data size $z_{i-1,i}$ sent from its preceding module w_{i-1}, which determines the number of instructions needed to complete the subtask defined in the module. The output

1. In some contexts, computing modules are also referred to as stages or subtasks.

data of size $z_{i,i+1}$ is, in turn, sent to its succeeding module w_{i+1} for further processing. The processing capability of a network node is a result of host factors such as processor frequency, bus speed, memory size, storage performance, and presence of coprocessors. For simplicity, we use a normalized quantity p_i to represent the overall computing power of network node v_i without specifying its details, i.e., the number instructions that can be executed in one unit of time. The transport link or edge between network nodes v_i and v_j is denoted by $e_{i,j}$, which is characterized by bandwidth (BW) $b_{i,j}$ and minimum link delay (MLD) $d_{i,j}$. We estimate the computing time of module w_i running on network node v_j to be $T_{computing}(w_i, v_j) = \frac{f_{w_i}(z_{i-1,i})}{p_j}$ and the transfer time of message size z over transport link $e_{i,j}$ to be $T_{transport}(z, e_{i,j}) = \frac{z}{b_{i,j}} + d_{i,j}$.

5.3 Objective Function for Maximum Frame Rate

Due to the disparate nature of data sources and intrinsic heterogeneity of network nodes, transport links, and application computing tasks, deploying component modules on different sets of computer nodes can result in substantial performance differences. The visualization pipeline mapping problem is to find an efficient mapping scheme that maps the computing modules onto a set of strategically selected nodes to maximize the frame rate for applications where time series data sets are generated and fed into a visualization pipeline to sustain a continuous and smooth data flow.

We represent the transport network as a graph $G = (V, E), |V| = n$, where V denotes the set of network or computer nodes and E the set of directed transport links or edges. Note that the transport network may or may not be a complete graph. A general computing pipeline consists of m sequential modules $w_0, w_1, \ldots, w_{m-1}$, where w_0 is a data source and w_{m-1} represents an end user. The objective of a mapping scheme is to divide the pipeline into q groups of modules denoted by $g_0, g_1, \ldots, g_{q-1}$, and map them onto a selected network path P of q nodes from source v_s to destination v_d in the computer network, where $v_s, v_d \in V, q \in [1, m]$, and path P consists of a sequence of not necessarily distinct nodes $v_{P[0]} = v_s, v_{P[1]}, \ldots, v_{P[q-1]} = v_d$.

We wish to maximize the frame rate to produce the smoothest data flow by identifying and minimizing the bottleneck time (BT), i.e., the time incurred on a bottleneck link or node, which is defined as:

$$\begin{aligned}
&T_{bottleneck}(\text{Path } P \text{ of } q \text{ nodes})\\
&= \max_{\substack{\text{Path } P \text{ of } q \text{ nodes}\\ i=0,1,\ldots,q-2}} \begin{pmatrix} T_{computing}(g_i), \\ T_{transport}(e_{P[i],P[i+1]}), \\ T_{computing}(g_{q-1}) \end{pmatrix}\\
&= \max_{\substack{\text{Path } P \text{ of } q \text{ nodes}\\ i=0,1,\ldots,q-2}} \begin{pmatrix} \frac{\alpha_{P[i]}}{p_{P[i]}} \max\limits_{j \in g_i, j \geq 1} (f_{w_j}(z_{j-1,j})), \\ \frac{\beta_{P[i],P[i+1]} \cdot z(g_i)}{b_{P[i],P[i+1]}} + d_{P[i],P[i+1]}, \\ \frac{\alpha_{P[q-1]}}{p_{P[q-1]}} \max\limits_{j \in g_{q-1}, j \geq 1} (f_{w_j}(z_{j-1,j})) \end{pmatrix},
\end{aligned} \tag{2}$$

where $\alpha_{P[i]}$ is the number of modules assigned to node $v_{P[i]}$, and $\beta_{P[i],P[i+1]}$ is the number of data sets transferred over link $e_{P[i],P[i+1]}$ between nodes $v_{P[i]}$ and $v_{P[i+1]}$. We assume equal share of node computing power and link bandwidth among concurrent module executions and data transfers, respectively. We use $z(g_i)$ to denote the output data size of group g_i, which is the same as that of the last module in the group. We also assume that the first module w_0 only transfers data from the source node and the last module w_{m-1} only performs certain computation without data transfer.

5.4 Transport Time Estimation

We present a linear regression model to estimate the bandwidth of a virtual link or transport path using active traffic measurement based on [30]. Due to complex traffic distribution over wide-area networks and the nonlinear nature of transport protocol dynamics (in particular, TCP), the throughput achieved in actual message transfers is typically different from the link or 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 and use the active measurement technique to estimate the EPB and MLD of a given network path.

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. The cost model that measures the end-to-end delay in transmitting a message of size z on a path P with l physical links is defined as:

$$\begin{aligned}
d(P, z) &= d_{BW}(P, z) + \sum_{i=1}^{l} \big(d_{p,i}(P) + d_{q,i}(P, z)\big)\\
&= \frac{z}{EPB(P)} + MLD(P).
\end{aligned} \tag{3}$$

Based on (3), we can apply a linear regression method to estimate EPB and MLD using a number of sample data sets with various sizes.

5.5 Computing Module Performance Modeling

The time for executing visualization tasks depends on various factors including available system resources, data sizes, visualization methods, 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 for extracting isosurfaces from a data set 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. We design performance estimation methods using analytical models and statistical measurements for commonly used visualization techniques: isosurface extraction, ray casting, and streamline.

5.5.1 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. In general, the time to extract isosurfaces from a data set is determined by the number of blocks containing isosurfaces n_{blks}, the number of cells in a block S_{blk}, and the average time of extracting isosurfaces from a block t_{blk}, which depends on S_{blk}. We define the performance model for isosurface extraction as:

$$t_{extraction}(n_{blks}, S_{blk}) = n_{blks} \times t_{blk}(S_{blk}). \qquad (4)$$

In this model, n_{blks} and S_{blk} depend directly on the data partitioning method, which is usually known beforehand. However, since t_{blk} is also controlled by the isovalue selected by the user at runtime, it is difficult to provide an exact expression relating t_{blk} to the other parameters. We employ a statistical method to predict the isosurface extraction time t_{blk} using a set of testing data sets sampled from various applications. We run the isosurface extraction algorithm on these data sets divided into different block sizes with a large number of possible isovalues. For each of 15 cases, including the one with no isosurface, we measure the frequency of the related cells found inside a block as well as the time spent on each case, $T_{Case}(i)$, where $i \in [0, 14]$. The case probability $P_{Case}(i)$ is defined as the averaged occurrence for each case. At runtime, we estimate the average time spent on a block as:

$$t_{blk}(S_{blk}) = S_{blk} \times \sum_{i=0}^{14} (T_{Case}(i) \times P_{Case}(i)), \qquad (5)$$

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

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

$$t_{render} = n_{blks} \times S_{blk} \times \sum_{i=0}^{14} (n_{\triangle}(i) \times P_{Case}(i)). \qquad (6)$$

We tested this cost model using 64 cubed and 256 cubed volume blocks of data sets from real applications in various medical, engineering, and science domains, and consistently achieved a high prediction accuracy with a relative prediction error of less than 5.0 percent in actual runs.

5.5.2 Ray Casting

Similar to isosurface extraction, without loss of generality, we assume ray casting is also performed at the block level. The performance estimation for ray casting is much harder than that 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:

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

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. Thus, n_{rays} and $n_{samples}$ only depend on the viewing vector and are constant for a given view if orthographic projection is used. t_{sample} can be considered as a constant and can be easily computed by running ray casting algorithm on a test data set for each machine. Such estimation would be more accurate if each nonempty block is semitransparent. We conducted ray casting experiments on a PC equipped with 2.4 GHz CPU and 2 GBytes memory using a data set with 512 nonempty blocks of 643 voxels, and observed that the time to render each block is 0.387 seconds on average with a relative standard deviation of 4.7 percent.

5.5.3 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 can be defined as:

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

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 runtime performance estimation.

6 Problem Classification and Complexity Analysis

For the mapping objective of MFR, we consider three different types of mapping constraints: 1) no node reuse or share (MFR-NNRS), i.e., a node on the selected path P executes exactly one module; 2) contiguous node reuse and share (MFR-CNRS), i.e., two or more contiguous modules in the pipeline are allowed to run on the same node (the selected path P contains self-loops); and 3) arbitrary node reuse and share (MFR-ANRS), i.e., two or more modules, either contiguous or noncontiguous in the pipeline, are allowed to run on the same node (the selected path P contains loops). In MFR-NNRS, any node in the network can be used at most once, which requires selecting the same number of nodes for one-to-one and onto module mapping. Node reuse is allowed in both MFR-CNRS and MFR-ANRS. However, in the former, the modules mapped onto the same node must be contiguous along the pipeline, which is not required in the latter. These two problems appear to be

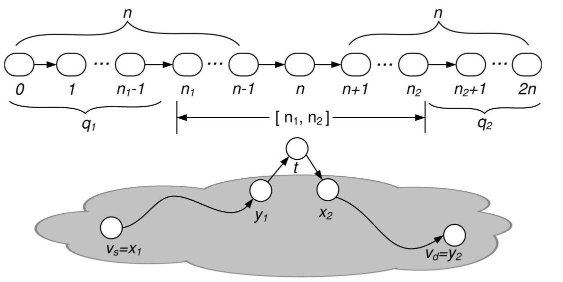


Fig. 3. Reduction from DISJOINT-CONNECTING-PATH problem.

similar to the Maximum n-hop Shortest Path problem, but are proved to be NP-complete when resources are shared.

The MFR of a computing pipeline is limited by the bottleneck unit, i.e., the slowest data transfer or module execution along the entire pipeline. In MFR-NNRS, we attempt to find the widest path with exact n nodes to map n modules in the pipeline on a one-to-one basis. Here, "widest" refers to the maximum bottleneck among all feasible paths. The MFR-NNRS problem can be simplified to the exact n-hop widest path problem by assuming identical modules, which is equivalent to the Exact n-hop Shortest Path problem (ENSP) in complexity. The reduction from the Hamiltonian Path problem, which is known to be NP-complete, to ENSP is trivial.

6.1 NP-Completeness Proof of MFR-CNRS and MFR-ANRS

We define MFR-CNRS as a decision problem as follows:

Definition 1. *Given a linear computing pipeline of m modules, a directed weighted transport network G and a bound T, does there exist a mapping of the pipeline to the network with contiguous node reuse such that the BT does not exceed T?*

Theorem 1. *MFR-CNRS is NP-complete.*

Proof. We use a reduction from DISJOINT-CONNECTING-PATH (DCP) [20], which has been shown to be NP-complete even when restricting to two paths in the case of directed graphs (2DCP) [19]. The problem clearly belongs to NP: given a division of the pipeline into q groups and a path P of q nodes, we can compute the BT in polynomial time and check if the bound T is satisfied. We prove its NP-hardness by showing that 2DCP $\leq_p$ MFR-CNRS.

Consider an arbitrary instance $\mathcal{I}_1$ of 2DCP, i.e., a graph $G = (V, E)$ and two disjoint vertex pairs (x_1, y_1), $(x_2, y_2) \in V^2$. We ask whether G contains two mutually vertex-disjoint paths, one going from x_1 to y_1, and the other going from x_2 to y_2. The number of nodes in the graph is $n = |V| \geq 4$. We can construct the following instance $\mathcal{I}_2$ of MFR-CNRS as shown in Fig. 3. The pipeline consists of $m = 2n + 1$ modules, and the computational complexity of each module w_i is $f_{w_i}(z) = 1$ for $1 \leq i \leq 2n, i \neq n$ and $f_{w_n}(z) = n^2$. In other words, only module w_n has a much higher complexity than the other modules. The network consists of $|V| + 1$ nodes and $|E| + 2$ links: starting from graph G, we add a new node t which is connected to G with an incoming link from y_1 to t and an outgoing link from t to x_2. For each node $v_j \in V$, the processing capability is set to $p_j = 1$, except for node t, whose processing power is n^2 much higher than others. Link bandwidths are set to be very high so that transport time between nodes is ignored compared to computing time. The source and destination nodes v_s and v_d are set to be x_1 and y_2, respectively. We ask whether we can achieve a BT of $T_{MFR} = 1$. Obviously, this instance transformation can be done in polynomial time.

We show that given a solution to $\mathcal{I}_1$, we can find a solution to $\mathcal{I}_2$ of MFR-CNRS. Let q_i be the length of the path from x_i to y_i, for $i = 1, 2$. We have $q_i \leq n$ for both paths and paths are disjoint. We map the first q_1 modules with no node reuse on q_1 nodes of the path from x_1 to y_1, and thus, w_0 is mapped on the source node $x_1 = v_s$. The rest $n - q_1$ modules are also mapped on y_1 and only contiguous modules are mapped on this node. Each of these n modules incurs a delay of 1. We map module w_n with a complexity of n^2 to the fast node t, which incurs a delay of $n^2/n^2 = 1$. Similarly, the last n modules are mapped on the path from x_2 to y_2: the next $n - q_2$ contiguous modules are mapped on x_2, and the remaining q_2 modules are mapped on the path with no node reuse. Each of these n modules also incurs a delay of 1. This mapping scheme only involves contiguous node reuse according to the solution to $\mathcal{I}_1$: the two paths are vertex-disjoint, hence no node is reused for noncontiguous modules. Moreover, since the delays on all modules are identical, the BT of the entire pipeline is $1 \leq T_{MFR}$, thus producing a valid solution to $\mathcal{I}_2$.

Reciprocally, if $\mathcal{I}_2$ has a solution, we show that $\mathcal{I}_1$ also has a solution. We prove that the mapping scheme of $\mathcal{I}_2$ has to be of a similar form as the one described above, and thus, there exist disjoint paths $x_1 \rightarrow y_1$ and $x_2 \rightarrow y_2$. This property is due to the fact that node t must be used in the mapping. Indeed, if node t is not used to process module w_n, this module will incur a delay of n^2 much larger than the bottleneck cost $T_{MFR} = 1$. Since the BT of $\mathcal{I}_2$ is no larger than $T_{MFR} = 1$, module w_n must be mapped on node t in the solution to $\mathcal{I}_2$. Let $[n_1, n_2]$ denote the range of modules that are assigned to node t in the mapping, where $n_1 \leq n \leq n_2$. Since the only links connecting w_n to the network are $y_1 \rightarrow t$ and $t \rightarrow x_2$, the mapping scheme of $\mathcal{I}_2$ must also involve nodes x_2 and y_1. Moreover, since w_0 is mapped on x_1 and w_{2n} is mapped on y_2, the source and destination nodes x_1 and y_2 are also used. Therefore, the mapping scheme is as follows: modules w_0 to w_{n_1-1} are mapped to a path of nodes between x_1 and y_1, modules w_{n_1} to w_{n_2} are mapped on node t, and modules w_{n_2+1} to w_{2n} are mapped to a path of nodes between x_2 and y_2. Since only contiguous modules along the pipeline can be deployed on the same node in this mapping, nodes in both paths $x_1 \rightarrow y_1$ and $x_2 \rightarrow y_2$ are distinct, and they are connected by edges in G by construction of $\mathcal{I}_2$. Thus, we found disjoint paths and a solution to $\mathcal{I}_1$. Proof ends. □

The NP-completeness proof for MFR-ANRS is based on the Widest path with Linear Capacity Constraints (WLCC) [31] problem. An LCC-graph is a triplet $(G = (V, E), C, b)$, where the capacity of each link $e \in E$ is a variable x_e, and (C, b) represents a set of m linear capacity constraints $Cx \leq b$, C is a 0-1 coefficient matrix of size $m \times |E|$, x is an $|E| \times 1$

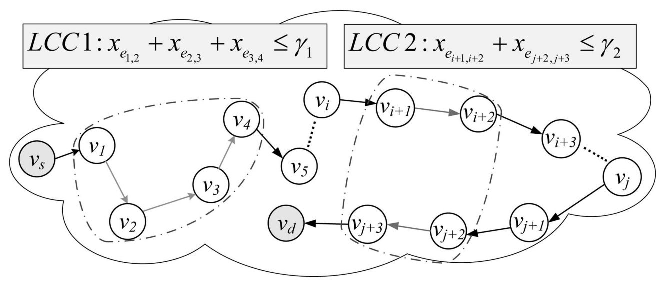

$LCC1: x_{e_{1,2}} + x_{e_{2,3}} + x_{e_{3,4}} \le \gamma_1$
$LCC2: x_{e_{i+1,i+2}} + x_{e_{j+2,j+3}} \le \gamma_2$
v_s
v_1
v_2
v_3
v_4
v_5
v_i
v_{i+1}
v_{i+2}
v_{i+3}
v_j
v_{j+1}
v_{j+2}
v_{j+3}
v_d

Fig. 4. An example of the LCC-graph.

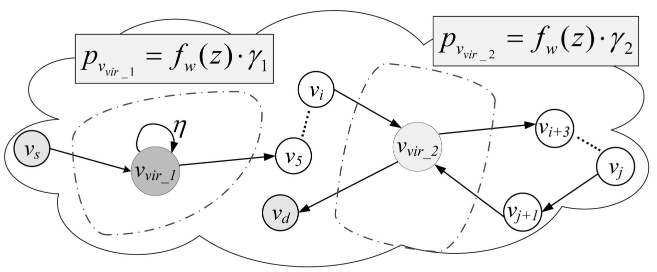

$p_{v_{vir\_1}} = f_w(z) \cdot \gamma_1$
$p_{v_{vir\_2}} = f_w(z) \cdot \gamma_2$
η
v_s
v_{vir_1}
v_5
v_i
v_{vir_2}
v_{i+3}
v_j
v_{j+1}
v_d

Fig. 5. The LCC-graph replaced with virtual nodes.

vector of link capacity variables, and $b \in R^m$ is a capacity vector. Each link $e \in E$ has a capacity $c(e) \ge 0$. Given an LCC-graph (G, C, b), the width $\omega(P)$ of a path $P = (e_1, e_2, \cdots, e_k) \subset G$ is defined as the bottleneck capacity x_{BN} subject to $x_{e_j} = 0, \forall e_j \notin P, Cx \le b$, and $x_{e_1} = x_{e_2} = \cdots = x_{e_k} = x_{BN}$. We define WLCC as a decision problem: Given an arbitrary instance $(G = (V, E), C, b)$ of WLCC, two nodes $v_s, v_d \in V$, and a positive integer $K \le Max\{b_i\}$, does there exist a path P from v_s to v_d whose width, i.e., bottleneck capacity x_{BN}, is no less than K? This problem was shown to be NP-complete in [31]. Similarly, we define MFR-ANRS as a decision problem:

Definition 2. *Given a linear computing pipeline of m modules, a transport network G, and a bound B, does there exist a mapping of the pipeline to the network with arbitrary node reuse such that the frame rate, i.e., the inverse of the BT, is no less than B?*

Theorem 2. *MFR-ANRS is NP-complete.*

Proof. For a given solution to an instance of MFR-ANRS, we can go through the entire path to calculate the frame rate for each link and node and check if it satisfies the bound. This evaluation process can be done in polynomial time, which means $\mathrm{MFR-ANRS} \in \mathrm{NP}$. We prove its NP-hardness by reducing WLCC problem to it.

Given an arbitrary instance $\mathcal{I}_1$ in WLCC problem, in which there are several LCCs among the links, as shown in Fig. 4, we can transform it into an instance $\mathcal{I}_2$ of the MFR-ANRS problem, i.e., $\mathcal{I}_1 \in WLCC \Rightarrow \mathcal{I}_2 = F(\mathcal{I}_1) \in$ MFR-ANRS, where $F(\cdot)$ is a polynomial-time transformation function. We first construct a pipeline that consists of $|V|$ identical modules w, whose computational complexity is denoted as a function $f_w(\cdot)$ of the same input data size z. We then make a copy of the entire topology of G and denote it as graph $G' = (V', E')$, where $V' = V$ and $E' = E$. Any vertex $v \in V$ in G not on a constrained link and any link $e \in E$ in G not in constraints (C, b) remain unchanged in V' and E', respectively, including the capacity of each link. Here, we consider three types of LCCs:

- Case 1 of adjacent LCC: If there are η constrained links within one LCC that are all adjacent, their corresponding vertices in V' are bundled together and replaced with a virtual node v_{vir}, which contains the same number of self-loops as that of contiguously adjacent constrained links defined in the LCC. The processing power p of the virtual node is set to $f_w(z) \cdot \gamma$, where γ is the LCC capacity. For example, in Fig. 4, the links $e_{1,2}, e_{2,3}$, and $e_{3,4}$ have an LCC $x_{e_{1,2}} + x_{e_{2,3}} + x_{e_{3,4}} \le \gamma_1$ among them, so we have $\eta = 3$ and $\gamma = \gamma_1$;
- Case 2 of nonadjacent LCC: If there are ζ constrained links within one LCC that are all nonadjacent, their corresponding vertices in V' are bundled together and replaced with a virtual node v_{vir}. The processing power p of the virtual node is again set to $f_w(z) \cdot \gamma$, where γ is the LCC capacity. For example, in Fig. 4, the links $e_{i+1,i+2}$ and $e_{j+2,j+3}$ have an LCC $x_{e_{i+1,i+2}} + x_{e_{j+2,j+3}} \le \gamma_2$ between them, so we have $\zeta = 2$ and $\gamma = \gamma_2$.
- Case 3 of mixed LCC: This is a combination of the first two cases. The corresponding vertices in V' of all adjacent and nonadjacent constrained links within one LCC are bundled together and replaced with a virtual node v_{vir}, whose properties (self-loops and processing power) are treated the same way as in the first two cases.

Furthermore, the processing power of all the other nodes is set to infinity. The newly constructed graph G' with one adjacent LCC and one nonadjacent LCC is shown in Fig. 5. Finally, we select a bound $B = K$. Obviously, this transformation can be done in polynomial time. The question for MFR-ANRS is: does there exist a mapping path P' from v'_s to v'_d in G' that provides frame rate no less than B?

We show that given a solution to WLCC problem, we can find a solution to the MFR-ANRS problem. Suppose that there exists a path P from v_s to v_d in G of width no less than K. We first identify a path P' in G' corresponding to path P in G, and then sequentially map the modules in the pipeline onto the nodes along the path P' from v'_s to v'_d, with the first module mapped onto source node v'_s and the last module mapped onto destination node v'_d. Between v'_s and v'_d, we sequentially map each module onto a regular node along the path, and when a virtual node is encountered, we have three different mapping schemes:

- Case 1 of adjacent LCC: If this virtual node is converted from η adjacent constrained links, among which η' ($\eta' \le \eta$) links are on path P, we map η' contiguous modules to it;
- Case 2 of nonadjacent LCC: If this virtual node is converted from ζ nonadjacent constrained links, among which ζ' ($\zeta' \le \zeta$) links are on path P, we map one module to it every time when the path passes through it for total ζ' times;
- Case 3 of mixed LCC: If this virtual node is converted from both adjacent and nonadjacent constrained links, we map modules to it according to a combination of the mapping strategies for

virtual nodes converted from either a single adjacent LCC or a single nonadjacent LCC as specified in the first two cases.

Any remaining contiguous modules are mapped onto the nonvirtual destination node v'_d. If the destination node v'_d itself is a virtual node in G', we can create a special node with infinite processing power and connect the destination node v'_d to it with infinite bandwidth on the link. This special node is then considered as the new destination node to run the last module and all remaining unmapped modules. We consider the following four cases:

- In case 1, where the maximum capacity of path P is not on any LCC link, the frame rate in G' is equal to the corresponding maximum capacity in G. Therefore, path P' from v'_s to v'_d in G' provides frame rate that is no less than $K = B$;
- In case 2, where the maximum capacity of path P is on one of the adjacent LCC links, we calculate the frame rate on the virtual node in G' as: $1/(f_w(z)/\frac{p}{\eta'})$. After plugging in $p = f_w(z) \cdot \gamma$, the frame rate becomes: $1/(f_w(z)/\frac{f_w(z)\cdot\gamma}{\eta'}) = \frac{\gamma}{\eta'}$. In WLCC problem, the bottleneck bandwidth x_{BN} of the widest path P has the following inequality: $K \leq x_{BN} \leq \frac{\gamma}{\eta'}$ since all constrained links share the capacity γ. Hence, the corresponding mapping path P' from v'_s to v'_d in G' provides frame rate at $\frac{\gamma}{\eta'} \geq K = B$;
- In case 3, where the maximum capacity of path P is on one of the nonadjacent LCC links, we calculate the frame rate on the virtual node in G' in the same way as we do in case 2, except for replacing η' with ζ';
- In case 4, where the maximum capacity of path P is on one of the mixed LCC links, we calculate the frame rate on the virtual node in G' in the same way as we do in case 2, except for replacing η' with $\eta' + \zeta'$.

Therefore, we conclude that the mapping path P' is the solution to the instance $\mathcal{I}_2$ of the MFR-ANRS problem.

Now we show that if there is a solution to the MFR-ANRS problem, we can also find a solution to the WLCC problem in polynomial time. Given a path P' from v'_s to v'_d in G' with frame rate $\geq B$, we first identify a corresponding path P in G. We also consider the following four cases:

- In case 1, where the frame rate is incurred on a network link in G', the corresponding network link in G has the bottleneck bandwidth of the widest path in WLCC problem;
- In case 2, where a virtual node converted from an adjacent LCC incurs the frame rate as: $1/(f_w(z)/\frac{p}{\eta'}) = 1/(f_w(z)/\frac{f_w(z)\cdot\gamma}{\eta'}) = \frac{\gamma}{\eta'} \geq B$, the corresponding path P in G has the bottleneck bandwidth $x_{BN} = \frac{\gamma}{\eta'} \geq B = K$ of the widest path in WLCC problem;
- In cases 3 and 4, the solution derivation steps are very similar to those in case 2, except for replacing η' with ζ' and $\eta' + \zeta'$, respectively.

Therefore, we conclude that path P in G from v_s to v_d is the solution to the instance $\mathcal{I}_1$ of the WLCC problem. This completes our proof. □

7 ELPC Heuristic Algorithms

We propose a set of solutions, Efficient Linear Pipeline Configuration (ELPC), in which a heuristic algorithm is designed for each mapping problem for MFR. We will also briefly present two other mapping algorithms used for performance comparison.

7.1 Dynamic Programming-Based Solutions

The MFR is achieved when the path BT is minimized. Let $T^{j-1}(v_i)$ denote the minimum BT with the first j modules mapped to a path from source node v_s to node v_i in an arbitrary computer network. We have the following recursion based on DP leading to the final solution $T^{m-1}(v_d)$:

$$\underset{j=2 \text{ to } m, v_i \in V}{T^{j-1}(v_i)} = \min\left(\begin{array}{c} \max\left(\begin{array}{c} T^{j-2}(v_i), \\ \dfrac{\alpha_i f_{w_{j-1}}(z_{j-2,j-1})}{p_i} \end{array} \right), \\ \min\limits_{v_u \in adj(v_i)} \left(\max\left(\begin{array}{c} T^{j-2}(v_u), \\ \dfrac{\alpha_i f_{w_{j-1}}(z_{j-2,j-1})}{p_i}, \\ \dfrac{\beta_{u,i} z_{j-2,j-1}}{b_{u,i}} + d_{u,i} \end{array} \right) \right) \end{array} \right), \tag{9}$$

with the base condition computed as:

$$\underset{v_i \in V, \text{ and } v_i \neq v_s}{T^1(v_i)} = \begin{cases} \max(f_{w_1}(z_{0,1})/p_i, \; z_{0,1}/b_{s,i} + d_{s,i}), & \forall e_{s,i} \in E, \\ \infty, & \text{otherwise}, \end{cases} \tag{10}$$

on the second column in the 2D table and $T^0(v_s) = 0$. Note that α_i denotes the number of modules assigned to node v_i and $\beta_{u,i}$ the number of data sets transferred over link between nodes v_u and v_i. In MFR-NNRS, $\alpha = 1$ and $\beta = 1$.

The recursive DP process fills out a 2D table, as shown in Fig. 6. Every cell $T^{j-1}(v_i)$ in the table represents a partial mapping solution that maps the first j modules to a path between source v_s and node v_i. During the mapping process, we consider two subcases at each recursive step, the minimum of which is chosen as the minimum BT to fill in a new cell $T^{j-1}(v_i)$: 1) In subcase 1, we run the new module on the same node running the last module in the previous mapping subproblem $T^{j-2}(v_i)$. In other words, the last two or more modules are mapped to the same node v_i. Therefore, we only need to compare the computing time of the last module on node v_i with the previous BT and use the larger as the current BT, which is represented by a direct incident link from its left neighbor cell in the 2D table. Since the resource is shared by multiple modules assigned to the same node, adding the new module to the current node v_i may change the optimality of the solution $T^{j-2}(v_i)$ to the previous mapping subproblem, resulting in a nonoptimal final solution and 2) In subcase 2, the new module is mapped to node v_i, and the last node v_u in a previous mapping subproblem $T^{j-2}(v_u)$ is one of the neighbor nodes

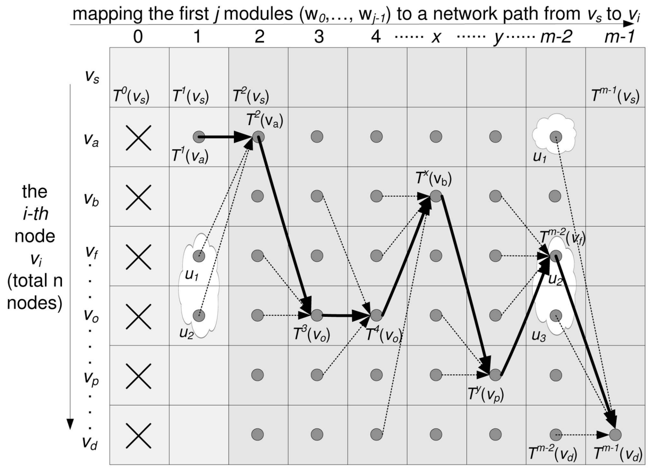

mapping the first j modules (w0,…, wj-1) to a network path from vs to vi
0 1 2 3 4 …… x …… y …… m-2 m-1
vs
va
vb
vf
vo
vp
vd
the i-th node vi (total n nodes)
T0(vs) T1(vs) T2(vs) Tm-1(vs)
T2(va)
T1(va)
Tx(vb)
Tm-2(vf)
T3(vo) T4(vo)
Ty(vp)
Tm-2(vd) Tm-1(vd)
u1 u2 u3

Fig. 6. Construction of 2D matrix using DP.

of node v_i, which is represented by an incident link from a neighbor cell on the left column to node v_i. In Fig. 6, a set of neighbor nodes $adj(v_i)$ of node v_i are enclosed in a cloudy region in the previous column. We calculate the BT for each mapping of an incident link of node v_i and choose the minimal one, which is further compared with the one calculated in subcase 1. The minimum of these two subcases is selected as the minimum BT for the partial computing pipeline mapping to a path between node v_s and node v_i. The complexity of this algorithm is $O(m \cdot |E|)$, where m denotes the number of modules in the linear computing pipeline and $|E|$ is the number of edges in the network.

The ELPC algorithm for MFR-ANRS exactly follows the DP procedure defined in (9) and (10), and its pseudocode is shown in Algorithm 1. The algorithms for MFR-CNRS and MFR-NNRS are similar with slight modifications. In MFR-CNRS, we simply skip a cell if its corresponding node has been used in the path selection so far to ensure a loop-free path, while in MFR-NNRS, each cell is only calculated from its adjacent nodes and is skipped if its corresponding node has been used. We would like to point out that these solutions are heuristic in nature because when a node has been selected by all its neighbor nodes at previous optimization steps, we may miss an optimal solution if this node is the only one leading to the destination node or obtain a suboptimal solution if there are multiple nodes leading to the destination node.

Algorithm 1. ELPC. **Input:** A linear computing pipeline with m modules, a heterogenous network $G = (V, E)$, where $|V| = n$, and a pair of nodes (v_s,v_d) in V representing the source and destination, respectively.
Output: The MFR of the pipeline mapped onto a selected network path P from v_s to v_d.

```
Initialize the first column of 2D table: $T_{[i][0]} = NULL$,
$i = 0, 1, \ldots, n-1$;
for all nodes $v_i$ from $i = 0$ to $n-1$ with only two modules
$w_0$ and $w_1$ do
  if $e_{s,i} \in E$ then
    Calculate the BT for cell $T_{[i][1]}$ in the 2nd column as the
    base condition;
  else
    $T_{[i][1]} = +\infty$;
  end if
end for
for all modules $w_j$ from $j = 2$ to $m-1$ do
  for all nodes $v_i$ from $i = 0$ to $n-1$ do
    if module $w_{j-1}$ is mapped to node $v_i$ then
      Map module $w_j$ to node $v_i$, calculate $BT_1$;
    else
      for all neighbor nodes $adj(v_i)$ directly connected to
      $v_i$ do
        Map module $w_j$ to node $v_i$, calculate $BT(adj(v_i))$;
      end for
    end if
    Choose the minimum BT among all neighbor nodes of
    $v_i$: $BT_2 = \min(BT(adj(v_i)))$;
    $T_{[i][j]} = \min(BT_1, BT_2)$;
  end for
end for
return $1/T_{[n-1][m-1]}$ as the MFR.
```

7.2 Algorithms for Comparison

7.2.1 Streamline Algorithm

Agarwalla et al. proposed a grid scheduling algorithm, *Streamline*, for graph dataflow scheduling in a network with n resources and $n \times n$ communication edges. The Streamline algorithm considers application requirements in terms of per-stage computation and communication needs, application constraints on colocation of stages (node reuse), and availability of computation and communication resources. Two parameters, *rank* and *blevel*, are used to quantitate these features: *rank* calculates the average computation and communication cost of a stage, and *blevel* estimates the overall remaining execution time of a data item after being processed by a stage. Based on these two parameters, the stages are sorted in a decreasing order of resource needs and the nodes are sorted in a decreasing order of resource availability. This scheduling heuristic works as a global greedy algorithm that expects to maximize the throughput of an application by assigning the best resources to the most needy stages in terms of computation and communication requirements at each step. The complexity of this algorithm is $O(m \cdot n^2)$, where m is the number of stages or modules in the dataflow graph and n is the number of resources or nodes in the network.

7.2.2 Greedy Algorithm

A greedy algorithm iteratively obtains the greatest immediate gain based on certain local optimality criteria at each step, which may or may not lead to the global optimum. We design a heuristic mapping scheme based on a greedy approach that calculates the pipeline bottleneck for the mapping of a new module onto the current node when node reuse is allowed on one of its neighbor nodes and chooses the maximum one. This greedy algorithm makes a module mapping decision at each step only based on current information without considering the effect of this local decision on the mapping performance in later steps. The complexity of this algorithm is $O(m \cdot |E|)$, where m denotes the number of modules in the linear computing pipeline and $|E|$ is the number of links in the network.

TABLE 1
MFR Performance Comparison among Three Algorithms under Different Constraints

Case #	# of Modules (m), Nodes (n), Links (l)	MFR-ANRS (frm/sec)			MFR-CNRS (frm/sec)		MFR-NNRS(frm/sec)		
		ELPC	Greedy	Streamline	ELPC	Greedy	ELPC	Greedy	Streamline
1	m=4 n=6 l=35	22.10	10.97	4.75	11.36	11.36	33.47	10.97	28.15
2	m=6 n=10 l=96	36.50	36.50	26.31	48.54	39.19	47.93	36.50	6.19
3	m=10 n=15 l=222	63.33	59.31	4.18	60.15	54.06	54.33	24.81	6.10
4	m=13 n=20 l=396	76.01	56.67	26.50	54.21	12.58	56.67	30.44	5.68
5	m=15 n=25 l=622	36.98	35.33	26.23	51.25	46.18	37.64	5.07	8.48
6	m=19 n=28 l=781	48.92	48.92	48.90	54.21	50.07	48.92	19.77	3.21
7	m=22 n=31 l=958	67.95	53.74	57.82	49.52	32.62	47.62	25.85	6.51
8	m=26 n=35 l=1215	61.52	57.90	47.41	21.42	14.61	38.70	30.95	10.33
9	m=30 n=40 l=1598	96.49	57.90	13.63	40.61	6.13	52.51	32.13	2.42
10	m=35 n=45 l=2008	36.46	36.46	27.17	32.19	7.70	36.46	24.09	3.65
11	m=38 n=47 l=2200	45.09	15.18	45.09	61.45	57.86	45.09	24.97	7.69
12	m=40 n=50 l=2478	27.43	27.43	27.42	60.57	55.57	27.43	25.65	8.07
13	m=45 n=60 l=3580	61.01	49.53	55.24	34.34	32.25	52.59	18.61	4.62
14	m=50 n=65 l=4220	89.65	59.70	56.56	52.50	16.99	43.92	10.61	2.10
15	m=55 n=70 l=4890	50.80	50.80	35.45	40.97	34.13	50.80	22.20	5.85
16	m=60 n=75 l=5615	94.00	53.82	45.19	50.23	49.92	43.41	25.19	3.95
17	m=75 n=90 l=8080	15.35	15.35	15.35	50.52	50.02	15.35	29.04	1.17
18	m=80 n=100 l=9996	32.60	28.90	32.59	54.25	32.38	32.60	18.32	6.96
19	m=90 n=150 l=22476	88.95	55.49	24.47	56.07	41.54	55.63	35.58	6.52
20	m=100 n=200 l=39990	84.94	55.98	20.68	56.86	34.31	56.70	18.35	4.86

8 Simulation-Based Performance Evaluation

8.1 Simulation Settings

The proposed ELPC algorithms are implemented in C++ and run on a PC equipped with a 3.0 GHz CPU and 2 Gbytes memory. For performance comparison purposes, we implement the other two algorithms, namely, Streamline and Greedy, in C++ on the same computing platform. We conduct an extensive set of mapping experiments for MFR using a wide variety of simulated application pipelines and computing networks. We generate these simulation data sets by randomly varying the following pipeline and network attributes within a suitably selected range of values: 1) the number of modules as well as the complexity and input/output data sizes of each module; 2) the number of nodes as well as the processing power of each node; and 3) the number of links as well as the link bandwidth and minimum link delay of each link.

For each mapping problem, we designate a pair of source and destination nodes to run the first module and the last module of the pipeline. This is based on the consideration that the system knows where the raw data is stored and where an end user is located before optimizing the pipeline configuration over an existing network.

8.2 Performance Comparison

With the simulated application pipelines and computing networks, we perform an extensive set of simulations of pipeline mapping with different constraints using ELPC, Streamline, and Greedy, respectively. The measured execution time of these algorithms varies from milliseconds for small-scale problems to seconds for large-scale ones. A set of typical MFR performance measurements is tabulated in Table 1 for comparison, which are collected in 20 cases using different problem sizes from small to large specified in the second column. The relative performance differences of these three algorithms observed in other cases are qualitatively similar.

For a visual comparison, we plot the MFR performance measurements produced by these three algorithms under different constraints with three sets of samples in Figs. 7, 8, and 9, respectively. We observe that ELPC exhibits comparable or superior performances in maximizing frame rate over the other two algorithms in all the cases that we studied. We do not compare with Streamline in the case of contiguous node reuse because Streamline does not allocate the resources by the stage's (module's) sequence number so that the previous node is unknown when the current one is being allocated. The MFR, i.e., the reciprocal of the BT in a selected path, is not particularly related to the path length, and hence, these MFR performance curves lack an obvious increasing or decreasing trend in response to varying problem sizes.

9 Implementation and Experimental Results

We implement a proof-of-concept system for RICSA in Java, C++, and MPI Fortran on Linux using Google Web Toolkit (GWT) [12] for the Ajax Web developments. In this section, we describe the implementation details and present experimental results by several network deployments.

9.1 Graphical User Interface

Fig. 10 displays a screenshot of the graphical user interface (GUI) of RICSA developed using GWT. The sod shock tube

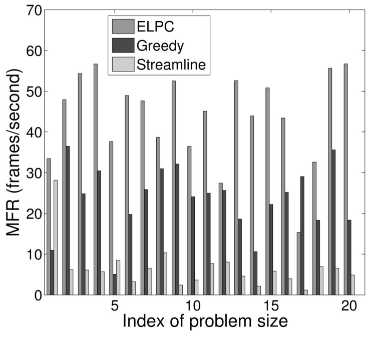


Fig. 7. MFR-NNRS comparison.

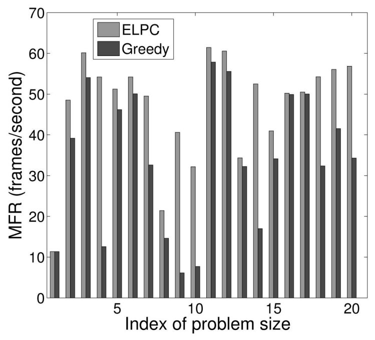


Fig. 8. MFR-CNRS comparison.

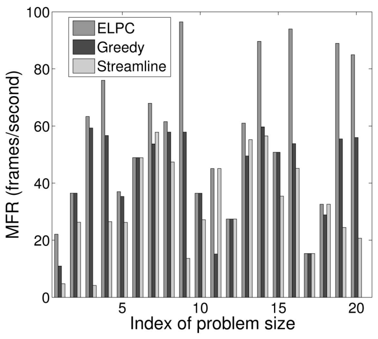
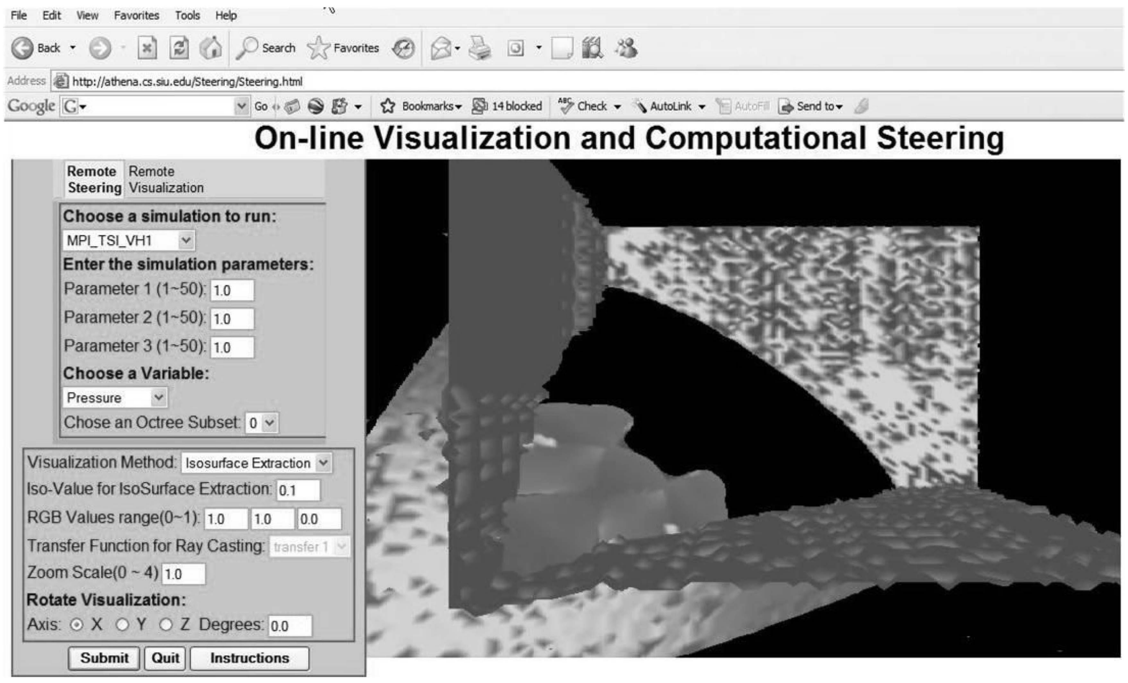


Fig. 9. MFR-ANRS comparison.

simulation, a classical hydrodynamics problem, is running on a Linux cluster of eight nodes for parallel computation and visualization. Each newly generated simulation data set traverses through a linear visualization pipeline over the wide-area network using various scientific visualization techniques to achieve MFR with available distributed computing resources. By interacting with the Web components in a browser, a user can choose from a list of serial or parallel simulation programs to run, specify computation control parameters, and select visualization parameters such as the variable of interest, octree subset region, visualization technique, and viewing parameters including zoom factor and rotation angle. Direct mouse interactions with the image will also trigger the rerendering of images. While the simulation process is running, the user can dynamically steer simulation/visualization parameters based on visual feedback. When a new image arrives at the client side, only the image component in the user interface will be updated and the rest of the components remain unchanged. Such a data-driven model from Ajax technology makes our Web application more responsive compared with traditional Web technologies. In addition to real-time steering, RICSA can also support remote visualization for archival data sets.

9.2 Universal Steering Framework for Various Simulation Programs

RICSA is designed as a universal framework to support various simulation programs written in different programming languages. The RICSA system can be easily modified to integrate any given simulation program. We achieve code reuse and system modularity by developing several generic C++ visualization and network Application Programming Interface (API) functions and packaging them in a shared library to be used by any simulation program. These API function calls are inserted at certain points in simulation code (typically developed in a non-C++ programming language in science fields) to set up socket communications, transfer data sets, and intercept steering commands from the client. Such system design and structure greatly improve code portability between different implementation platforms. Fig. 11 illustrates how six essential RICSA API functions are called at the appropriate locations in the computational loops of Virginia Hydrodynamics (VH1) simulation code written in Fortran [13].



Address http://athena.cs.siu.edu/Steering/Steering.html

On-line Visualization and Computational Steering

Choose a simulation to run: MPI_TSI_VH1
Enter the simulation parameters:
Parameter 1 (1~50): 1.0
Parameter 2 (1~50): 1.0
Parameter 3 (1~50): 1.0
Choose a Variable: Pressure
Chose an Octree Subset: 0
Visualization Method: Isosurface Extraction
Iso-Value for IsoSurface Extraction: 0.1
RGB Values range(0~1): 1.0 1.0 0.0
Transfer Function for Ray Casting: transfer 1
Zoom Scale(0 ~ 4) 1.0
Rotate Visualization:
Axis: [✓ X] Y Z Degrees: 0.0

Fig. 10. A screenshot of the RICSA GUI.

```
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;
                    Main computational loops
Do
    sweepx;
    sweepy;
    sweepz;
    RICSA_PushDataToVizNode();
    RICSA_ReceiveHandleMessage();
    if ( Message is NewSimulationParameters)
    RICSA_UpdateSimulationParameters();
While( cycels not endcycle)
                    End of main computational loops
```

Fig. 11. Visualization and network API function calls are inserted into the Virginia Hydrodynamics simulation program.

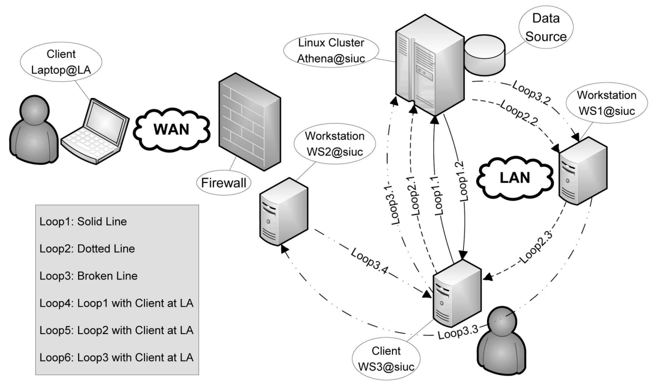


Fig. 12. Six network loops with different nodes.

9.3 Remote Visualization Experimental Results

We wish to demonstrate that the pipeline partition and mapping scheme chosen by our system outperforms all other alternatives in terms of frame rate in real scenarios. In the experiments, an MPI-based simulation of the Stellar Wind problem in astrophysics is visualized using ray casting technique on the same Linux cluster with eight nodes. We set up a LAN-WAN testbed consisting of five nodes: three PC workstations (WS1-3), one Linux cluster (Athena) located at Southern Illinois University, Carbondale (SIUC), and one laptop located in Los Angeles (LA) to compare the performance of different network loops, as shown in Fig. 12. For all network loops, the simulation node acts as the DS and the Web server is deployed on the Athena Linux cluster. The CS and CM nodes are mapped onto two Linux workstations WS1 and WS2. The Web client nodes are either selected to be the laptop in LA or the workstation WS3 at SIUC. In Fig. 12, we collect experimental results by running the system on three different pipeline mapping loops for the same hydrodynamics problem and visualization technique: 1) Loop 1 maps CM, DS, and CS nodes to Athena and client to WS3; 2) Loop 2 maps CM and CS to WS1 and client to WS3; and 3) Loop 3 maps CM to WS1, CS to WS2, and client to WS3. These three experiments are repeated with Loops 4, 5, and 6, respectively, using the client deployed in LA. These six node deployment schemes are tabulated in Table 2. We measure and plot the frame rates in unit of frames per second (fps) achieved at the Web clients by all six pipeline loops in Fig. 13. We observe that Loops 2 and 5, selected by ELPC, achieved the highest frame rate in each set of experiments.

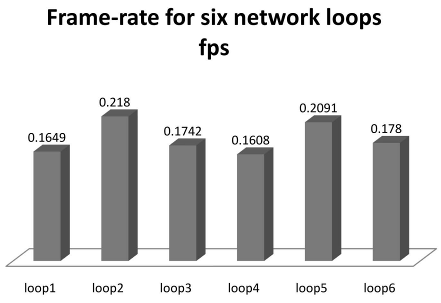


Fig. 13. The frame rates achieved with different network loops.

Our experiments further illustrate that different pipeline mapping schemes will lead to considerably different streaming performance. Such performance disparities will become more evident if applications are distributed across WAN with heterogeneous computing resources and time-varying network conditions. Thus, dynamically choosing an efficient network configuration is critical to achieving smooth data flow with satisfactory network performance. It is also interesting to note that different clients did not cause noticeable difference on the frame rate, which indicates that the bottleneck is not on the transport link from the Web server to the client. However, if we conduct remote visualization operations for a single archived data set, a remote Web client will incur longer end-to-end delay due to the much longer image transport time. We also observe that Loop 2 utilized one extra computing node and two extra transport links than Loop 1 but produced a higher frame rate. This is due to the fact that the bottleneck of Loop 1 is on the computation modules mapped to the heavily loaded Athena and the

TABLE 2
Six Node Deployment Schemes in the Experiments

Loop #	Client	Web Server	Central Management	Data Source	Computing Service
Loop 1	Workstation 3 @ SIUC	Athena			
Loop 2	Workstation 3 @ SIUC	Athena	Workstation 1	Athena	Workstation 1
Loop 3	Workstation 3 @ SIUC	Athena	Workstation 1	Athena	Workstation 2
Loop 4	Laptop @ Los Angeles	Athena			
Loop 5	Laptop @ Los Angeles	Athena	Workstation 1	Athena	Workstation 1
Loop 6	Laptop @ Los Angeles	Athena	Workstation 1	Athena	Workstation 2

utilization of WS2 in Loop 2 is able to share the computing load with lower BT despite two extra data transport links.

10 Conclusion and Future Work

The objective of the proposed RICSA system is to visualize, monitor, and steer computations over wide-area networks to support large-scale scientific applications. We constructed mathematical models for mapping a visualization pipeline to networks, and classified the pipeline mapping problems for MFR into three classes. We proved the NP-completeness of these problems and proposed a DP-based approach to compute an efficient visualization pipeline configuration that maximizes the frame rate of the system. The performance of the proposed solutions was evaluated using simulation results in comparison with existing algorithms and was also verified using experimental results collected on a prototype implementation deployed over the Internet.

The proposed system has been extensively tested and used by teams of geographically distributed scientists in various Internet environments. Through this work with focus on both practical and theoretical aspects, we learned that implementing a practical computational steering system with a set of fully working functionalities for real distributed applications requires significantly greater efforts than formulating the underlying problems and designing computing solutions in a theoretical framework. In some cases, the efficiency of the implementation is just as important as that of the algorithms. Special caution also needs to be taken when the system is deployed in network environments with restrictive firewall settings that only allow unidirectional communication. The use of newly emerged Web development techniques such as Ajax greatly improves the friendliness of the system and extends the usability to end users who do not own well-equipped end hosts.

It is 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 large-scale simulation programs from different disciplines such as biology, chemistry, and physics. In addition to the Internet, we plan to deploy the system over dedicated networks, such as DOE UltraScience Net [27], for experimental testing especially for large data sets. We plan to incorporate new transport methods that can 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

The authors would like to thank Anne Benoit and Yves Robert at ENS Lyon for their help with the NP-completeness proof. The authors would also like to thank the anonymous reviewers for their constructive comments that considerably improved the quality of this paper. This research is sponsored by the US National Science Foundation under Grant No. CNS-0721980 and the Oak Ridge National Laboratory, US Department of Energy, under Contract No. PO 4000056349 with the University of Memphis and Contract No. PO 6400006042 with Southern Illinois University, Carbondale.

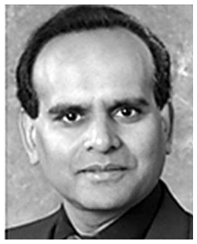
References

[1] Terascale Supernova Initiative (TSI), http://www.phy.ornl.gov/tsi, 2009.
[2] ParaView, http://www.paraview.org/HTML/Index.html, 2009.
[3] ASPECT, http://www.aspect-sdm.org, 2009.
[4] Computational Eng. Int'l, http://www.ceintl.com/products/ensight.html, 2009.
[5] CUMULVS, http://www.csm.ornl.gov/cs/cumulvs.html, 2009.
[6] RealityGrid, http://www.realitygrid.org, 2009.
[7] Parallel Virtual Machine, http://www.csm.ornl.gov/pvm, 2009.
[8] AVS, http://www.avs.com, 2009.
[9] Common Data Format, http://nssdc.gsfc.nasa.gov/cdf, 2009.
[10] Hierarchical Data Format, http://hdf.ncsa.uiuc.edu, 2009.
[11] Network Common Data Form, http://my.unidata.ucar.edu/content/software/netcdf, 2009.
[12] GWT, http://code.google.com/webtoolkit/, 2009.
[13] VH1, http://wonka.physics.ncsu.edu/pub/VH-1, 2009.
[14] B. Agarwalla, N. Ahmed, D. Hilley, and U. Ramachandran, "Streamline: A Scheduling Heuristic for Streaming Application on the Grid," *Proc. 13th Multimedia Computing and Networking Conf.*, 2006.
[15] A. Benoit and Y. Robert, "Mapping Pipeline Skeletons onto Heterogeneous Platforms," *Proc. Int'l Conf. Computational Science,* Y. Shi, D. van Albada, J. Dongarra, and P. Sloot, eds., pp. 591-598, 2007.
[16] 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," *Proc. ACM/IEEE Conf. Supercomputing,* 2000.
[17] I. Bowman, J. Shalf, K. Ma, and W. Bethel, "Performance Modeling for 3d Visualization in a Heterogeneous Computing Environment," Technical Report 2005-3, Dept. of Computer Science, Univ. of California at Davis, 2005.
[18] K. Brodlie, D. Duce, J. Gallop, M. Sagar, J. Walton, and J. Wood, "Visualization in Grid Computing Environments," *Proc. IEEE Conf. Visualization,* pp. 155-162, 2004.
[19] S. Fortune, J. Hopcroft, and J. Wyllie, "The Directed Subgraph Homeomorphism Problem," *Theoretical Computer Science,* vol. 10, pp. 111-121, 1980.
[20] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1979.
[21] A. Kaufman, *Trends in Visualization and Volume Graphics, Scientific Visualization Advances and Challenges.* CS Press, 1994.
[22] H.J. Kushner and D.S. Clark, *Stochastic Approximation Methods for Constrained and Unconstrained Systems.* Springer-Verlag, 1978.
[23] E.J. Luke and C.D. Hansen, "Semotus Visum: A Flexible Remote Visualization Framework," *Proc. IEEE Conf. Visualization,* pp. 61-68, 2002.
[24] M. Oberhuber, S. Rathmayer, and A. Bode, "Tuning Parallel Programs with Computational Steering and Controlled Execution," *Proc. 31st Hawaii Int'l Conf. System Sciences,* vol. 7, pp. 157-166, 1998.
[25] S.G. Parker, D.M. Weinstein, and C.R. Johnson, "The Scirun Computational Steering Software System," *Modern Software Tools in Scientific Computing,* Birkhaüser Boston, pp. 1-44, 1997.
[26] N.S.V. Rao, Q. Wu, and S.S. Iyengar, "On Throughput Stabilization of Network Transport," *IEEE Comm. Letters,* vol. 8, no. 1, pp. 66-68, Jan. 2004.
[27] N.S.V. Rao, W.R. Wing, S.M. Carter, and Q. Wu, "Ultrascience Net: Network Testbed for Large-Scale Science Applications," *IEEE Comm. Magazine,* vol. 43, no. 11, pp. s12-s17, Nov. 2005. http://www.csm.ornl.gov/ultranet.
[28] S. Stegmaier, M. Magallon, and T. Ertl, "A Generic Solution for Hardware-Accelerated Remote Visualization," *Proc. Symp. Data Visualization,* pp. 87-95, 2002.
[29] Q. Wu, Y. Gu, M. Zhu, and N.S.V. Rao, "Optimizing Network Performance of Computing Pipelines in Distributed Environments," *Proc. 22nd IEEE Int'l Parallel and Distributed Processing Symp.,* Apr. 2008.
[30] Q. Wu, N.S.V. Rao, and S.S. Iyengar, "On Transport Daemons for Small Collaborative Applications Over Wide Area Networks," *Proc. 24th IEEE Int'l Performance Computing and Comm. Conf.,* Apr. 2005.
[31] Y. Zhu and B. Li, "Overlay Network with Linear Capacity Constraints," *IEEE Trans. Parallel and Distributed Systems,* vol. 19, no. 2, pp. 159-173, Feb. 2008.

Qishi Wu received the BS degree in remote sensing and GIS from Zhejiang University, People's Republic of China, in 1995, the MS degree in geomatics from Purdue University in 2000, and the PhD 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 in the Department of Computer Science at the University of Memphis. His research interests include computer networks, remote visualization, distributed sensor networks, high-performance computing, algorithms, and artificial intelligence. He is a member of the IEEE.

Mengxia Zhu received the PhD degree in computer science from Louisiana State University in 2005. She spent two years in the Computer Science and Mathematics Division at Oak Ridge National Laboratory for her PhD dissertation from 2003 to 2005. She is currently an assistant professor in the Computer Science Department at Southern Illinois University, Carbondale. Her research interests include distributed and high-performance computing, remote visualization, bioinformatics and sensor networks. She is a member of the IEEE.

Yi Gu received the BS degree in computer science from Jiangsu University, People's Republic of China, in 2005, and the MS degree in computer science from the University of Memphis in 2008. She is currently working toward the PhD degree in the Department of Computer Science at the University of Memphis. Her research interests include parallel and distributed computing, high-performance networks, and wireless sensor networks. She is a student member of the IEEE.



Nageswara S.V. Rao received the BTech degree from the National Institute of Technology, Warangal, India, in electronics and communications engineering in 1982, the ME degree in computer science and automation from the Indian Institute of Science, Bangalore, in 1984, and the PhD degree in computer science from Louisiana State University in 1988. He is currently a corporate fellow in the Computer Science and Mathematics Division, Oak Ridge National Laboratory, where he joined in 1993. He has been on assignment at the Missile Defense Agency as technical director of the C2BMC Knowledge Center since 2008. He was an assistant professor in the Department of Computer Science at Old Dominion University during 1988-1993. He has published more than 250 technical conference and journal papers in the areas of sensor networks, information fusion, and high-performance networking. He is a fellow of the IEEE and received the 2005 IEEE Technical Achievement Award for his contributions in the information fusion area.