

ADAPTIVE REMOTE VISUALIZATION SYSTEM WITH OPTIMIZED NETWORK PERFORMANCE FOR LARGE SCALE SCIENTIFIC DATA

A Dissertation
Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in
The Department of Computer Science

by
Mengxia Zhu
B.S., Zhejiang University, P.R. China, 1996
M.S., Louisiana State University, 2002
December 2005

Copyright 2005
Mengxia Zhu
All rights reserved

This dissertation is dedicated to my beloved husband, Qishi.

Acknowledgements

First of all, I would like to express my sincere gratitude and appreciation to my advisor, S.S.Iyengar and co-advisor N.S.V.Rao, for their inspiration to guide me to a deeper understanding of knowledge, and their invaluable comments during the whole research work on this dissertation. They always push the envelope of my ultimate research ability and never accept less than my best efforts.

I will also give special thanks to my husband Qishi Wu for a fruitful collaboration. I am privileged to have him as my life-time partner as well as my academic colleague. I am very grateful to Song Ding, Jinzhu Gao, and Jian Huang who all have worked with me on the implementation of the system. Your contribution is highly appreciated.

I also want to thank everyone who has read this manuscript and given constructive comments, especially my committee members, Richard Brooks, Fereydoun Aghazadeh, Bijaya B. Karki, Rajgopal Kannan, and Ding Shih. Thanks also go to my colleagues in the Division of Computing Science and Mathematics at Oak Ridge National laboratory for providing a good working atmosphere, especially the persons in the Networking and Visualization groups who have provided me access to their visualization and networking facilities which are essential for my experimental tests.

The research in this dissertation has been supported in part by the High Performance Networking Program of the Office of Science, U.S. Department of Energy, under contract No DE-AC05-00OR22725 with UT-Battelle, LLC.

Finally, I would like to thank my parents and brother for their persistent love.

Table of Contents

ACKNOWLEDGEMENTS	iv
LIST OF TABLES	vii
LIST OF FIGURES	viii
ABSTRACT	x
CHAPTER 1. INTRODUCTION.....	1
1.1 Large-scale Scientific Data.....	1
1.2 High-performance Computer Network.....	1
1.3 Commercial and Open-source Visualization Software	2
1.4 Our Approaches.....	4
1.5 Dissertation Organization.....	6
CHAPTER 2. LITERATURE REVIEW.....	7
2.1 Scientific Visualization Basics.....	7
2.1.1 Grid, Lattice and Sampling.....	8
2.1.2 Interpolation Function.....	9
2.1.3 Data Classification.....	9
2.1.4 View and Shading.....	10
2.2 Surface Fitting Techniques.....	10
2.2.1 Marching Cubes.....	10
2.2.2 Dividing Cubes.....	11
2.3 Volume Rendering Techniques.....	11
2.3.1 Object Order Splatting.....	12
2.3.2 Image Order Ray Casting.....	13
2.4 Algorithm Optimization Techniques.....	15
2.5 Parallel Rendering Techniques.....	16
2.6 Remote Visualization System.....	18
2.6.1 Efforts on Visualziation Techniques.....	18
2.6.2 Efforts on Network Transport and System Architecture.....	19
CHAPTER 3. ANALYTICAL MODEL.....	21
3.1 Minimal Total Delay.....	22
3.2 Maximal Frame Rate.....	23
CHAPTER 4. REMOTE COMPUTATIONAL MONITORING AND STEERING.	24
4.1 Introduction.....	24
4.2 Terascale Supernova Initiative Project.....	24
4.3 Imminent Challenges and Strategic Solutions.....	26

CHAPTER 5. REMOTE WIRELESS SENSOR NETWORK MONITORING.....	29
5.1 Introduction.....	29
5.2 Periodic Environment Monitor.....	30
5.2.1 Sensor Data Buffer and Dispatch Protocol.....	30
5.2.1.1 Relative Approach.....	32
5.2.1.2 Timeout Approach.....	33
5.3 Event Detection.....	33
5.3.1 Threshold-OR Fusion Method.....	33
5.3.1.1 Problem Formulation.....	36
5.3.1.2 Threshold Bounds Derivation.....	38
5.3.1.3 Simulation and Results Discussion.....	41
5.3.1.4 Conclusions and Future Work.....	44
CHAPTER 6. TECHNICAL SOLUTIONS FOR NETWORK OPTIMIZATION....	46
6.1 Computing Power Measurement for Processing Time Estimation.....	46
6.1.1 Addressing Accumulated Variance.....	47
6.1.2 Marching Cubes.....	47
6.1.3 Raycasting.....	50
6.1.4 Streamline Construction.....	51
6.2 Bandwidth Measurement for Transport Time Estimation.....	51
6.3 System Optimization Based on Dynamic Programming.....	53
6.3.1 Minimal Total Delay.....	54
6.3.1.1 Surjective Mapping with Linearly Arranged Network Nodes..	55
6.3.1.2 Injective Mapping with Arbitrary Network.....	57
6.3.1.3 A General Mapping Scheme.....	58
6.3.2 Maximal Frame Rate.....	59
6.3.2.1 Surjective Mapping with Linearly Arranged Network Nodes.	60
6.3.2.2 Injective Mapping with Arbitrary Network.....	61
6.3.2.3 A General Mapping Scheme.....	61
CHAPTER 7. SYSTEM ARCHITECTURE AND IMPLEMENTATION.....	63
7.1 Post Processing System.....	63
7.1.1 System Function Diagram.....	63
7.1.2 Message-based Control Flow.....	65
7.2 Computational Monitoring and Steering System.....	67
7.3 Wireless Sensor Network Monitoring System.....	69
CHAPTER 8. EXPERIMENTAL RESULTS.....	74
8.1 Experiment Network Configuration.....	74
8.2 Performance Comparison among Loops.....	74
8.3 Performance Comparison with ParaView.....	78
CHAPTER 9. CONCLUSION.....	81
BIBLIOGRAPHY.....	83
VITA.....	88

List of Tables

5.1	Numeric results with different deployment radius for heterogenous system.....	43
5.2	Comparison of different decision rules based fusion.....	44
6.1	Problem category of pipeline decomposition and network mapping.....	53

List of Figures

1.1	A general visualization pipeline: Visualization modules and data flow.....	3
1.2	A general remote visualization system in client and server architecture.....	3
2.1	A single voxel.....	8
2.2	Marching cubes cases.....	11
2.3	Convolution of data samples.....	12
2.4	Ray casting from eye to objects.....	14
2.5	Ray casting pipeline.....	14
2.6	Back to front and front to back depth order.....	14
2.7	Sort-first(left) and sort-last(right)[31].....	16
3.1	Mathematical model for pipeline partitioning and network mapping.....	22
4.1	Schematic architecture of a simple computational steering application.....	25
4.2	Crayx1 supercomputer at ORNL.....	26
5.1	Framework of remote visualization system for WSN.....	30
5.2	Relative data buffer model.....	31
5.3	Timeout data buffer model.....	31
5.4	Sensor deployment for target detection.....	34
5.5	Normal distribution based hit rate and false alarm rate calculation for heterogenous system.....	36
5.6	Application of Chebyshev's inequality in calculation the lower bound of the system hit rate.....	39
5.7	Homogeneous system ROC curve with different sensor node number.....	42
5.8	Heterogenous system ROC curve with different deployment radius.....	42

6.1	The Average-case Number of Triangles(ANT) distribution obtained from four datasets, each tested with three randomly selected isovalues.....	48
6.2	The linear performance model of marching cubes: Different data points correspond to various iso-values chosen from the range between 30 and 150, assuming 8-bit values on the voxels.....	49
6.3	End-to-end delay measurements between ORNL and LSU.....	53
6.4	Dynamic programming for minimal transport delay time.....	55
6.5	Construction of a 2D matrix of T(i,j).....	55
6.6	Illustration of n-hops shortest path algorithm.....	57
6.7	Construction of 2D matrix in dynamic programming.....	59
7.1	System function diagram.....	64
7.2	System control flow diagram.....	68
7.3	Interaction diagram for computational steering modules	70
7.4	Simulation pseudocode for computational steering	71
7.5	Mica2 sensor mote	71
7.6	Sensor data viz window	72
8.1	Network configuration of distributed visualization experiment	75
8.2	Distributed visualization performance estimations and measurements along the optimal loop ORNL-LSU-GaTech-UT-ORNL	75
8.3	Performance comparisons between different visualization loops: Loop 1 along ORNL-LSU-GaTech-UT-ORNL (optimal), Loop 2 along ORNL-LSU GaTech-NCState-ORNL, Loop 3 along ORNL-LSU-OSU-NCState-ORNL, Loop 4 along ORNL-LSU-OSU-UT-ORNL, Loop 5 along ORNL-GaTech-ORNL (PC-PC), and Loop 6 along ORNL-OSU-ORNL (PC-PC).....	76
8.4	Performance comparisons between different visualization loops: optimal loop along ORNL-LSU-GaTech-UT-ORNL, ParaView –crs mode along ORNL-UT-GaTech.....	80

Abstract

This dissertation discusses algorithmic and implementation aspects of an automatically configurable remote visualization system, which optimally decomposes and adaptively maps the visualization pipeline to a wide-area network. The first node typically serves as a data server that 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 can be located anywhere on the network and often include workstations, clusters, or custom rendering engines. We employ a regression model-based network daemon to estimate the effective bandwidth and minimal delay of a transport path using active traffic measurement. Data processing time is predicted for various visualization algorithms using block partition and statistical technique. Based on the link measurements, node characteristics, and module properties, we strategically organize visualization pipeline modules such as filtering, geometry generation, rendering, and display into groups, and dynamically assign them to appropriate network nodes to achieve minimal total delay for post-processing or maximal frame rate for streaming applications. We propose polynomial-time algorithms using the dynamic programming method to compute the optimal solutions for the problems of pipeline decomposition and network mapping under different constraints. A parallel based remote visualization system, which comprises a logical group of autonomous nodes that cooperate to enable sharing, selection, and aggregation of various types of resources distributed over a network, is implemented and deployed at geographically distributed nodes for experimental testing. Our system is capable of handling a complete spectrum of remote visualization tasks expertly including post processing, computational steering and

wireless sensor network monitoring. Visualization functionalities such as isosurface, ray casting, streamline, linear integral convolution (LIC) are supported in our system. The proposed decomposition and mapping scheme is generic and can be applied to other network-oriented computation applications whose computing components form a linear arrangement.

Chapter 1

Introduction

1.1 Large-scale Scientific Data

Technological advances in scientific computations and experiments have generated data of unprecedented size, often on the scale of terabytes, presenting various challenges to current computing hardware, network infrastructures, storage devices, as well as to processing algorithms for people who want to analyze and visualize large data at remote ends in real time.

One government-sponsored project, the Advanced Simulation and Computing Initiative(ASCI) has implemented powerful and capacious computational infrastructure, such as supercomputers and clusters with speed measured in teraflops [17]. It is predicted that the Petaflops supercomputer will be built between 2005 and 2010. The generation of large, complex, and multidimensional datasets has been a common component in many scientific fields, including computational fluid dynamics, computational mechanics and computational combustion [19]. Manual data exploration and analysis is impractical and prone to error due to unwieldy data size and complexity. Visual representations extracted from raw data is a commonly employed technique to strengthen data comprehension through the capacity for keen perception in the human brain. Unfortunately, our ability to produce an overwhelming amount of data is not accompanied by similarly enhanced capabilities in data manipulation, network transport and efficient visualization.

1.2 High-performance Computer Network

Along with the mass of scientific data being produced every day, there is an insatiable demands for high network bandwidth for fast data shipment. Optical networks use

light waves rather than electron-based wire as the medium for data transmission or switching to achieve higher bandwidth. Optical fiber is now being laid out in the US at a rate of approximately 4k miles per day [14]. However, optical switching technology lags behind optical transmission. Thus, most deployed optical networks, such as SONET, are a combination of optical and electronic networks, which still rely on electronic equipment for data switching. Lambda optical network adopts the technology of using different “colors,” or “lambdas,” with different wavelengths of light on a single fiber; each lambda is capable of carrying its own data load without interfering the transmission of other lambdas. The current Wave Division Multiplexing(WDM) technique can carry approximately 32 to 80 wavelengths per fiber. Many dedicated optical network with a potential bandwidth up to 40Gbps, such as DOE UltraScience Net [6], NSF CHEETAH [3], and LSU LONI [10] are being constructed across the country at a furious pace.

Existing remote visualization systems could not capitalize on all the advantages provided by underlying high performance computer networks due to stiff system configurations and outdated transport means. Since most intermediate visualization data must be shipped within academic institutes having access to dedicated networks, such as universities, government agencies, and national laboratories, a remote visualization system that can take advantage of underlying massive bandwidth is very desired.

1.3 Commercial and Open-source Visualization Software

Fig. 1.1 shows a high-level general abstraction of a visualization pipeline along with the flow of data produced at each pipeline module.

In many scientific applications, raw data usually takes a multivariate format and is organized in structures, such as CDF, NetCDF, and HDF [4, 7, 12]. The preprocessing module filters interfering noises from either the background or the sensing device itself, and conducts necessary precomputation to shrink data size. The transformation module typically uses a surface-fitting technique (such as isosurface extraction) to derive 3D ge-

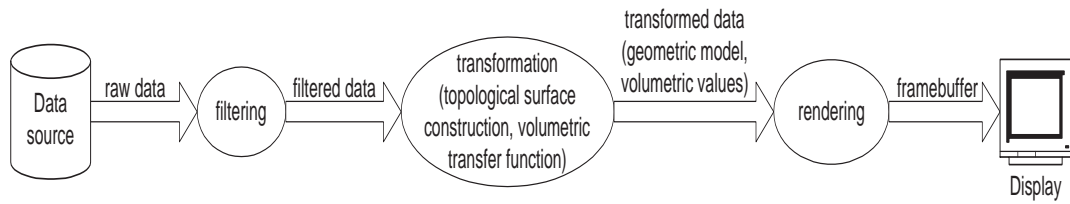


Figure 1.1: A general visualization pipeline: Visualization modules and data flow

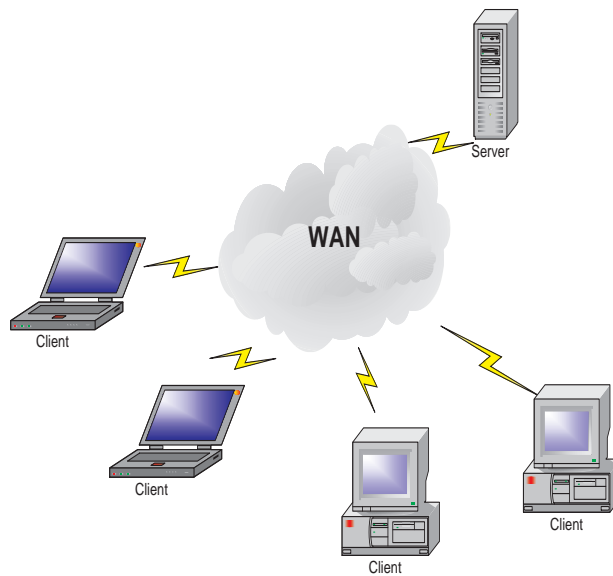


Figure 1.2: A general remote visualization system in client and server architecture.

ometries (such as polygons, target trajectory), or it uses transfer functions to perform color and opacity classifications. The 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 is stored in a frame buffer for final display on an end device. It should be noted that the visualization pipelines in real applications may significantly vary due to disparate implementation procedures and the use of different preprocessing and visualization techniques.

In general, a remote visualization system consists of a server operating as a data source, a client providing local rendering and display, zero or more intermediate hosts, and

a network connecting them all together. The overall system performance is affected not only by the capability and effectiveness of each individual component, but also relies to a great extent on how efficiently the visualization pipeline is decomposed and mapped to the network nodes.

Many commercial and open-source software products have emerged, such as TeraScaleBrowser, Vis5D+, ParaView, ASPECT, and Ensign for remote visualization [2, 5, 15, 18, 22]. However, they employ a predetermined partition of visualization pipelines. Most of them adopt a client and server architecture (Fig. 1.2), which can respond to multiple client requests simultaneously: the client submits a visualization request to the server; the server does the calculation and rendering tasks; the results, represented either as geometric objects or frame-buffers, are sent directly back to the client. While such schemes are common, easy to implement and widely used, they are not always optimal for the large-scale visualizations that typically require dynamic intelligent task decomposition and redistribution in order to deal with terabytes of data.

For example, for a remote visualization system exchanging geometric objects between server and client, a particular visualization entity might create a geometric data deluge, which unquestionably will throttle a low bandwidth link for an disproportionate period of time. Sending an image with fixed-size data would ease the burden on data transport. The transport problem is further compounded by limited network bandwidths and time-varying dynamics of network conditions, particularly over wide-area connections. Thus, a remote visualization system capable of selecting proper computing nodes and network links facilitating a visualization pipeline based on an integrated evaluation of current network resources and visualization tasks would be a significant advance in system performance.

1.4 Our Approach

Many research projects have been separately undertaken in diverse fields to develop state-of-the-art technologies for various aspects of this overall challenge. To date,

cutting-edge supercomputers such as IBM BlueGene and Cray X1, high speed networks such as DOE UltraScience Net [6], high performance storage systems, and large format displays and highly efficient graphics hardware have been in place or are being deployed. Typically, these resources are distributed in various laboratories and universities across the country or around the globe. A uniform application platform that integrates the existing component technologies is needed to fully utilize these resources and optimize the network performance.

We present a framework that addresses both the analytical and application aspects of realizing the optimal visualization pipeline decomposition and adaptive network mapping. Our primary design goal is to optimize the utilization of distributed data sources and system resources, such as computing power and network bandwidth, from the global perspective. The problem under investigation is formulated to minimize the total end-to-end delay, including both computation and transmission overheads, and to maximize the frame rate for a streaming application.

The resulting computational model enables us to analyze algorithmic aspects, such as complexity and the optimality of mapping the visualization pipeline onto a given network. Available system resources are estimated using domain-specific approaches: for computing time, we statistically estimated the cost for different visualization techniques on each available computing facility; for transport time, we employed a linear regression-based network daemon to measure the effective bandwidth and minimal delay of a transport path using active traffic measurement.

Based on cost estimation of the visualization modules, as well as the node and link measurements, we present the optimal decomposition and mapping schemes using the dynamic programming-based algorithm for total delay minimization or frame rate maximization. Finally, we designed and implemented our intelligent distributed visualization system to deal with scientific data from different sources, namely archived data for post

processing, on-the-fly simulation data for computational steering, and real-time wireless sensor network data for environmental monitoring.

1.5 Dissertation Organization

In Chapter 2, we review scientific visualization techniques and discuss the previous work on remote visualization. In Chapter 3, we describe the analytical model for the problem of pipeline partitioning and network mapping for minimal total delay and maximal frame rate. Chapter 4 discusses various aspects of the computational steering system for the Terascale Supernovae Initiative project. Chapter 5 proposes our data buffer dispatch protocol and decision fusion algorithm used by remote visualization on wireless sensor networks. In Chapter 6, we present the technical solutions for computing time and bandwidth estimation, optimized pipeline partition and network mapping for remote visualization systems based on dynamic programming. System design and implementation for postprocessing, computational steering, and wireless sensor networks are elaborated on in Chapter 7. Experimental results are provided in Chapter 8. We present our conclusions in Chapter 9.

Chapter 2

Literature Review

2.1 Scientific Visualization Basics

Scientific visualization, referred to SciVis, is a method of data manipulation and graphical representation that enables researchers to gain insights into their data and see the unseen; it can inspire profound and unexpected scientific discovery. Why does visualization play such a revolutionary role in science? The maxim that “one picture is worth a thousand words” is a perfect portrayal of the relationship. Scientists have found that half of the human brain is devoted directly or indirectly to vision [8]: “The human visual system can detect and discriminate between an incredibly diverse assortment of stimuli that may be chromatic or achromatic, stationary or mobile, patterned or un-patterned, two or higher dimensional” [1].

Recently, increased research efforts have been focused on volumetric data. Volumetric data can be obtained by simulation, modeling, and sampling techniques such as Computed Topography (CT) and Magnetic Resonance Imaging (MRI). As opposed to geometric data that is mathematically represented, volumetric data is typically a set of samples (x, y, z, d) with d representing the data property at a location determined by (x, y, z) . The set of samples can be defined on a regular grid with constant spacing along three axes. A rectilinear, curvilinear, or unstructured grid can also be applied. An unstructured grid needs to explicitly specify node connectivity. The data property at a certain location can take scalar, vector, or even tensor form. Voxelizing geometry primitives is another way to generate volumetric data.

Visualization techniques can be categorized into two areas: surface rendering techniques, and (direct) volume rendering techniques. Surface rendering is an indirect geometry-

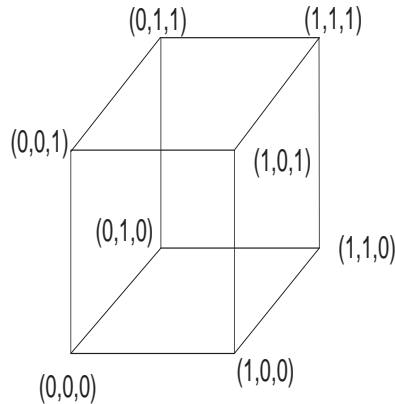


Figure 2.1: A single voxel

based technique, which first approximates a surface contained within the volume data using geometry primitives and then employs conventional computer graphics techniques to render those geometry primitives. The direct volume rendering technique transforms the entire 3D volume data into a single 2D image, in which both interior and exterior information is unveiled at the price of a higher computation cost. Amorphous phenomena, such as clouds, fire, and rivers, can only be modeled and rendered by the volume rendering technique due to the lack of tangible surfaces.

2.1.1 Grid, Lattices and Sampling

In most cases, data is taken at regular intervals along three orthogonal axes. The data set is referred to isotropic when the spacing along each axis is constant and equal, anisotropic when the three spacing constants are different. All elements are identical axis-aligned cubes in a cartesian grid. A regular grid has identical elements and axis-aligned rectangular parallelepiped (not cubes). A rectilinear grid with arbitrary spacing along each axis defines hexahedra and axis-aligned elements, which may not be identical. Elements on a structured grid are non-axis-aligned hexahedra. A block-structured grid is a set of structured grids that are put together to fill a space. Examples of structured grids include spherical and curvilinear lattices. An unstructured grid consists of polyhedra without ex-

PLICIT connectivity. A combination of above-mentioned grids is called a hybrid volume.

Some volumetric data is acquired by sampling. To view the details of size S , we need to sample at intervals of $S/2$. In other words, in order to view signal at frequency $f = 1/S$, the sampling frequency should be $2f = 2/S$. This is NYQUIST frequency.

2.1.2 Interpolation Function

The data type of an array S , holding some measured property of a point at discrete location (x, y, z) in space, can have scalar, vector, or tensor value. In order to describe the value at any location, an approximate function $f(x, y, z)$ over R^3 can be constructed by applying some interpolation function to array values. The simplest interpolation function is zero-order, which is essentially a nearest neighbor function. Another commonly used high order interpolation function is trilinear interpolation.

In Fig.2.1, for any point falling inside this voxel which has a length of one for each side, the interpolated value can be calculated from eight vertices. The value is assumed to vary linearly along direction parallel to three orthogonal axes.

$$f(x, y, z) = \sum_{i=0,1} \sum_{j=0,1} \sum_{k=0,1} (1-W_i)(1-K_j)(1-M_k)S(x_i, y_j, z_k) \quad (2.1)$$

where

$$W_i = |x - x_i|, K_j = |y - y_j|, M_k = |z - z_k| \quad (2.2)$$

2.1.3 Data Classification

Data classification is a difficult and important subject in scientific visualization. In surface fitting, it means the proper selection of isosurface value. In volume rendering, a transfer function that maps a scalar value to color and opacity values is usually refined by a series of trials for easy human observation. The opacity table is designed to expose interesting parts and hide trivial parts. For example, in CT data, bone density values would map to white/opaque; muscle density values map to red/semi-transparent; fat density values

map to beige/mostly-transparent [28]. Transfer function can be directly applied to scalar data or interpolated scalar data for visual attributes assignments.

2.1.4 View and Shading

Most volume rendering algorithms use orthographic viewing since perspective viewing is fraught with ray divergence problems. Both parallel and perspective views are very common in surface-fitting algorithm. Some techniques such as animation, depth-brightness attenuation, and shading are used to reflect depth clues when the parallel view is used [28]. A central difference formula, which measures how fast the value changes across adjacent points, is used to approximate the normal to an imaginary surface touching the point. The normals derived from the gradients at each point can be utilized by standard graphics shading models to produce a realistic image.

$$\begin{aligned}
 d(x) &= s(i+1, j, k) - s(i-1, j, k)/2; \\
 d(y) &= s(i, j+1, k) - s(i, j-1, k)/2; \\
 d(z) &= s(i, j, k+1) - s(i, j, k-1)/2;
 \end{aligned}
 \tag{2.3}$$

2.2 Surface Fitting Techniques

A binary segmentation function over volumetric data defines value to be 1 if that point belongs to an object, and zero otherwise. The surface is where the value changes from 0 to 1. However, with continuous interpolation function, an isosurface can be created with faces sharing the same value.

2.2.1 Marching Cubes

Marching cubes algorithm uses a triangle mesh to approximate an iso-valued surface. The basic principle is to subdivide space into a sequence of small cubes. The algorithm “marches” through each of the cubes establishing the intercepting points and replacing the cube with an appropriate set of polygons. A cube is interior if all the intensities of each corner are above surface value; it is exterior if all the intensities are below the surface value. Otherwise, an isosurface intersects the cube. Each cube has eight corners, which

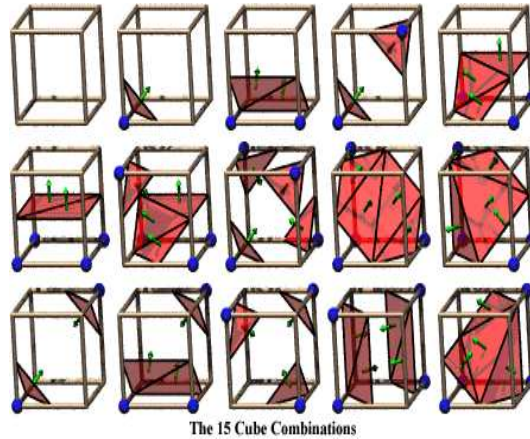


Figure 2.2: Marching cubes cases

can be either inside or outside of the shape, thus we have $2^8 = 256$ possible combinations. Topology symmetry reduces the number of different combinations to 15. Each of these 15 topologies has its own unique set of triangles pre-stored in a look-up table. The exact location of the vertices of those triangles can be determined by linear interpolation. Normal values can be computed using gradients to render smooth-shaded images.

2.2.2 Dividing Cubes

Marching cubes algorithm may generate a large number of triangles and many of them are cast into a single pixel on the raster display. As the number of such triangles increases, it becomes more efficient to directly display a point primitive other than a triangle on the screen. If an isosurface intersects a cube, the cube is further divided into sub-cubes, with each sub-cubes projecting onto a single pixel on the image plane.

2.3 Volume Rendering Techniques

The volume rendering technique investigates how to visualize a set of sampled, simulated scalar or vector data in a 3D setting without extract 3D primitives contained within the data. Projecting all voxels onto the view plane generates a 2D colored and semi-transparent image. The color and opacity for each voxel is calculated by a transfer function.

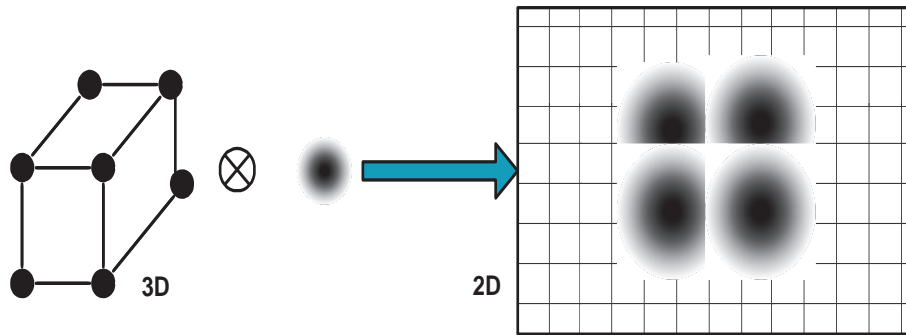


Figure 2.3: Convolution of data samples

Because all voxels are involved in the rendering process, the algorithm complexity and rendering time increases significantly. However, the ability to reveal not only the exterior, but also the interior structure contained with the data, makes volume rendering especially attractive in the field of medical image analysis.

2.3.1 Object Order Splatting

Splatting, first proposed by Westover in 90's [63], was to put the kernel of interpolation at the center of a data sample, which is further splatted onto the screen. This forward mapping of data evaluates the contribution of every 3D data sample to multiple pixels in 2D screen. Every data sample is convolved with a reconstruction kernel to produce a 3D image, which is projected on the 2D image plane. The projected area of each data sample forms the footprint of that data sample. The integration of all the footprints produces the final image. The Gaussian function is commonly used as the reconstruction kernel. Because convolution and integration are both linear operations, the order of operation can be reversed, that is, first integrate the kernel and then convolve with data. During the process of mapping to the image plane, the order of manipulating each voxel in the object cannot be arbitrary. If two voxels project to the same pixel, the one that is projected later overwrites the first one, even if it is farther from the image plane. People choose to traverse the data in either a back-to-front order or front-to-back order. In back-to-front order, the voxels,

which are further away from the viewer, will be mapped earlier onto the image plane. The alternative would be front-to-back order. Voxels are visited in an increasing distance to the viewer. Despite the increased complexity of implementing this order, efficiency is achieved by terminating some projection at early stages. In other words, when a voxel is projected onto a pixel, other voxles, which go to the same pixel, will be ignored. The traversal can be implemented by three nested loops, x , y and z , with z to be the outermost loop. Z is the axis parallel to the view direction. Color for pixel p on image is given by the integral along the ray

$$I(p) = \int f(p + s)ds \quad \text{where } f(r) = f(p + s) \text{ is density function,}$$

$$s \text{ is vector along the ray}$$

$f(r)$ is a function on 3D continuous space, and should be constructed from discrete

voxels. The interpolation can be done by:

$$f(r) = \sum_k w(r - r_k) f(r_k)$$

where $f(r_k)$ is density at sample k , $w(r - r_k)$ reconstruction kernel

$$I(p) = \int f(p + s)ds = \int \sum_k w(p + s - r_k) f(r_k) ds$$

$$\text{Thus} \quad = \sum_k f(r_k) \int w(p + s - r_k) ds$$

where $\int w(p + s - r_k) ds$ is called splat footprint.

In x, y, z coordinates(ray in z -axis), footprint is defined as: $splat(x, y) = \int w(x, y, z) dz$

Then, we obtain $I(x, y) = \sum_k f(r_k) splat(x - x_k, y - y_k)$ [16].

The splat of a Gaussian reconstruction function is: $\int e^{-(x^2/a^2 + y^2/b^2 + z^2/c^2)} dz$

2.3.2 Image Order Ray Casting

A ray is shot for every pixel in the image plane from the viewer through the volume. The color and opacity of every intercepted sample along the ray trajectory is integrated until a threshold is reached or the changing rate approaches zero. This backward mapping of data calculates the influence of all 3D data samples on each pixel in the 2D screen [42].

In comparison with object order techniques that determine how the object affects the pixels on the image plane, image order rendering determines the contribution of each object voxel to each pixel in the image plane. In Ray casting, every pixel in the image will

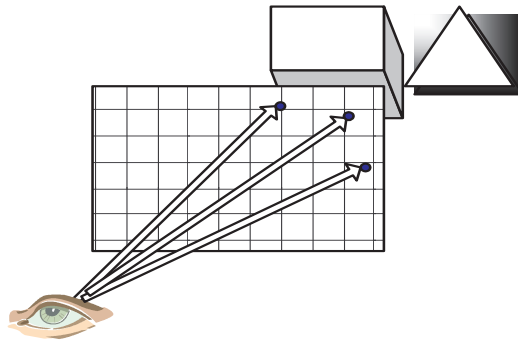


Figure 2.4: Ray casting from eye to objects



Figure 2.5: Ray casting pipeline

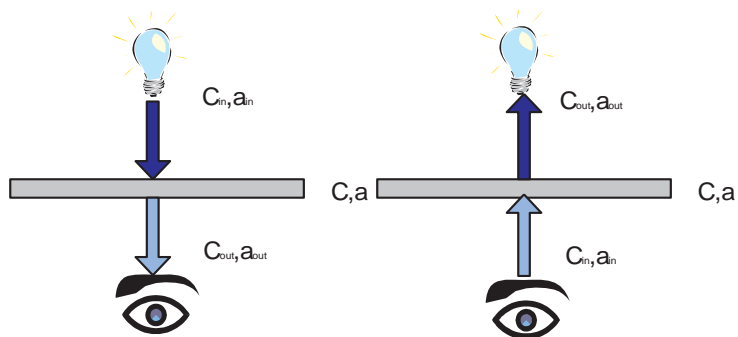


Figure 2.6: Back to front and front to back depth order

cast a ray through the volume. The ray intersects a line of voxels along its path. While the rays pass through the object, the color and opacity of the pixels are composited.

The merging of semi-transparent objects depends on two different depth orders: front to back and back to front. The color and opacity composition functions are:

$$C_{out} = C \times (1 - \alpha_{in}) + C_{in} \quad \alpha_{out} = \alpha \times (1 - \alpha_{in}) + \alpha_{in} \text{ for front to back.}$$

$$C_{out} = C_{in} \times (1 - \alpha) + C \quad \alpha_{out} = \alpha_{in} \times (1 - \alpha) + \alpha \text{ for back to front.}$$

There are several projection schemes for ray casting with different visual effects. We can choose either the maximum value along the ray trajectory and calculate the average value, or compute the composition value. Different visual effects can be noticed for above three composition methods.

2.4 Algorithm Optimization Techniques

In order to handle dramatically increasing sizes of datasets with limited computing capacity, much research effort have been directed to the development of new algorithms and the design of efficient data structures for conventional visualization techniques.

For example, in volume rendering, computing time for brute force algorithm grows linearly with the size of the dataset. A lower resolution image is sometimes generated for a quicker response. A pyramid data structure has $\log N$ layers for a dataset of N^3 samples, with N^3 samples at the bottom layer and only one sample at the top layer. A corresponding layer with the desired resolution can be selected for image rendering within reasonable amount of time.

From the color and opacity composition functions, we observe that there is no effect if a ray hits an empty voxel with zero opacity. Computation time can be saved if we can omit evaluating empty voxels. Another observation is that an object usually contains coherent regions of empty voxels. Several methods proposed to encode the coherence in the volume data include octree hierarchical spatial enumeration, polygonal representation of bounding surfaces, and octree representation of bounding surfaces. Levoy employs an octree enumeration represented by a pyramid of binary volumes.

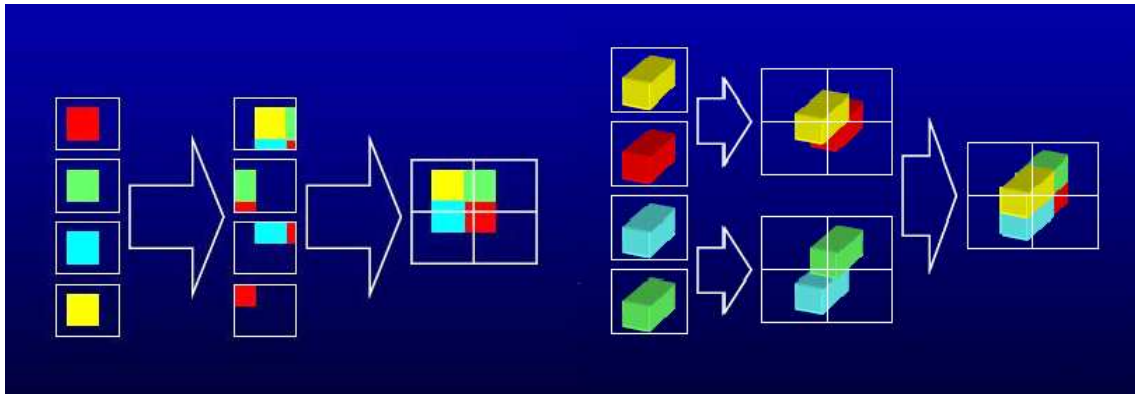


Figure 2.7: Sort-first(left) and sort-last(right) [31].

Confining data traversal to valid segments reduces unproductive processing costs. For example, polygon assisted ray casting, known as PARC [52], limits the ray within the data sample space. Another example is adaptive termination of the color and opacity composition in ray casting.

2.5 Parallel Rendering Techniques

Recent advances in scientific simulation and experimental technology has produced gargantuan far greater volume of datasets than has been previously possible. High dimensional, time-varying datasets of terabytes are fairly common today. A commensurate increase in the capabilities of data exploration, rendering, and display is in demand. The issue becomes even more critical when petabyte scale datasets appear. Generally speaking, the scalability problem emerges along two axes: size of datasets, and number of display pixel.

Although modern technology generates impressive performance in CPU and rendering cards, there are still some colossal datasets and super-complex geometries that are capable of eclipsing even the fastest of them. For example, a powerful rendering card that can render 20 million triangular facets per second can display geometries up to 330,000 triangular polygons, with an interactive frame rate of 20 per second. This may seem to be a

fairly large number of polygons for most visualization applications, however, some scientific visualization applications require much greater polygon counts. Moreover, scientists are now viewing their data via a big and wide-angle tiled display wall to achieve a seamless view. An enormous number of pixels would break down the best projector. Cluster graphic stations can utilize not only the aggregated computation power, but also the aggregated rendering power embedded in the collection of graphics accelerators. Depending on the situation, parallelism can be categorized into two methods:

2D screen space: It is also called a sort-first (tiled) rendering, targeting display scalability. In order to cope with the high frame rate, the frame buffer is subdivided into rectangular tiles, which may be rendered parallel by the M hosts of a rendering cluster. The primitives can be moved and duplicated for load balancing.

3D data space: It is also called sort-last (Z-compositing) rendering. A 3D dataset is broken down into N parts, which are rendered parallel by M processors. Those rendered images are composited according to Z buffer and alpha values for final image display.

Hybrid parallel rendering: Sort-first and sort-last rendering may be combined into a hybrid configuration [31].

Our system partitions the whole data volume into N blocks that will be evenly distributed to M computing nodes for geometry extraction and rendering. For parallel rendering, we have N machines, each with its own graphic card. Each card is responsible for the rendering of a geometry subset. An even work-load division strategy is preferred to split the polygon rendering task among all cards; however, it is not a trivial task to divide the geometry equally. Spatial coherence property is explored in our parallel system implementation for work-load balance. The sub-images and depth buffers from each rendering card are sent to a master node for final image composition. The merging procedure is quite straightforward, namely a pixel in the final image is the pixel in the sub-image with the smallest depth value. Image composition can also be done in a hierarchical or concurrent manner to reduce the data traffic and speed up the whole process. Binary-swap algorithm

[39], which requires 2^n nodes and progresses in n stages by splitting and exchanging partial image at each stage, is implemented in our system. At the end, a simple assembly process is needed to collect all sub-images on all nodes.

2.6 Remote Visualization System

In order to achieve interactive remote visualization for large datasets, a considerable amount of research effort have been made to either improve visualization algorithms for less data transmission, or design flexible remote visualization architectures that properly assign visualization modules across network nodes for optimal work-load balance. To our best knowledge, neither comprehensive research work nor system design has ever been conducted to address the optimal network performance aspect of a remote visualization system.

2.6.1 Efforts on Visualization Techniques

Image streaming is a simple remote visualization technique that allows a client with minimal rendering capability to explore remote datasets in an efficient way. Several trials have been conducted to achieve high throughput for streaming the images across the network. Engel *et al.* [37] applied image streaming techniques for web-based visualization: images are streamed across the network from a high-end graphics server to clients with a large variety of system architectures using Common Object Request Broker Architecture (CORBA). Ma and Camp [48] presented a solution that employs pipelining, efficient processor management, and compression to achieve high resource utilization and low data transfer costs when streaming images. Ding *et al.* [33] developed a remote visualization system to browse image-based databases using a light field rendering technique and Logistical Networking technology. In some remote visualization applications, data or geometries can also be sent to a client with sufficient computing and rendering capabilities. With data or geometries available locally, the client is given more freedom in manipulating viewing parameters; it may also reduce network traffic in the long run. Significant research efforts

have focused on minimizing the network transfer time to support interactive data exploration by, for example, reducing the size of data or geometries sent through the network and enforcing the client update from the server only when necessary. Most view dependent and isosurface extraction algorithms [43, 67] fall into this category. Recently, Neeman *et al.* [49] introduced a remote isosurface visualization technique that uses a chessboarded interval tree data structure to reduce the isosurface extraction cost and send compressed data in order to reduce network transport delays.

2.6.2 Efforts on Network Transport and System Architecture

Other related works include the optimization of network transport and system architecture. Brodlie *et al.* [35] explored the extension of existing dataflow visualization systems to Grid environments. Stegmaier *et al.* [57] presented a generic solution for hardware-accelerated remote visualization that is independent of application and architecture. Bethel *et al.* [62] designed a new architecture that utilizes high speed WANs and network data caches for data staging and transmission. The visualization process pipelining was supported by massively parallel hardware to achieve high data throughput and network utilization. Luke and Hansen [46] presented a flexible remote visualization framework capable of multiple partition scenarios, which is tested and evaluated on a local network. The framework supports multiple rendering methods, including image streaming, geometry rendering, and ZTex rendering. Bowman *et al.* [25] proposed a framework to predict the processing times of visualization modules using analytic models; these predictions can be used to obtain suitable mapping of the visualization pipeline. Bokhari [24] addressed the problem of optimally assigning the computing modules over processors based on a sum-bottleneck path algorithm. However, the selection of network nodes are predetermined for both chains to chains mapping and multiple chains to host-satellite mapping. We aim to drop this constraint by allowing arbitrary nodes selection.

However, little work has been done to address the global optimization of the decom-

position and mapping of a remote visualization pipeline under varying network conditions in a distributed computing environment. In addition, the existing approaches that map visualization modules to network nodes are mostly empirical and require manual configuration, and their implementations are typically limited to a traditional client and server mode without considering intermediate nodes.

Chapter 3

Analytical Model

We now describe an analytical model for the general problem of visualization pipeline decomposition and its network mapping. As shown in Fig. 3.1, the visualization pipeline consists of $n + 1$ sequential modules,

$$M_1, M_2, \dots, M_{u-1}, M_u, \dots, M_{v-1}, \dots, M_w, \dots, M_{x-1}, M_x, \dots, M_{n+1},$$

where M_1 is a data source. Module $M_j, j = 2, \dots, n + 1$ 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 $v_1, v_2, \dots, v_k, v_{k+1}$. Node $v_i, i = 1, 2, \dots, k, k + 1$ has a normalized computing power p_i ¹ and is connected to its neighbor node $v_j, j = 1, 2, \dots, k, k + 1, j \neq i$ 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

¹For 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.

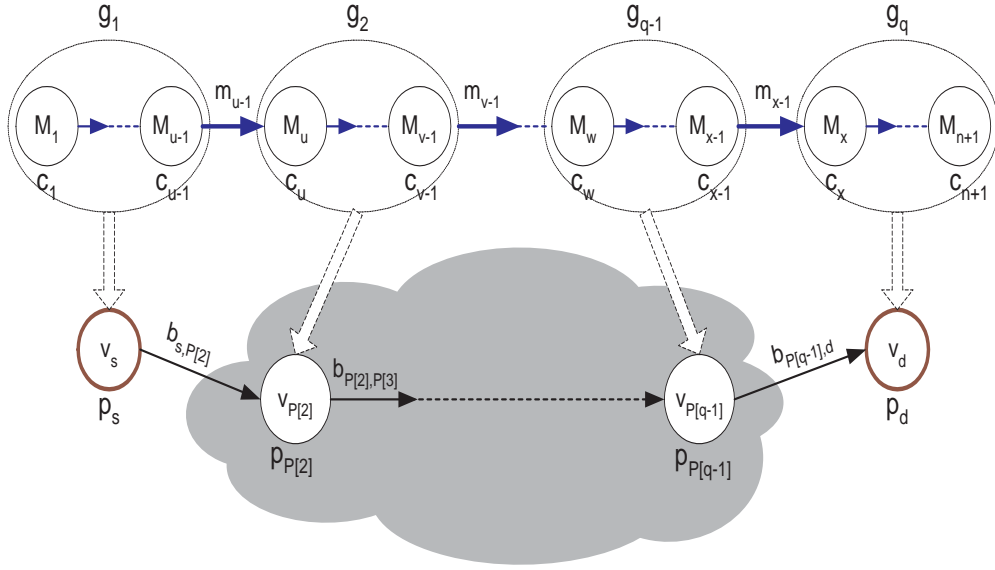


Figure 3.1: Mathematical model for pipeline partitioning and network mapping

$v_{P[1]} = v_s, v_{P[2]}, \dots, v_{P[q-1]}, v_{P[q]} = v_d$. The visualization pipeline is decomposed into q visualization groups denoted by $g_1, g_2, \dots, g_{q-1}, g_q$, which are mapped one-to-one onto the q nodes on transport path P . The data flow between two adjacent groups is the one produced by the last module in the upstream group; for example in Fig. 3.1, we have $m(g_1) = m_{u-1}, m(g_2) = m_{v-1}, \dots, m(g_{q-1}) = m_{x-1}$. 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.

3.1 Minimal Total Delay

$$\begin{aligned}
T_{total}(\text{Path } P \text{ of } q \text{ nodes}) &= T_{computing} + T_{transport} \\
&= \sum_{i=1}^q T_{g_i} + \sum_{i=1}^{q-1} T_{L_{P[i],P[i+1]}} \\
&= \sum_{i=1}^q \left(\frac{1}{p_{P[i]}} \sum_{j \in g_i, j \geq 2} (c_j m_{j-1}) \right) + \sum_{i=1}^{q-1} \left(\frac{m(g_i)}{b_{P[i],P[i+1]}} \right).
\end{aligned} \tag{3.1}$$

The most important requirement in many collaborative visualization applications is the interactivity of the system. For applications in which the remote visualization is employed, we consider the optimization problem minimizing the total end-to-end delay:

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

3.2 Maximal Frame Rate

Our goal is to maximize the frame rate by minimizing the time incurred on a bottleneck link/node for applications with streaming data, given as in Eq.3.2.

$$\begin{aligned}
T_{bottleneck}(\text{Path } P \text{ of } q \text{ nodes}) &= \max_{\substack{\text{Path } P \text{ of } q \text{ nodes} \\ i=1,2,\dots,q-1}} \left\{ \begin{array}{l} T_{computing}(g_i), \\ T_{transport}(L_{P[i],P[i+1]}), \\ T_{computing}(g_q) \end{array} \right\} \\
&= \max_{\substack{\text{Path } P \text{ of } q \text{ nodes} \\ i=1,2,\dots,q-1}} \left\{ \begin{array}{l} \frac{1}{P_{P[i]}} \sum_{j \in g_i \text{ and } j \geq 2} (c_j m_{j-1}), \\ \frac{m(g_i)}{b_{P[i],P[i+1]}}, \\ \frac{1}{P_{P[q]}} \sum_{j \in g_q \text{ and } j \geq 2} (c_j m_{j-1}) \end{array} \right\} \quad (3.2)
\end{aligned}$$

Chapter 4

Remote Computational Monitoring and Steering

4.1 Introduction

Large, computation-intensive simulation tasks for complex physical phenomena are usually carried out in batch mode. Several files containing the initial conditions and parameters together with simulation programs are submitted to a batch queue. The simulation program get executed from the beginning to the end according to the initial input files. Results saved in the hard disk can only be accessed and processed after entire computation is finished. This approach could result in resource-inefficiency if no human intervention in the loop is allowed during the time-consuming evolution process.

Technology advances in computer processing power and network bandwidth has significantly changed the nature of simulation process. Interactive manipulation of the simulation code become possible with the help of visual output using standard visualization techniques. User can navigate and explore the result in real time. Enormous Computation cycles and time can be saved if bad jobs can be terminated or stray simulation parameters can be corrected in time. The ability to visualize computational output and modify the computational parameters while simulation is running is called computational steering. Fig. 4.1 demonstrates the schematic architecture of a simple computational steering job. We aim to design a steering system to achieve the maximal possible frame rate by efficiently allocating networking resources.

4.2 Terascale Supernova Initiative Project

Terascale supernova initiative(TSI)project is a multidisciplinary collaboration efforts, which involve researcher from one national laboratory(ORNL) and eight other Universities across this country. People in TSI project are in the search for the explosion

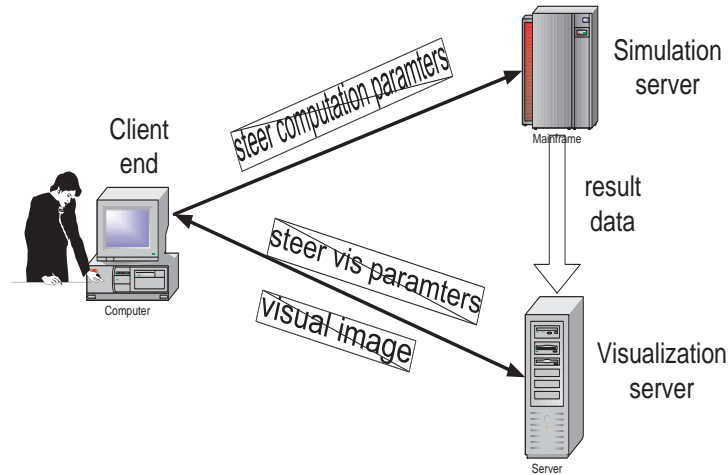


Figure 4.1: Schematic architecture of a simple computational steering application

mechanism of core collapse supernovae, the most energetic explosion in the universe[19], and the computation of the nucleosynthesis.

Experts in hydrodynamics, fusion energy, and high energy physics collaborate on the mathematical modeling and computation of supernovae phenomena. Astrophysics group at NCSU simulate the core collapse supernovae using Virginia Hydrodynamics(VH-1) code, which was written by the numerical astrophysics group at the University of Virginia in early 90s based on piece-wise parabolic method (PPM). A large portion of code writing and testing was done by John Hawley, John Blondin, Greg Lindahl, and Eric Lufkin. The normal data file size reaches up to 2-3Gb per time step and the amount of daily-generated data exceeds terabyte currently.

After the simulation process gets started, several files storing the initial condition and parameters will be read. A separate data file is written out when the calculation for certain cycles is finished. A binary dump file describing the history information is written periodically to allow a restart if anything goes wrong in the middle [21]. Both serial and MPI version of VH-1 code is available for deployment on desktop and cluster machine.

Since research personnel and computing resources are distributed across the coun-



Figure 4.2: Crayx1 supercomputer at ORNL

try, remote collaborative efforts are frequently practised for TSI researchers. For example, simulation process running on local supercomputer will be monitored and manipulated by remote users.

4.3 Imminent Challenges and Strategic Solutions

An efficient remote computational steering system couples many areas including advanced visualization, virtual environments, high performance computing and high speed communications to achieve maximal frame rate. Data is continuously propagated from the simulation server to visualization server and other computing devices for computation, rendering and transport.

Our main challenge comes from the unwieldy big data size. A TSI simulation process may takes about 8 hours to run and produce one terabyte of data. The ability to generate, process and transport such large amount of data is essential for efficient computational steering.

A 256-processor next-generation supercomputer Cray X1 sponsored by DOE is housed in Center for Computational Sciences at ORNL to evaluate the processors, memory subsystem, scalability of the new architecture. The micro-benchmarks and kernel benchmarks architecture feature of Cray X1 demonstrates exceptionally fast performance especially for large scale problem where the entire problem cannot be loaded into cache. Large

scale problems are the most demanding classes of scientific and engineering applications that are vital for ultra-scale simulation. The Cray X1 is the first U.S. computer to have both vector processing and massively parallel processing capabilities in a single architecture. A follow-up Cray system is expected to reach a speed of 100-teraflop (trillions of calculations per second) in 2006, with the potential to grow to 250 teraflops in 2007. TSI code is now being deployed in Cray X1 for daily simulation jobs.

Supercomputers where raw data is initially created usually are not equipped with accelerated graphic card for geometry rendering. Intermediate data has to be shipped to a nearby visualization cluster to create 2D image. In our system, a MPI-based visualization module is deployed on a visualization cluster at University of Tennessee at Knoxville to render data generated by Cray X1 at ORNL.

The ability to continuously ship huge amount of data out of ORNL in a timely manner poses another challenge. To enhance the connectivity between CCS and research community across the country, the center is connected to Internet2 via a new OC192 (10 Gbit/s) connection. Even with the high speed physical transport media, efficient transport protocols which support refined flow control and retransmission mechanisms are still indispensable to fully utilize the underlying physical media. For example, the data generated in 8 hours at CCS may take up to 14 hours to be transferred to NCSU using traditional TCP channel which is intolerable. New transport protocols have been developed to overcome the limitations of traditional TCP and UDP in terms of throughput, stability and dynamics [64, 65]. The highest throughput of the new class of transport protocols is about 960Mbps for a 1Gbps link. We intend to replace our current TCP with new reliable UDP-based transport protocols to achieve throughput maximization and throughput stability.

The second challenge stems from implementation concern. Most simulation code including VH-1 is written in Fortran language, a popular programming language for scientific computation. However, our visualization module and network transport module are

written in C++ and java. To integrate modules in different languages, a nice style of mixing programming is required. Namely, modules need to be as independent as possible for future code reuse and maintenance. However, interaction and message passing between modules should be as intimate as possible for seamless cooperation.

In order to achieve above goals and construct a universal computational steering framework, several generic viz/network function calls are provided and wrapped into a library. Function calls from this library are inserted at strategic points in the VH-1 code to deliver results or intercept steering commands from the client. Communication between simulation node and visualization node is carried out by socket for data and control message transfer. Such implementation design promote modular programming and code portability.

Chapter 5

Remote Wireless Sensor Network Monitoring

5.1 Introduction

Research on WSN began in the early 1980's for US defense establishment. Technology advances have enabled many wireless sensor network (WSN) based civil and military applications, such as environmental monitoring, event detection, target tracking, etc. A large scale WSN, comprised of thousands or even millions of unattended tiny sensing devices may be deployed in a human-unfriendly environments such as a battlefield, a rain forest, and bottom of the ocean. Users normally sit at a remote safe end and operate a command control and visualization system via a wide area network connection in real time.

Most WSN visualization systems make use of only simple visualization techniques for graphical sensor data representation, such as line chart and simple glyph display with a light computation load. However, if the number of sensor nodes increases considerably, direct display would create indistinguishable cluttering. Consequently, large scale WSN visualization applications call for advanced scientific visualization techniques, such as iso-surface extraction, streamlines, and volume rendering to extract hidden information from raw data and present informative graphical representations. On the other hand, some sensor data even with tractable size must undergo intensive computation cycles to establish an accurate prediction model for example. Such massive computation load will overwhelm most computer nodes and has to be carried out by some powerful computing devices accessible on the network.

The complicated computing and visualization pipeline made up of many computing components with various complexities and hardware requirements necessitate a smart pipeline handling mechanism for optimized network performance.

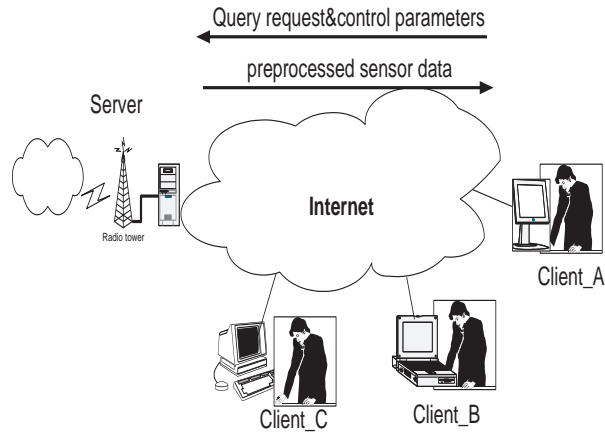


Figure 5.1: Framework of remote visualization system for WSN

In this Chapter, we proposed several critical techniques related to WSN operations, namely the data buffer and dispatch protocols for periodic monitoring and a decision fusion algorithm for event detection. All these technologies are deployed at the data collection end, and network-based system optimization starts from data collection afterwards.

5.2 Periodic Environment Monitor

Sensor nodes measure environmental variables at certain intervals and route data back to the base station. In our system, all sensor data registered to the same time stamp will be temporarily buffered in an individual file for later dispatch when sufficient amount of data has arrived.

5.2.1 Sensor Data Buffer and Dispatch Protocol

A debate over the selection of buffer time encourages an investigation into the overall data collection mechanism to achieve a satisfactory tradeoff between data integrity and data promptness. The ad hoc nature of WSN suggest an elastic and fault tolerant mechanism since network topology and connectivity is at constant change.

We present two heuristic methods for sensor data buffer and dispatch, namely relative based approach and timeout based approach. Information of sensor ID, time step and

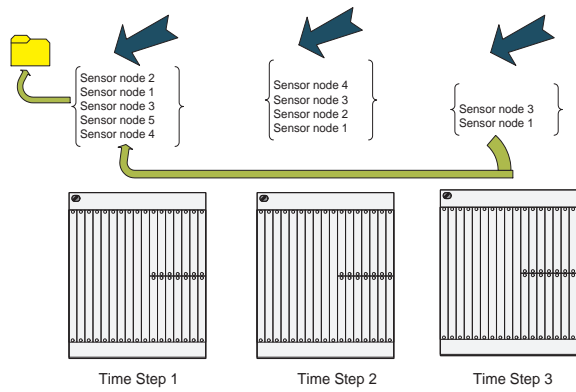


Figure 5.2: Relative data buffer model

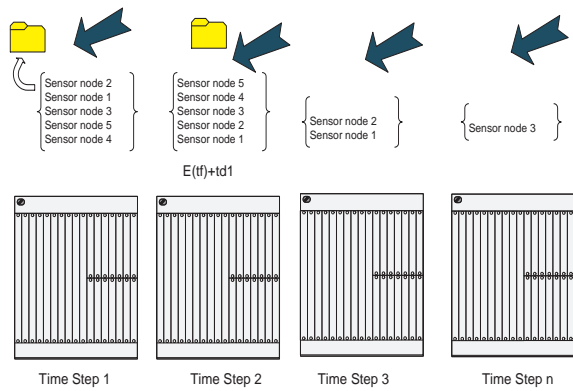


Figure 5.3: Timeout data buffer model

measurement are all included within each data packet. To evaluate the performance of each data collection method, the following variables are used: T defines the sampling interval; t_{if} specifies the arrival time for the first packet at time step i ; t_{il} represents the arrival time for the last packet at the time step i ; n_i gives the total packets number at time step i ; We aim to minimize the objective function evaluating both the quality of average waiting time and the total number of retrieved sensor packets.

$$\begin{aligned} C_i &= C_{WaitingTime} + C_{DataIntegrity} \\ &= \frac{w_t(t_{il}-t_{if})}{n_i} + w_g|n_{i-1} - n_i| \end{aligned} \quad (5.1)$$

wherein w_t and w_g represent the weighted factors for data promptness and data integrity respectively.

5.2.1.1 Relative Approach

Instead of exploiting the usage of timeout for data dispatch, a simple relative data preemption method is adopted. Namely, each bucket is responsible for data from a specific time step, and there are always fixed number of buckets. Whenever there are enough packets from the last bucket, all packets from first bucket will be pushed out for dispatch. The empty bucket will be moved to the tail to buffer the next recent packets. The definition of “enough” is dynamic and is referred from history record on last dispatch file. Namely, higher the number of total packets from last dispatch file, higher is set for “enough” bar. Any data arriving after its destination bucket is emptied will be discarded as late punishment.

There are several merits by using such approach: Firstly, it avoids manual setting for a good timeout for a volatile WSN, and such method adaptively gears to the current WSN size without human administration. Secondly, the simplicity of such method favors energy conservation to prolong system life. For sensor network depending on limited battery and computing devices, running complicated protocols would quickly deplete resources.

In spite of these advantages, scenario does exist that would compromise the per-

formance of this approach. For example, if “enough” sensor nodes are in close vicinity to base station, nodes that further away from the base station will be discarded. Moreover, high sampling frequency will further aggravate this situation.

5.2.1.2 Timeout Approach

A dynamic timeout for each time step is dynamically defined for this method. After system initialization, timeout for the next time stamp i is recursively calculated based on knowledge from previous time step. It is chosen as: $Max(t_{if} + (t_{(i-1)l} - t_{(i-1)f}), t_{if} + T)$.

The number of initially deployed nodes is usually known after system initialization. For the first time step, base station waits for sufficient amount of time to catch most data.

The arrival time for the far-est nodes is implicitly utilized for timeout selection. Problem as described in the relative method no longer remains a problem here. However, this method assumes that the distance from far-est node to base station is comparably stable. If such constraint is violated, nodes lying out of this virtual circumference centered at the base station will be lost. To get around this problem, we can pose an additional constraint by withholding the data until “enough” data arrive.

5.3 Event Detection

Since sensor failures are common, redundant nodes are usually necessary to ensure uninterrupted and reliable operations. Measurements or decisions from individual sensors are sent to a fusion center over wireless channels to achieve a more accurate situation assessment.

5.3.1 Threshold-OR Fusion Method

We consider a network of sensors that are distributed in a three dimensional environment to monitor a region of interest for possible intrusion of a target. The sensors collect environmental measurements that are subject to independent additive noise. Each sensor node employs a threshold rule on the measurements to detect a target in the presence of random background noise. The threshold value employed by sensor nodes and

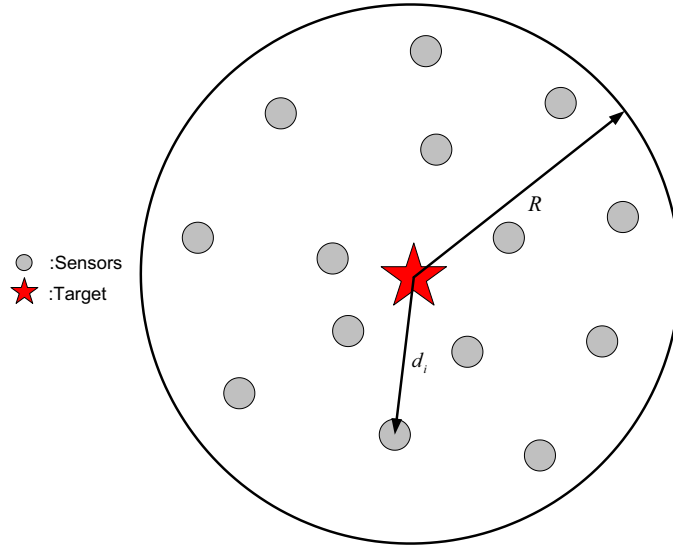


Figure 5.4: Sensor deployment for target detection.

their distances to the target determine both their hit and false alarm probabilities or rates using a signal attenuation model. When a target enters the monitoring region, it could be detected by multiple sensor nodes depending on their distances from the target. Typically, nearby sensors produce larger measurements while distant sensors receive less quantity of the measured signal. Our objective is to combine the decisions made by individual sensor nodes to achieve system detection performance beyond weighted average of individual sensor nodes.

At the core, this problem is a specific instance of the classical distributed detection problem that has been studied extensively over past several decades [61]. The problem of fusing binary decisions taken at various sensor nodes has been solved using various schemes such as logical AND, OR, voting, Neyman-Pearson, and Bayes rule [23, 30, 55, 56, 58]. Logical AND, OR, and voting techniques are examples of simple fusion schemes that do not require sensor probability distribution functions. Voting schemes can be further classified into threshold schemes and plurality schemes. Unanimity voting, majority voting,

and m -out-of- n voting are threshold schemes, which try to minimize the probability of wrong decisions. Plurality voting designates the hypothesis that receives the most votes to be the best possible decision. Although it involves only inexpensive computations and provides some degree of fault tolerance, a simple voting scheme generally lacks necessary performance guarantees in terms of error rates.

Chair and Varshney [26] proposed an optimal fusion rule based on Bayes rule, requiring *a priori* probabilities. However, in practice, these *a priori* probabilities are often unavailable. Niu *et. al.* [53] presented a fusion method that chooses an optimal sensor threshold to achieve the maximal system hit rate given a certain system false alarm rate. This method demands a very large number of sensor nodes to apply the Central Limit Theorem. Unfortunately, this criterion cannot be always satisfied in real scenarios. Neyman-Pearson decision rule provides an optimal solution minimizing Type II error when subject to a chosen upper bound on Type I error probability or vice versa based on the likelihood ratio test. Although no *a priori* probability is required, Neyman-Pearson fusion rule requires sensor readings and continuous sensor probability distribution functions to apply the notion of confidence level, and these information may incur high computational cost or even not be available.

Other researchers focused on correlated sensor data fusion. Chen and Ansari [27] developed an adaptive fusion algorithm to estimate *a priori* and conditional probability through reinforcement learning. Although their method does not rely on *a priori* probability, it requires certain number of iterations for system convergence. In addition, the weight rate is updated using an approximation method that ignores certain variables dependence in partial derivative calculations. Rao [54] developed fusion methods that do not require the knowledge of sensor probability distribution functions but need to be trained using measurements. Such training measurements may not be available in some cases, and in others too many measurements may be needed to ensure reasonable levels of performance

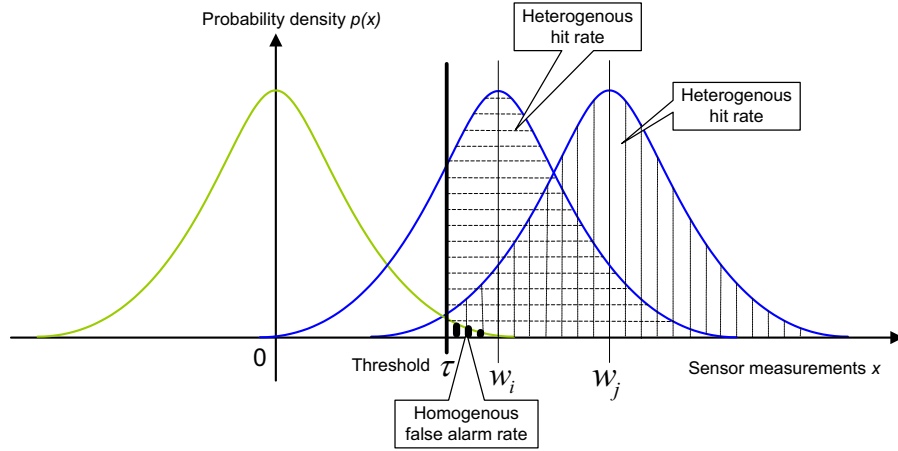


Figure 5.5: Normal distribution based hit rate and false alarm rate calculation for heterogeneous system

guarantees.

We propose a centralized threshold-OR fusion rule for combining the individual sensor node decisions in the sensor network described above. We derive threshold bounds for accumulated decisions using Chebyshev’s inequality based on individual hit and false alarm probabilities but without requiring an *a priori* knowledge of the underlying probability densities. We mathematically ensure that the fused method achieves a higher hit rate and lower false alarm rate compared to the weighted averages of individual sensor nodes. Restraining conditions are also derived. Simulation results using Monte Carlo method show that the error probabilities in the fused system are significantly reduced and approach zero.

5.3.1.1 Problem Formulation

We consider N sensor nodes deployed in a three dimensional region of interest (ROI), centering around a target within radius R , as shown in Fig. 5.4.

$$n_i \sim \mathcal{N}(0, 1). \quad (5.2)$$

At sensor i , the noise n_i in a sensor measurement is independently and identically distributed (iid) according to the normal distribution. The sensor measurements are subjective

to additive term due to such noise. Each sensor i makes a binary local decision as:

$$\begin{aligned} H_1 : & \quad r_i = w_i + n_i \\ H_0 : & \quad r_i = n_i \end{aligned} \quad (5.3)$$

where r_i is the actual sensor reading, w_i is the ideal sensor measurement, and n_i is the noise term.

We consider an isotropic signal attenuation power model defined by:

$$w_i = \frac{w_0}{\sqrt{1 + \beta d_i^n}}, \quad (5.4)$$

where w_0 is the original signal power emitted from the target located at point (x_0, y_0, z_0) , β is a system constant, and d_i represents the Cartesian distance between the target and the sensor node, which is defined in Eq. 5.5. Parameter n is the signal attenuation exponent typically ranging from 2 to 3.

$$d_i = \sqrt{(x_i - x_0)^2 + (y_i - y_0)^2 + (z_i - z_0)^2}. \quad (5.5)$$

This model describes a three-dimensional unobstructed region monitored by a set of sensors that detect the signal emitted from a target within the monitoring area.

Suppose that every sensor node retains the same threshold τ for decision making regardless of its distance to the target. The expected signal strength w_i for sensor i can be computed from Eq. 5.4 according to its distance to the target. Thus, the hit rate p_{h_i} and false alarm rate p_{f_i} for sensor i can be derived as follows [53]:

$$p_{h_i} = \int_{\tau}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-w_i)^2}{2}} dx, \quad (5.6)$$

$$p_{f_i} = \int_{\tau}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} dx. \quad (5.7)$$

Fig. 5.5 illustrates the use of a normal distribution function for computing individual sensor hit rate and false alarm rate in a heterogenous system. A simplified homogeneous system ignores the impact of various distances to the target on sensor detection capabilities

so that every sensor has the same hit rate and false alarm rate. The heterogenous sensor network system considers different hit rates and the same false alarm rates as computed by Eq. 5.6 and Eq. 5.7, respectively.

5.3.1.2 Threshold Bounds Derivation

Each sensor i makes an independent binary decision S_i as either 0 or 1. The fusion center collects local decisions and computes S as a binomial distribution:

$$S = \sum_{i=1}^N S_i, \quad (5.8)$$

which is then compared with a threshold T to make a final decision. The mean and variance of S are given as follows when a target is present:

$$\begin{aligned} E(S|H_1) &= \sum_{i=1}^N p_{h_i}, \\ \text{Var}(S|H_1) &= \sum_{i=1}^N p_{h_i}(1 - p_{h_i}). \end{aligned} \quad (5.9)$$

Similarly, the mean and variance of S when a target is absent are defined as:

$$\begin{aligned} E(S|H_0) &= \sum_{i=1}^N p_{f_i}, \\ \text{Var}(S|H_0) &= \sum_{i=1}^N p_{f_i}(1 - p_{f_i}). \end{aligned} \quad (5.10)$$

The threshold value T is critical to the system performance. Let P_h and P_f denote the hit rate and false alarm rate of the fused system. If we let $P_h > 0.5$ and $P_f < 0.5$, T should be bounded by $\sum_{i=1}^N p_{f_i} < T < \sum_{i=1}^N p_{h_i}$.

The weighted averages of p_{h_i} and p_{f_i} , $i = 1, 2, \dots, N$ are defined as follows:

$$\sum_{i=1}^N \frac{p_{h_i}}{\sum_{j=1}^N p_{h_j}} p_{h_i} = \frac{\sum_{i=1}^N p_{h_i}^2}{\sum_{i=1}^N p_{h_i}}, \quad (5.11)$$

$$\sum_{i=1}^N \frac{1 - p_{f_i}}{\sum_{j=1}^N (1 - p_{f_j})} p_{f_i} = \frac{\sum_{i=1}^N (1 - p_{f_i}) p_{f_i}}{\sum_{i=1}^N (1 - p_{f_i})}. \quad (5.12)$$

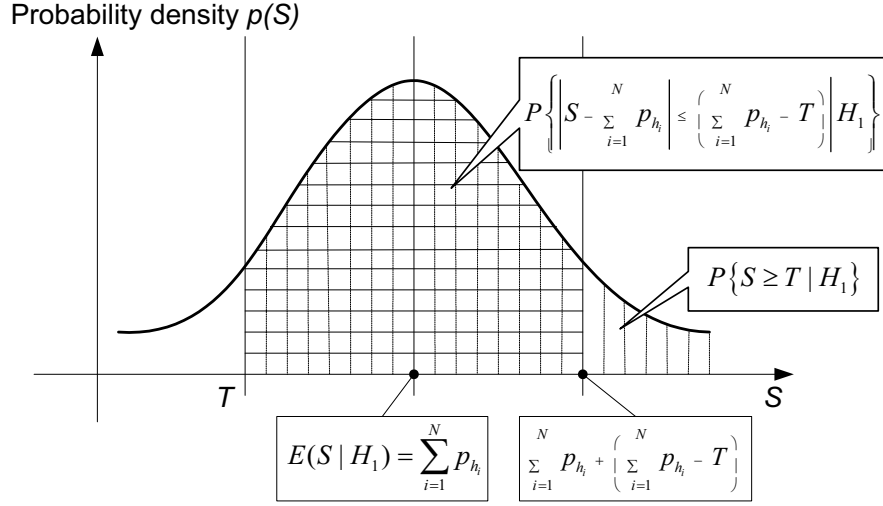


Figure 5.6: Application of Chebyshev's inequality in calculating the lower bound of the system hit rate.

We desire better detection performance of the fused system than the corresponding weighted averages in terms of higher hit rate and lower false alarm rate such that:

$$P_h > \frac{\sum_{i=1}^N p_{h_i}^2}{\sum_{i=1}^N p_{h_i}}, \quad (5.13)$$

$$P_f < \frac{\sum_{i=1}^N (1 - p_{f_i}) p_{f_i}}{\sum_{i=1}^N (1 - p_{f_i})}. \quad (5.14)$$

We first consider a lower bound of the hit rate of the fused system :

$$\begin{aligned} P_h &= P\{S \geq T | H_1\} \\ &\geq P\{|S - \sum_{i=1}^N p_{h_i}| \leq (\sum_{i=1}^N p_{h_i} - T) | H_1\} \\ &\geq 1 - \frac{\sum_{i=1}^N p_{h_i} (1 - p_{h_i})}{(\sum_{i=1}^N p_{h_i} - T)^2}, \end{aligned} \quad (5.15)$$

where we applied Chebyshev's inequality in the second step as shown in Fig. 5.6.

Now the condition Eq. 5.13 can be ensured by the following sufficient condition.

$$1 - \frac{\sum_{i=1}^N p_{h_i}(1 - p_{h_i})}{(\sum_{i=1}^N p_{h_i} - T)^2} \geq \frac{\sum_{i=1}^N p_{h_i}^2}{\sum_{i=1}^N p_{h_i}}. \quad (5.16)$$

Following that, an upper bound of T can be derived from Eq. 5.16 as follows:

$$T \leq \sum_{i=1}^N p_{h_i} - \sqrt{\sum_{i=1}^N p_{h_i}}. \quad (5.17)$$

Similarly, for the false alarm rate, we carry out a similar procedure to compute the lower bound from Eq. 5.18 to Eq. 5.22. Again, Chebyshev inequality is applied in the second step in Eq. 5.20.

$$P_f = P\{S \geq T | H_0\} = 1 - P\{S < T | H_0\}, \quad (5.18)$$

$$P\{S < T | H_0\} \geq P\{|S - \sum_{i=1}^N p_{f_i}| \leq (T - \sum_{i=1}^N p_{f_i}) | H_0\}, \quad (5.19)$$

$$P_f \leq 1 - P\{|S - \sum_{i=1}^N p_{f_i}| \leq (T - \sum_{i=1}^N p_{f_i}) | H_0\} \leq \frac{\sum_{i=1}^N p_{f_i}(1 - p_{f_i})}{(T - \sum_{i=1}^N p_{f_i})^2}, \quad (5.20)$$

$$\frac{\sum_{i=1}^N p_{f_i}(1 - p_{f_i})}{(T - \sum_{i=1}^N p_{f_i})^2} \leq \frac{\sum_{i=1}^N (1 - p_{f_i})p_{f_i}}{\sum_{i=1}^N (1 - p_{f_i})}, \quad (5.21)$$

$$T \geq \sum_{i=1}^N p_{f_i} + \sqrt{\sum_{i=1}^N (1 - p_{f_i})}. \quad (5.22)$$

Therefore, we define the range of T using the upper bound in Eq. 5.17 and lower bound in Eq. 5.22 as follows:

$$\left[\sum_{i=1}^N p_{f_i} + \sqrt{\sum_{i=1}^N (1 - p_{f_i})}, \sum_{i=1}^N p_{h_i} - \sqrt{\sum_{i=1}^N p_{h_i}} \right]. \quad (5.23)$$

For the case of a homogeneous system, the bounds of threshold T are simplified as follows:

$$\left[Np_f + \sqrt{N(1-p_f)}, Np_h - \sqrt{Np_h} \right]. \quad (5.24)$$

To ensure that the upper bound is larger than the lower bound, we have the following restrictions on individual hit rates, individual false alarm rates, and the number of sensor nodes, for the heterogeneous and homogeneous systems, respectively :

$$\sum_{i=1}^N p_{f_i} + \sqrt{\sum_{i=1}^N (1-p_{f_i})} - \sum_{i=1}^N p_{h_i} + \sqrt{\sum_{i=1}^N p_{h_i}} \leq 0, \quad (5.25)$$

$$p_h - p_f \geq \frac{\sqrt{p_h} + \sqrt{1-p_f}}{\sqrt{N}}. \quad (5.26)$$

Note that for the homogeneous system, the number of sensor nodes N can be set large enough to satisfy Eq. 5.26.

5.3.1.3 Simulation and Results Discussion

Our simulation based on 100,000 runs¹ produced the receiver operative characteristic (ROC) curve, a plot of the system hit rate against the false alarm rate for different possible thresholds. There is a tradeoff between sensitivity and specificity, namely any increase in sensitivity of hit rate will be accompanied by an increase in non-specificity of false alarm rate. The curvature of the ROC curve determines the detection accuracy of the system, namely, the closer does the curve follow the left-hand border and then the top border of the ROC space, the more accurate is the system. The closer does the curve approach the 45-degree diagonal of the ROC space, the less accurate is the test.

For the homogeneous case, we set sensor hit rate to be 0.65, and sensor false alarm rate to be 0.2. In Fig. 5.7, four system ROC curves with sensor node number going from 15 to 25 are plotted. The desirable segment on the ROC curve is located by restricting

¹To simplify calculation, we restrict the sensor deployment in a 2-D plane by setting coordinate z to be zero. The simulation results for a 3-D region is qualitatively similar to the 2-D case.

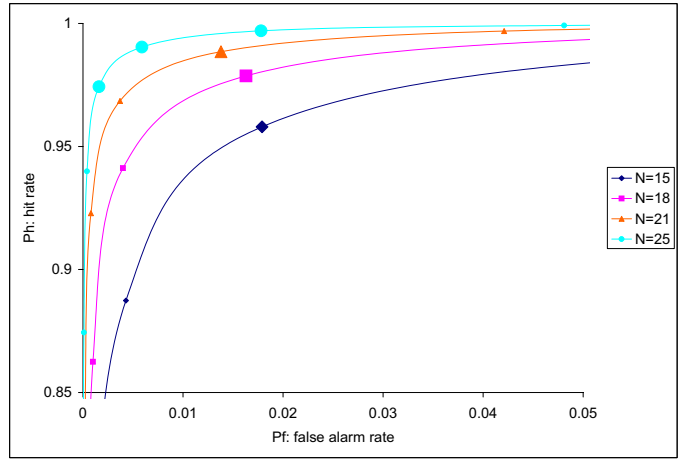


Figure 5.7: Homogeneous system ROC curve with different sensor node number

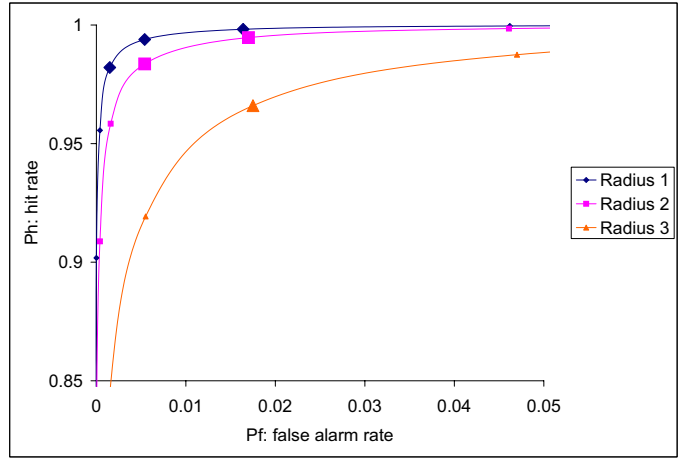


Figure 5.8: Heterogenous system ROC curve with different deployment radius

Table 5.1: Numeric results with different deployment radius for heterogenous system.

$N = 25, W_{p_f} = 0.2$	T_l	T_u	W_{p_h}	P_{h_l}	P_{h_u}	P_{f_l}	P_{f_u}
R=1	10	12	.6788	.9821	.9982	.0015	.0164
R=2	10	11	.628	.9836	.9947	.0054	.017
R=3	10	10	.5612	.966	.966	.0175	.0175

thresholds selection. Due to the discrete nature of our binary decision system, we identify the selected ROC segment as individual enlarged markers. From Fig. 5.7, we observe that all selected points fall at the top left corner of the ROC space, and bear hit rate and false alarm rate that is by far superior to that of a single sensor node. When the number of sensor nodes increases, our system performances are further enhanced due to ample available resources.

For the heterogeneous case, from Eq. 5.6 and Eq. 5.7, we conclude that sensors have the same false alarm rate and different hit rates due to various distances to the target. In our simulation program, the given hit rate and false alarm rate for sensors in close proximity to the target are represented as p_h and p_f , respectively. Sensor threshold τ can be calculated from Eq. 5.7 given a known false alarm rate p_f . Consequently, original signal power w_0 can be computed from Eq. 5.6. w_i can be derived from Eq. 5.4 and therefore, p_{h_i} can be computed according to Eq. 5.6. Our approach is also capable of handling systems with heterogenous false alarm rates if necessary.

We set the total number of sensor nodes to be 25. The p_h is set to be 0.75 and p_f is set to be 0.2 in simulation tests. Sensor deployment radius ranges from 1 to 3 to compare different deployment strategies. Results are tabulated in Table 1 as system threshold bounds(T_l, T_u), weighted average hit rate and false alarm rate as W_{p_h} , and W_{p_f} respectively, system hit rate bounds(P_{h_l}, P_{h_u}), and system false alarm rate bounds(P_{f_l}, P_{f_u}) under different radius R . Dispersed sensor deployment negatively affects the system performance as a result of depressed detection performance and truncated scope of threshold bounds. It further validates the rule that sensors should be deployed as close to potential targets as

Table 5.2: Comparison of different decision rules based fusion.

Rules	Fusion type	Hypothesis	pdf	<i>a priori</i> probability	Performance
Voting	Hard	Multiple	No	No	No guarantee
Bayes Criterion	Soft/Hard	Multiple	Yes	Yes	Minimize cost function
MAP	Soft/Hard	Multiple	Yes	Yes	Minimize error probability
Neyman-Pearson	Soft	Binary	Yes	No	Fix one error prob. minimize the other
Proposed method	Hard	Binary	Yes	No	Significantly reduce error prob.

possible. From Fig. 5.8 with radius 3, our fusion system achieves excellent hit rate close to 0.97 and low false alarm rate below 0.02 as compared to weighted average of 0.5612 and 0.2, respectively, before fusion.

5.3.1.4 Conclusions and Future Work

We proposed a threshold-OR fusion method for a binary decision-based sensor fusion system. Current non-model and model-based fusion methodologies are derived upon some variants of decision rules such as Voting, Bayes Criterion, Maximum a Posterior Criterion(MAP), and Neyman-Pearson. Comparison of fusion type in the aspect of input sensor data, hypothesis, sensor probability density functions(pdf), *a priori* probability, and performance is provided in Table 2. Despite the simplicity, non-model based voting rule gives no system performance guarantee in terms of probability of errors. Bayes criterion minimizes the cost function based on the knowledge on *a priori* probability which is not always available. MAP, a similar decision rule to Bayes Criterion, simply minimizes the overall probability of errors since it seeks to maximize the posterior probability. However, *a priori* probability is still desired. Neyman-Pearson(NP) is attractive since it does not require knowledge of priors and a cost function as opposed to Bayes criterion [61]. Nevertheless, NP based fusion rule is essentially a type of soft fusion, which demands

continuous local sensor readings and pdfs to control one error probability under certain confidence level.

Our method falls into the category of hard fusion accepting discrete sensor decisions, and does not require the knowledge on *a priori* probability. Since our conditions are sufficient and not necessary when derive the threshold bounds, simulation results under such restrictive conditions yield significantly reduced probability of errors compared to that of single sensor or weighted average with both Type I and Type II error approaching zero.

The given threshold bounds allow users certain freedom in shifting between sensitivity and specificity instead of a single cutting point. For example, if Type I error is not tolerable, user can choose a threshold close to the upper bound. If Type II error needs to be minimized, user may tend to pick a threshold close to the lower bound. In addition, our approach has a low computational cost making practical deployment feasible with limited resource. For future work, we plan to apply weighted factors to local decisions based on various distances and signal noise ratios (SNR) when calculating binomial distribution in Eq. 5.8. Deployment of the proposed fusion rule in practical sensor network application is also of our future interest.

Chapter 6

Technical Solutions for Network Optimization

We present in this section the technical solutions for computing time estimation for various visualization techniques, network bandwidth measurement using active measurements, and system configuration optimization for minimum total delay and maximal frame-rate based on dynamic programming and an accurate estimation on global information.

6.1 Computing Power Measurement for Processing Time Estimation

Whether in parallel implementation or not, visualization software has traditionally employed coarse-grained mechanisms based on volume blocks in a wide-spread manner; because block-based processing allows great convenience, flexibility, and scalability, especially when handling large-scale datasets. The methods we develop to estimate processing times are designed for block-by-block type of procedures, including extracting isosurfaces, rendering volumes, and visualizing flow fields with streamlines. For practical use, we would desire our relative estimation errors to be consistently less than 5.0%, a conventional statistical level.

In general, we face two basic technical issues: (i) the accumulated variance by measuring multiple individual blocks and (ii) those performance variations caused by different user-specified parameters, such as iso-values in isosurface extraction routines and view angles in volume renderers. The first issue can be controlled by reducing variance for each block as much as possible. The second issue requires us to design cost models that can reliably predict running time required by each specific algorithm, while only requiring a minimal amount of meta data for each block. We describe our findings to address both

issues in the following sections, respectively.

6.1.1 Addressing Accumulated Variance

An arbitrary volume is divided into multiple blocks for processing time estimation. Herein, we model the overall computing time incurred by a whole volume, T_v , as:

$$T_v = \sum_{i=1}^N T_{b_i} \quad (6.1)$$

where, T_{b_i} is the processing time for the block, b_i , as estimated from a lookup table with the number of non-empty cells as table entry. The running time T_{b_i} is drawn from distribution: $\{\mu_i, \sigma_i^2\}$. With N independent blocks, by addition property of independent variables, the overall variance of T_v is: $\sum_1^N \sigma_i^2$.

We want to control the overall variance in regardless of the block number N . In sum, from the above observation, we understand that as long as the individual variance for each block can be significantly depressed at large sampling size, the coarse-grained mechanism does not pay a penalty of higher deviation. Having addressed the accumulated variance in the system, we describe below our algorithm-specific models developed for two popular visualization techniques: marching cube and ray-casting.

6.1.2 Marching Cubes

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

However, we found out that it is feasible to make such estimation with a small amount of additional meta data. Obviously, the processing time of a volume cell should highly depend on the number of extracted triangles. While a cell can generate triangles in different ways, previous researchers have identified the topologically unique cases out of all possibilities [45]. We note that at most a cell can generate four triangles. We designed

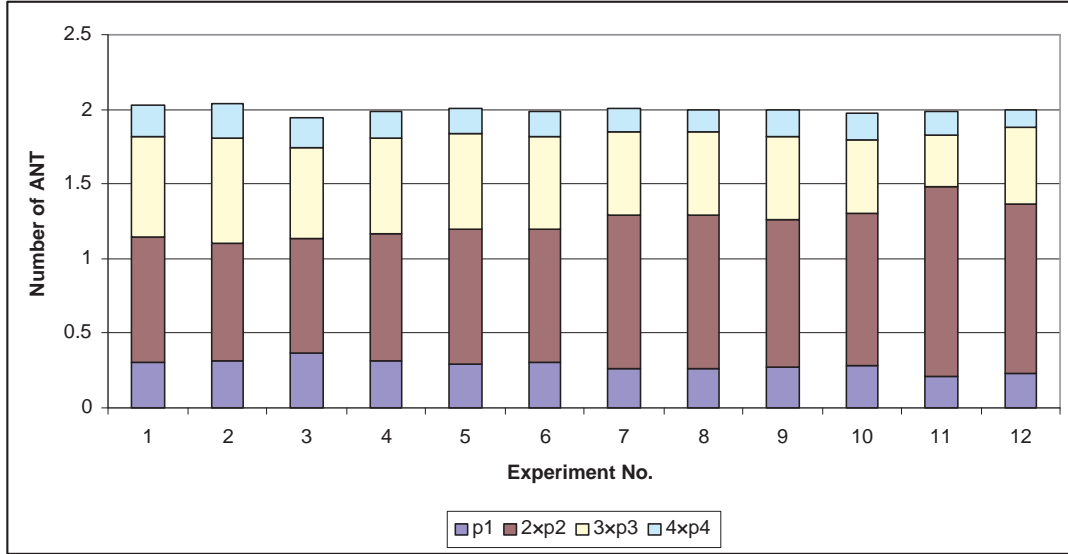
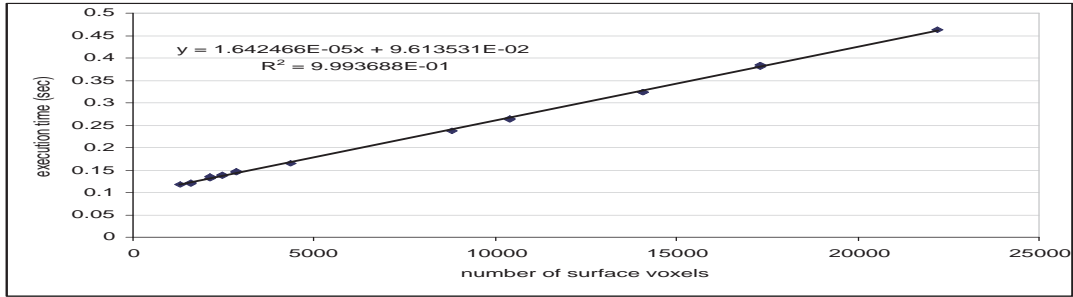


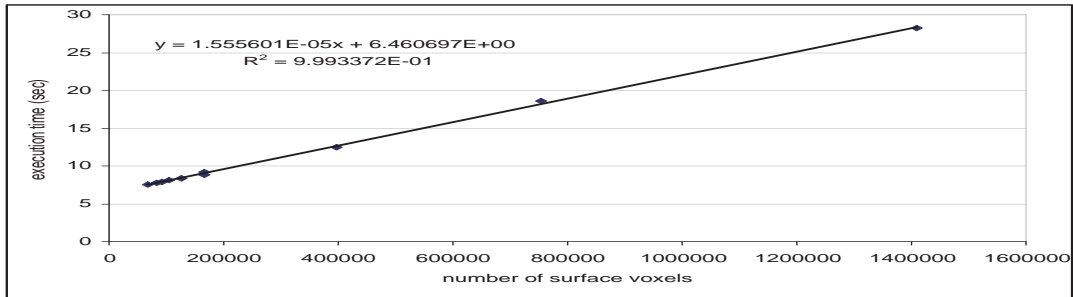
Figure 6.1: The *Average-case Number of Triangles*(ANT) distribution obtained from four datasets, each tested with three randomly selected isovalues.

a simple experiment to examine the probabilities, p_i , in which a surface voxel generates i ($=1,2,3$, or 4) triangles, respectively. By computing a straight-forward weighted average as $\sum_{i=1}^4 (i \times p_i)$, we obtain the mathematical expectation of the number of triangles generated by a surface voxel. We call the result of this weighted average *Average-case Number of Triangles* (ANT). Since different datasets as well as different iso-values chosen for the same dataset could both affect the value of ANT, we experimented with four different datasets (NegHip, Diesel Injection, UNC Head, MRI Skull) from both scientific and medical applications. For each dataset, three different isovalues are randomly chosen. We show the results of twelve (4×3) scenarios in Fig. 6.1, where each individual bar represents a separate test. The colored sections of each bar, in the bottom-up order, illustrate the values of $(i \times p_i)$'s for a surface voxel that generates $i = 1, 2, 3$, or 4 triangles, respectively. Although the exact values of $(i \times p_i)$'s varied on a small scale, the values of ANT remain relatively constant. The mean of ANT values is 1.99 with a standard deviation of 0.02.

Based on our observation on the consistency in the values of ANT, we construct a



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



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

Figure 6.2: The linear performance model of marching cubes: Different data points correspond to various iso-values chosen from the range between 30 and 150, assuming 8-bit values on the voxels.

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

We tested the linear cost model hypothesis using 64 cubed and 256 cubed volume blocks from four datasets described above. The linear regression results shown in Fig. 6.2 indicate that our linear model is highly coherent with practical scenarios in both cases.

In summary, our procedure to estimate the work load of marching cube algorithms is the following. First, for any block size and targeted computers chosen by a user, we move a small number of data blocks of that size striped from actual datasets to the targeted machines. Second, to obtain each data point in Fig 6.2, we repeatedly run a large number of

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

6.1.3 Raycasting

For raycasting [42], a common acceleration technique is early ray termination, which is a very simplistic yet very effective method on single processor. Unfortunately, as shown by previous researchers [34], early ray termination does not scale well in parallel implementations. To achieve scalability, visibility culling is usually performed at block level with pre-computed information such as *Plenoptic Opacity Function* (POF) [34]. The transfer functions for volume rendering are typically selected to produce semi-transparent volumes, in which case, early ray termination within voxel blocks may not lead to significant speedups. Therefore, without significantly compromising performances, we do not perform early ray termination but rely solely on block-based visibility culling. In this way, all volume blocks are subject to the same computation cost, i.e. $I_{b_i} = c$ where c is a constant value. This allows us to use a very simple metric to estimate the processing time of a volume dataset by multiplying c and the number of non-occluded blocks. The estimation of c can be done by running the ray casting algorithm on a large number of non-empty blocks and choosing the average time spent on a block.

In our experiment, we run the ray casting algorithm on a dataset with 512 non-

empty blocks of 64^3 voxels. The time to render each block is 0.387 seconds on average with a relative standard deviation of 4.7%.

6.1.4 Streamline Construction

For streamline computation, as long as the length of each streamline is predetermined, the computing time to construct even just a moderate number of streamlines is highly predictable.

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

$$t_{streamline}(n_{seeds}, n_{steps}) = n_{seeds} \times n_{steps} \times T_{advection}, \quad (6.2)$$

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

6.2 Bandwidth Measurement for Transport Time Estimation

We present a linear regression model to estimate the bandwidth of a transport path using active traffic measurement based on [66]. Due to complex traffic distribution over wide-area networks and the non-linear nature of transport protocol dynamics (in particular TCP), the throughput achieved in actual message transfers is typically different from both the link and available bandwidths, and typically contains a random component. We consider the *effective path bandwidth* (EPB) as the throughput achieved by a flow using a given transport module under certain cross traffic conditions. The notion of effective path bandwidth is specific to the transport protocol employed by the transport daemon. The active measurement technique we apply here is to estimate the effective path bandwidth

and minimum delay for each virtual link. Note that a virtual link in the overlay network of transport daemons may correspond to a multi-hop data path in wide-area networks, which usually consists of multiple underlying physical links from different networks.

There are three main types of delays involved in the message transmission over computer networks, namely, link propagation delay d_p imposed at the physical layer level, equipment-associated delay d_q mostly incurred by processing and buffering at the hosts and routers, and bandwidth-constrained delay d_{BW} . The delay d_q often experiences a high level of randomness in the presence of time-varying cross traffic and host loads. Also, since the transport protocol reacts to the competing traffic on the links, the delay d_{BW} may also exhibit randomness particularly over congested wide-area connections. We use Eq. 6.3 to measure the end-to-end delay in transmitting a message of size r on a path P with l physical links:

$$d(P, r) = d_{BW}(P, r) + \sum_{i=1}^l (d_{p,i}(P) + d_{q,i}(P, r)) \quad (6.3)$$

Due to the large size of data transfer in high-performance visualization applications, only the first term of Eq. 6.3 is significant and therefore the delay $d(P, r)$ of transmitting a message of size r along path P can be approximated by a linear model: $d(P, r) \approx r/EPB(P)$. The active measurement technique generates a set of test messages of various sizes, sends them to a destination node through a transport channel such as a TCP flow, and measures the end-to-end delays, on which we apply a linear regression to estimate the EPB.

The delay measurements between Louisiana State University (LSU) and Oak Ridge National Laboratory (ORNL) as well as its corresponding linear regression are illustrated in Fig. 6.3, 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 [13], 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

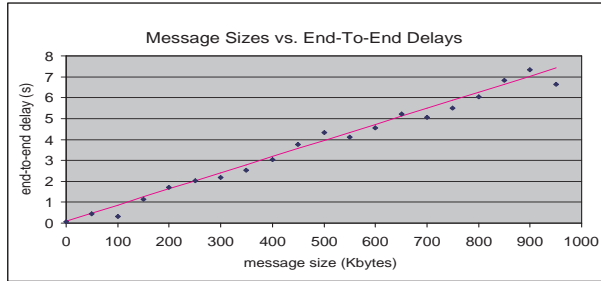


Figure 6.3: End-to-end delay measurements between ORNL and LSU

Table 6.1: Problem category of pipeline decomposition and network mapping.

prob	Mapping scheme	Module group	Node deployment	Solutions
1	Bijjective map	No	Linear arrangement	Trivial
2	Surjective map	Yes	predetermined number	Dynamic programming
3	Injective map	No	Arbitrary network with undetermined node	
4	General map	Yes		

transport parameters as the visualization modules, it is not sufficient to simply "plug-in" their estimates in Eq. 3.1.

6.3 System Optimization Based on Dynamic Programming

We propose approaches 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 solved by approximate solutions using the branch-and-bound technique but in exponential time[50]. However, with the understanding on the linear arrangement of the visualization modules, we are able to achieve optimal solutions to these problems in polynomial time using the dynamic programming method.

we consider several problems of pipeline decomposition and network mapping with different optimization goals under various constraint conditions, as tabulated in Table 6.3.

Since Problem 1 contains the same number of visualization modules and computing

nodes, both of which are a priori linearly arranged, a fixed one-to-one onto mapping is the only feasible solution and therefore no optimization can be applied. We list Problem 1 here for the sole purpose of comparison with other scenarios and will not discuss it any further. In Problem 2, the decomposition of the visualization pipeline is undetermined but all the intermediate participant nodes and their order are known in advance so that a surjective or onto mapping is required. Problem 3 maps each module of the visualization pipeline individually to a different node in an arbitrary computer network, in which the number and order of nodes are not determined. Problems 4 is a general cases of Problems 2 and 3. Note that in a general case, except for the source and destination nodes, the locations, number, and order of the intermediate nodes in the transport network selected for mapping are not determined prior to the visualization pipeline partitioning. The decomposition of the visualization pipeline is also undetermined. We propose polynomial-time optimal solutions based on dynamic programming for general case problems.

When the number of groups $q = 2$, the system results in the simplest client and server topology for remote visualization.

6.3.1 Minimal Total Delay

We consider Problems 2, 3, and 4 in Table 6.3 for total delay minimization under different stringent levels of constraint conditions in terms of pipeline decomposition and network node selections.

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. a mapped node only runs one 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. Algorithms and their complexities are discussed in details. Dynamic programming methods are exploited to reduce redundant

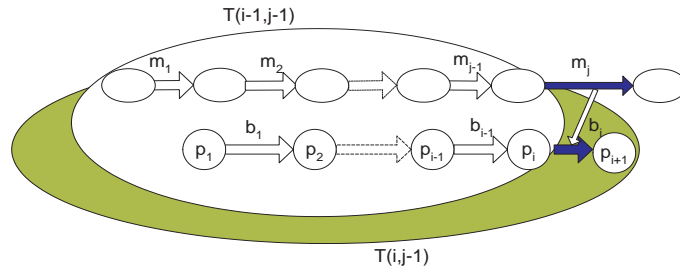


Figure 6.4: Dynamic programming for minimal transport delay time

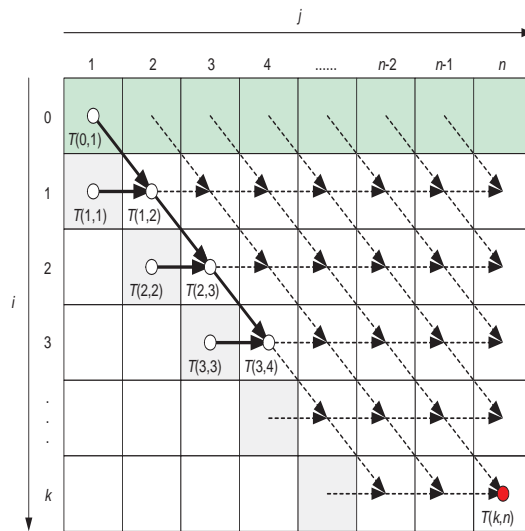


Figure 6.5: Construction of a 2D matrix of $T(i,j)$

calculation by utilizing previous computation results.

6.3.1.1 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 $m_j, j = 1, 2, \dots, n$, and k network links with bandwidths $b_i, i = 1, 2, \dots, k$ connecting $k+1$ computing nodes, each of which has processing speed $p_s, s = 1, 2, \dots, k+1$.

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 $T(i, j)$.

$$T(i, j) = \min_{i=1 \text{ to } k, j=2 \text{ to } n, i \leq j} \left\{ \begin{array}{l} T(i-1, j-1) + m_j/b_i + c_{j+1}m_j/p_{i+1} \\ T(i, j-1) + c_{j+1}m_j/p_{i+1} \end{array} \right\} \quad (6.4)$$

where the base conditions are computed as $T(0, t) = \sum_{i=1}^t c_{i+1}m_i/p_1$, $t = 1, 2, \dots, n$ in the first row and $T(t, t) = \sum_{i=1}^t m_i/b_i + \sum_{i=1}^t c_{i+1}m_i/p_{i+1}$, $t = 1, 2, \dots, k$ on the diagonal line. Note that the first module is assumed to be a data source 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. 6.4. At each step of the recursion, $T(i, j)$ takes the minimal of two subcases as illustrated in Fig. 6.4. 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 sub-problem 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 sub-problem of size i and $j-1$ with the addition of the computing time of the last module on the last node.

Fig. 6.5 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 is calculated from $T(2, 2)$ and $T(1, 2)$, which is in turn calculated from $T(1, 1)$ and $T(0, 1)$. Once the base cases are reached, the tracing process will stop and bounce back to its starting point. On the diagonal line with an equal number

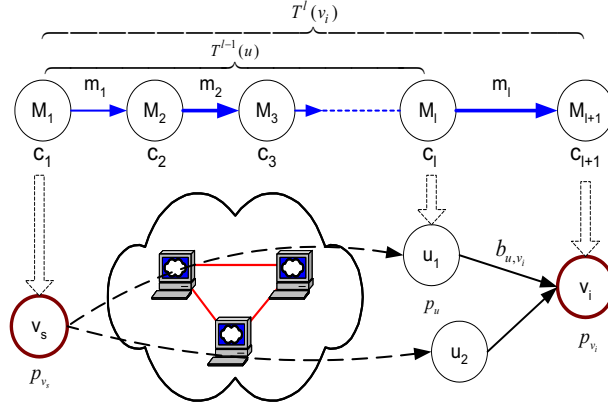


Figure 6.6: Illustration of n-hops shortest path algorithm

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.

6.3.1.2 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 $T^n(v_d)$.

$$T^l(v_i) = \left\{ \min_{u \in \text{adj}(v_i)} \left(T^{l-1}(u) + c_{l+1}m_l/p_{v_i} + m_l/b_{u,v_i} \right) \right\} \quad (6.5)$$

As Fig. 6.6 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: $T^1(v_i) = \begin{cases} c_2m_1/p_{v_i} + m_1/b_{v_s,v_i}, & \forall e_{v_s,v_i} \in E \\ \infty, & \text{otherwise} \end{cases}$. The complexity of this algorithm is $O(n \times |E|)$.

6.3.1.3 A General Mapping Scheme

we consider a general visualization pipeline partitioning and network mapping scheme. The objective of the system configuration is to minimize the total end-to-end delay for the fastest response by decomposing the visualization pipeline into any number of groups and mapping them to an arbitrary network. Let 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 $T^n(v_d)$ [68]:

$$T_{j=2 \rightarrow n, v_i \in V}^j(v_i) = \min \left(\begin{array}{l} T^{j-1}(v_i) + \frac{c_{j+1}m_j}{p_{v_i}}, \\ \min_{u \in \text{adj}(v_i)} \left(T^{j-1}(u) + \frac{c_{j+1}m_j}{p_{v_i}} + \frac{m_j}{b_{u,v_i}} \right) \end{array} \right) \quad (6.6)$$

with the base conditions computed as:

$$T_{v_i \in V, v_i \neq v_s}^1(v_i) = \begin{cases} \frac{c_2m_1}{p_{v_i}} + \frac{m_1}{b_{v_s,v_i}}, & \forall e_{v_s,v_i} \in E \\ \infty, & \text{otherwise,} \end{cases} \quad (6.7)$$

on the first column and on the first row in the 2D matrix of dynamic programming as shown in Fig. 6.7.

In Eq. 6.6, 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

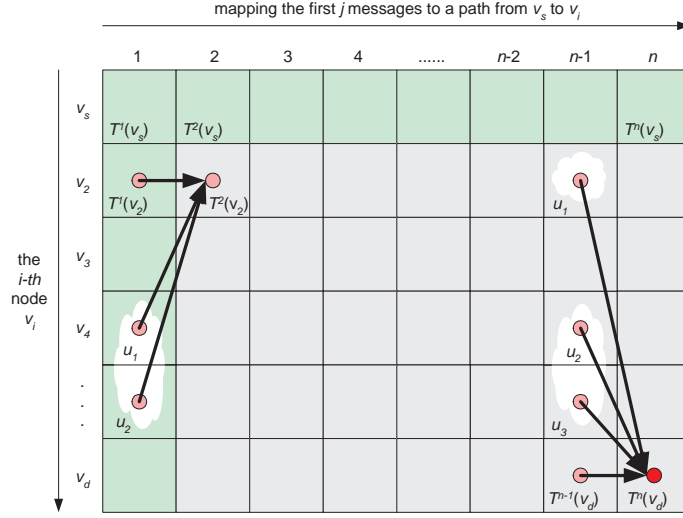


Figure 6.7: Construction of 2D matrix in dynamic programming

computing time of M_{j+1} on node v_i to $T^{j-1}(v_i)$, which is a sub-problem of node v_i of size $j - 1$. This 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. 6.7. 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 $O(n \times |E|)$.

6.3.2 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 these two different optimization goals, only the most general case is considered here.

6.3.2.1 Surjective Mapping with Linearly Arranged Network Nodes

This case is similar to the minimal frame rate case. 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 maximal frame-rate of the visualization pipeline. There are n messages flowing between $n+1$ visualization modules with sizes $m_j, j = 1, 2, \dots, n$, and k network links with bandwidths $b_i, i = 1, 2, \dots, k$ connecting $k + 1$ computing nodes, each of which has processing speed $p_s, s = 1, 2, \dots, k + 1$.

Let $T(i, j)$ denote the bottleneck processing time along the network path for each frame. We aim to minimize the bottleneck time to realize the maximal frame-rate. $S(i, j - 1)$ represents the total message size on node p_{i+1} when the first $j - 1$ messages mapped to the first i links. Then, we have the following recursion leading to the minimal bottleneck time.

$$T(i, j) = \min_{i=1 \text{ to } k, j=2 \text{ to } n, i \leq j} \begin{cases} \max(T(i-1, j-1), m_j/b_i, c_{j+1}m_j/p_{i+1}) \\ \max(\frac{S(i, j-1) + c_{j+1}m_j}{p_{i+1}}, T(j-1, i)) \end{cases} \quad (6.8)$$

where the base conditions are computed as $T(0, t) = \sum_{i=1}^t c_{i+1}m_i/p_1, t = 1, 2, \dots, n$ in the first row and $T(t, t) = \max_{i=1}^t \{m_i/b_i, c_{i+1}m_i/p_{i+1}\}, t = 1, 2, \dots, k$. The complexity of this algorithm is in the order of $O(n \times k)$.

Bokhari [24] solved the exact same problem by constructing a layered graph to find the optimal bottleneck path for maximal rate. However, the complexity of Bokhari's mapping chains to chains algorithm is much higher than our method and is in the order of $O(n^3 \times k)$.

6.3.2.2 Injective Mapping with Arbitrary Network

Similar to its minimal delay counterpart, 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 frame-rate is maximized. Since no grouping of visualization modules is allowed, the pipeline partition is actually predetermined.

Also this problem turns into an n -hops path problem. Let $T^l(v_i)$ denote the bottleneck 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 $T^n(v_d)$.

$$T^l(v_i) = \min_{l=1 \text{ to } n, v_i \in V} \left\{ \max(T^{l-1}(u), c_{l+1}m_l/p_{v_i}, m_l/b_{u,v_i}) \right\} \quad (6.9)$$

Based condition is computed as:

$$T^1(v_i) = \begin{cases} \max(c_2m_1/p_{v_i}, m_1/b_{v_s,v_i}), & \forall e_{v_s,v_i} \in E \\ \infty, & \text{otherwise} \end{cases}$$

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

6.3.2.3 A General Mapping Scheme

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 $T^n(v_d)$.

$$T^j(v_i) = \min_{j=2 \text{ to } n, v_i \in V} \left\{ \begin{array}{l} \max \left(T^{j-1}(v_i), (S^{j-1}(v_i) + c_{j+1}m_j)/p_{v_i} \right), \\ \min_{u \in \text{adj}(v_i)} \left(\max \left(T^{j-1}(u), c_{j+1}m_j/p_{v_i}, m_j/b_{u,v_i} \right) \right) \end{array} \right\} \quad (6.10)$$

with the base conditions computed as:

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

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

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, which achieves the maximal frame rate.

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 6.4- 6.10. 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.

Chapter 7

System Architecture and Implementation

Our distributed visualization system is a multi-threaded system and implemented in Java, Fortran, and C++ in Linux. The client GUI is based on GTK+ development. Our system consists of four virtual component nodes, i.e. client, central management (CM), data source (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 visualization routing path for the requested visualization task.

7.1 Post Processing System

Data is previously generated for post processing. Post processing system serves as the fundamental system for computational steering and wireless sensor network system.

7.1.1 System Function Diagram

The function diagram of four elements is shown in Fig. 7.1. Each node implements two types of communication channels: server channel for passive connection and client channel for active connection. In general, the incoming data are transmitted via the server

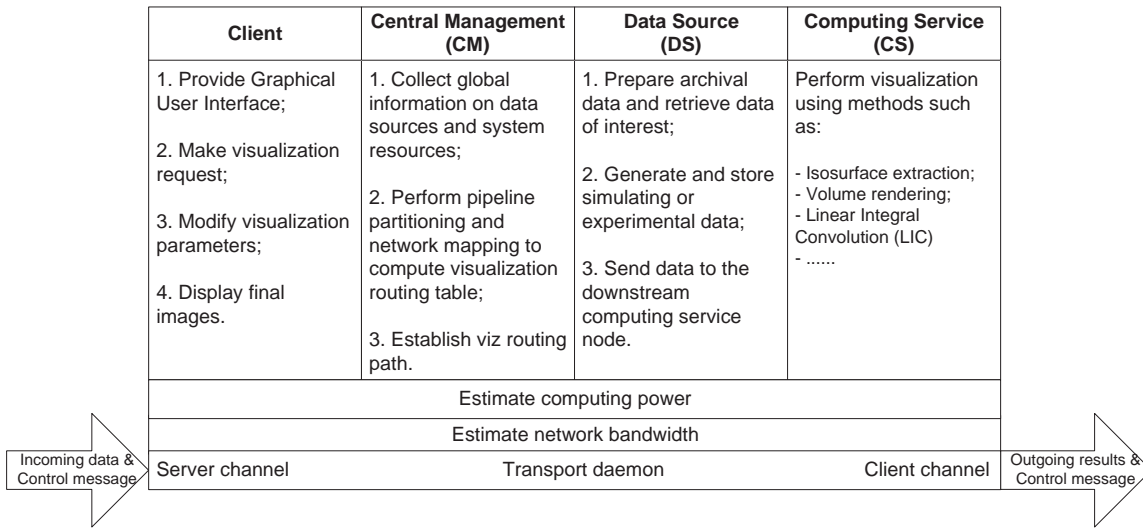


Figure 7.1: System function diagram.

channel from its upstream node, while the outgoing results are transmitted via the client channel to its downstream node. Control messages can be carried in both directions to report service failures, establish or close visualization routing paths, etc.

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

A client node usually resides on a host equipped with a display device ranging from a personal desktop to a powerwall. It displays the final images and provides end users an interface to interact with their visualization applications.

The main function of a CM node is to use the global information collected on data

sources and system resources to compute and establish the optimal visualization route for a specific visualization task. There are no critical computing and bandwidth requirements on the CM host because the CM may not perform any of the visualization modules.

A DS node usually stores pre-computed archival datasets. For online computational/experimental monitoring and steering, it might be a simulation running on a computing host or an experiment in process on an experimental facility. Filters are sometimes applied to retrieve data of interest hence reducing the raw data size. The retrieved data are sent to the downstream CS node along the routing path for further processing.

CS nodes can be located anywhere in the network and are of wide variety ranging from workstations to clusters and custom rendering engines. They receive data from upstream nodes, perform specific visualization computations, and output final or intermediate visualization results to downstream nodes. Note that some non-hardware-dependent computing modules may also be deployed on client, CM, or DS nodes to facilitate local processing.

7.1.2 Message-based Control Flow

Many previous remote visualization systems are based on a client-server model, which works very well as long as there are no intermediate nodes involved. While introducing intermediate nodes into remote visualization systems could lead to enriched functionality and better system throughput, the client-server model seems to lack a capability to support flexible and fault-tolerant control of such expanded remote visualization systems. A primary source of complexity in such systems stem from the fact that at any particular time, it may be a different subset of the intermediate nodes that are actually being utilized depending on the nature of a visualization job or request.

To support dynamic configuration of a different visualization pipeline across a different set of networked nodes, we adopted a message-based control flow. Each node in our system acts as an independent state machine. An operation is always triggered by a mes-

sage that has been received, while all outputs are sent out in the form of a message as well. On each node, to support a state machine, three threads are executed in endless loops, i.e. threads for receiving messages (RecvThread), processing data (ProcThread), and sending results (SendThread). While RecvThread and SendThread are the same among all nodes, the nature of ProcThread determines the type of a node. As described before, there are four types of system nodes. For ease of illustration, we have condensed the overall control flow for our entire system in Fig. 7.2.

A visualization job is always initiated by a client sending a request to a designated CM, after which the system is driven completely by user interaction (on the client) and control and data messages (for all other nodes including CM, DS, CS). System control messages are used to initiate/terminate a visualization sessions or report a service failure or close/establish visualization routing paths; visualization control messages are used to deliver visualization-specific information such as choice of visualization method, viewport resolution, viewing parameters, feature value (i.e. iso-value), and appearance definitions; data messages are used to transmit raw data, or visualization results such as geometry and intermediate volume rendered imagery results and final framebuffer. As shown in Fig. 7.2, the ProcThreads in different nodes takes their own specifically defined actions to process incoming messages.

One critical task in our system design and implementation is the dynamic computation and setup of a VRP by the CM node in response to a specific visualization request or sensed changes in network and computing environments. Upon receiving a new visualization request or modifications of visualization parameters (such as data source or visualization method), CM node employs the dynamic programming method to create a new routing table, and compares it with an existing one if any. Routing decisions are then made as per the following conditions: (i) if there is no existing routing path, CM initiates a new path for the pipeline and sends the routing table to appropriate nodes; (ii) if the existing

routing path is identical to the computed one, CM just retains the existing path; (iii) if the existing routing path is different from the computed one, CM closes the existing path and establishes a new one by sending the routing table to appropriate nodes; (iv) if the routing path exists and is the same as the computed one but has different assignments of computing modules, CM updates the assignments. Upon receiving a routing table, an intermediate node (DS or CS) simply creates a connection (if no connection exists) and forwards the routing table to its immediate downstream node.

The above routing decision conditions reduces the overhead incurred for the operations on routing tables and paths. Our system uses the latest VRP for all later communications unless a new visualization request arrives, a different visualization method is selected, or there are drastic changes in network traffic or node load conditions.

7.2 Computational Monitoring and Steering System

Computational steering system is built upon the post-processing system with additional simulation module. Fig. 7.4 demonstrates an exemplary interaction relationship among computational steering modules.

After establishing connection with CM, Client can issue a steering request to CM, which correspondingly makes a overall evaluation on current network resources and steering request. A routing table specifying the visualization pipeline partition and network mapping scheme for maximal frame rate is generated and propagated to the next node by CM. The network path is established while the routing table is being relayed downwards. The simulation node acting as data source initiates the simulation process upon the arrival of steering request and continuously pushes newly generated data to its downstream computing service node for further processing. This steering loop ends in the client for the final image display. Computing modules in Fig. 7.4 can locate on any physical computer node with proper computing capacity. Two or more modules can coexist on the same node if necessary.

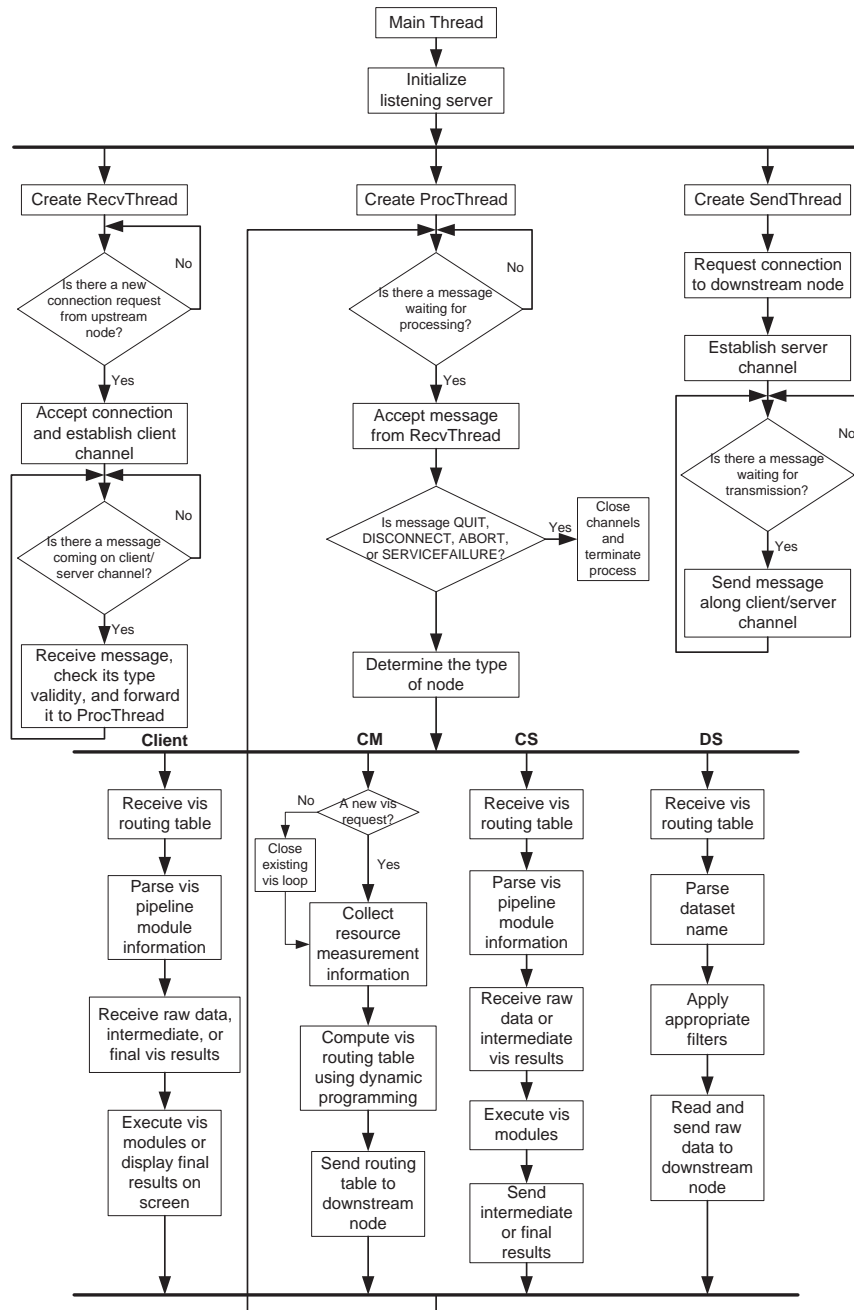


Figure 7.2: System control flow diagram.

As discussed in Chapter 4, visualization and transport function written in C++ are being called by the fortran simulation code. As shown in Fig.7.3, function VizStartupSimulationServer initiates the simulation channel for incoming message. VizWaitAcceptConnection function will accept connection request from CM. Simulation node will accept and handle incoming message until a new steering request gets in. The VH1 Fortran code initializes all parameters from input files and restart from old dump file if necessary. After these routines, VH1 code enters the main computational loops for pre-specified cycles. At specific cycles intervals, function VizPushDataToVizNode pushes partial data to next computing service node defined in the routing table for visualization processing. New steering parameters issued by client through GUI will be intercepted by function VizUpdateSimulationParameters as new set of computational parameters for next cycle.

We compile the visualization and transport related functions in C++ into a library using `g++ -c Simlib.cpp -o Simlib.o` and `g++ -o simulation Simlib.o vhone.o -lg2c` to generate the executable file by linking to the independently compiled fortran library `vhone.o`.

7.3 Wireless Sensor Networks Monitoring System

Wireless sensor networks system is also constructed based upon the post-processing system with additional sensor network control modules, namely java based Query Management module from TinyDB. To be on the same trajectory that the industry is traveling, we choose Crossbow (www.xbow.com) Mica2 platform. The sensor motes are implemented upon TinyOS(www.tinyos.net), an embedded operating system invented at the University of California, Berkeley.

Existing platforms for WSN operation include jWebDust, TinyDB, and Mote-VIEW [9, 11, 20]. They are focused on the operations of WSN, neither targeted at visualization methods nor optimized for network performance. We embedded TinyDB modules in our system for query processing through a declarative query interface that is similar to the SQL interface of a relational database system. Our sensor network system consists of one base

Virginia Hydrodynamics code

```
Viz_StartupSimulationServer();  
Viz_WaitAcceptConnection();  
Do Viz_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;  
    sweeey;  
    sweepz;  
  
    Viz_PushDataToVizNode();  
    Viz_ReceiveHandleMessage();  
    if ( Message is NewSimulationParameters)  
        Viz_UpdateSimulationParameters();  
While( cycels not endcycle)  
    End of main computational loops
```

Figure 7.3: Simulation pseudocode for computational steering

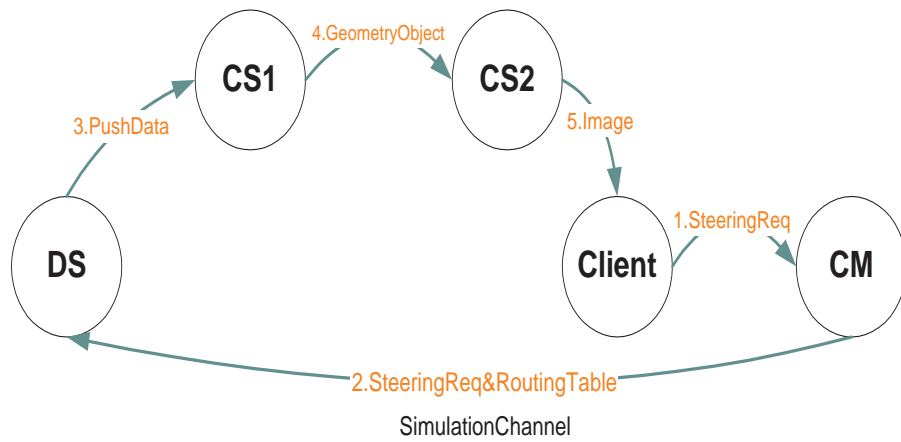


Figure 7.4: Interaction diagram for computational steering modules

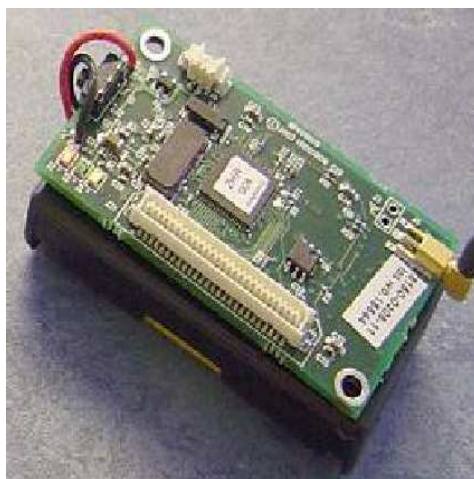


Figure 7.5: Mica2 sensor mote

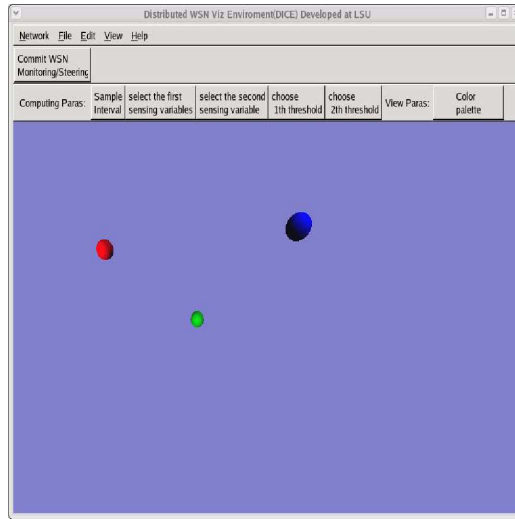


Figure 7.6: Sensor data viz window

station and multiple wireless sensor motes deployed in an ad hoc manner, and is treated as a separated data collecting entity.

Java based Query Management module interprets the query information from the client and broadcasts query to sensor motes via radio transmission emitted from the base station, through which the measured environmental data is routed back in multiple hops from viable sensor motes. Those motes that are not proprietary powered up or not within transmission range will be temporarily out of service. Sensor data belonging to the same time step will be saved as a single data file at DS node. CM will define a routing table for each sensor data file. Fig. 7.6 shows a snapshot from a sequence of temperature monitoring images from a remote sensor network with four motes. Different range of temperatures are represented by different colors for easy human visual perception. Furthermore, if equipped with GPS, our sensor motes are cable of self-localization, which enables outdoor environmental applications.

Unlike the wired network system, the ad hoc nature of WSN implies an inherent dynamic underlying infrastructure, which demands our system to be elastic and fault tolerant to handle any topological changes that would occur during the data collection process. We

design and implement two data buffer and dispatch protocols at the data server end to collect as much sensor data as possible within reasonable delay. No pre-assumed knowledge on the sensor network topology is required for data collection process. Namely, sensor motes are allowed to move and die during data collection.

Constrained by current 2D experimental terrain and limited number of sensor motes, our WSN visualization system uses simple visualization techniques. For example, vector data such as x, y velocity of a moving target can be used to plot the trajectory of a moving target using pathline technique; Scalar data such as temperature can be used to detect fire event.

Chapter 8

Experimental Results

8.1 Experiment Network Configuration

We deployed our system on six Internet nodes located at ORNL, LSU, University of Tennessee at Knoxville (UT), North Carolina State University (NCState), Ohio State University (OSU), and Georgia Institute of Technology (GaTech), respectively. Among them, the nodes at UT and NCState are clusters¹ with high-performance parallel computing capabilities, while the rest of nodes are PC Linux hosts with common hardware and software configurations. We run the client at ORNL, CM node at LSU, two DS nodes at OSU and GaTech, and two CS nodes at UT and NCState. The network configuration of the distributed visualization experiment is shown in Fig. 8.1.

We wish to visualize at ORNL (client) three experimental datasets, namely, Jet data of 16 MBytes, Rage data of 64 MBytes, and Visible Woman data of 108 MBytes², which are duplicated at OSU and GaTech. Note that whether or not to use MPI-based visualization modules installed on the clusters at either UT or NCState depends on the data objects, network connections, and host computing resources.

8.2 Performance Comparisons among Loops

We first perform a statistical analysis on transport measurements to estimate network bandwidths and on visualization measurements to estimate processing times. Using these estimates in the dynamic programming equations in Eqs 6.6 and 6.7, we compute

¹Due to the limitation of the graphics hardware support, only four processing nodes are utilized on each cluster.

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

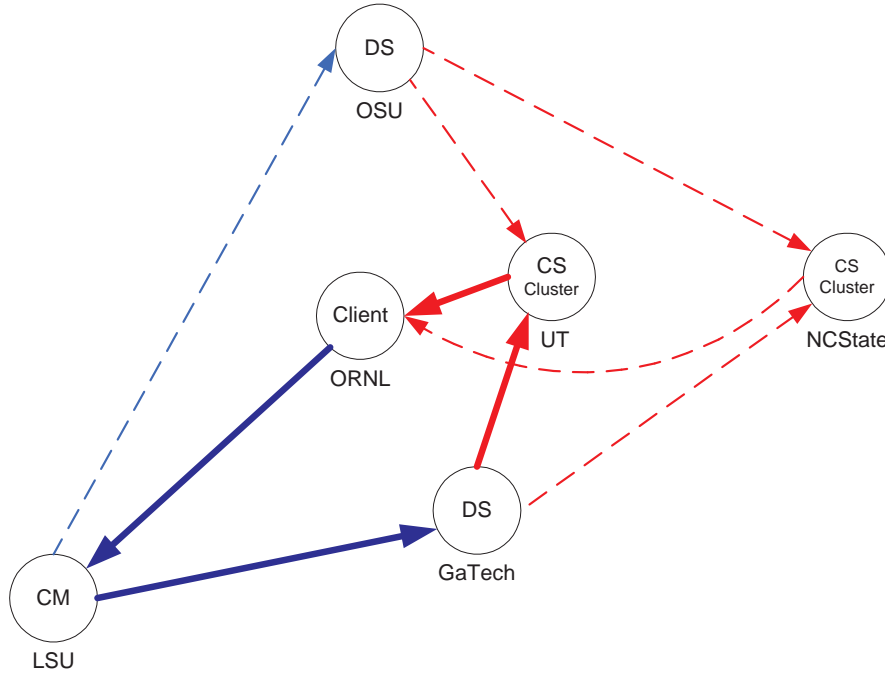


Figure 8.1: Network configuration of distributed visualization experiment.

Dataset size (MBytes)	Data sending time (seconds)		Image sending time (seconds)		Viz computing time (seconds)		Sum of measured times (SMT) (seconds)	Sum of estimated times (SET) (seconds)	Estimate error (%) $\frac{ SET-SMT }{SMT}$	Measured end-to-end delay (ME2ED) (seconds)	Overhead (ME2ED-SMT)	
	Measured	Estimated	Measured	Estimated	Measured	Estimated					Setup (sec)	Loop (sec)
Jet 16	3.51	3.49	0.23	0.22	5.92	5.55	9.66	9.26	4.14	10.52	0.86	
											0.50	0.36
Rage 64	13.81	13.17	0.24	0.23	9.60	10.32	23.65	23.72	0.30	24.60	0.95	
											0.70	0.25
Viswoman 108	23.38	22.93	0.23	0.23	13.94	13.76	37.55	36.92	1.68	38.40	0.85	
											0.61	0.24

Figure 8.2: Distributed visualization performance estimations and measurements along the optimal loop ORNL-LSU-GaTech-UT-ORNL.

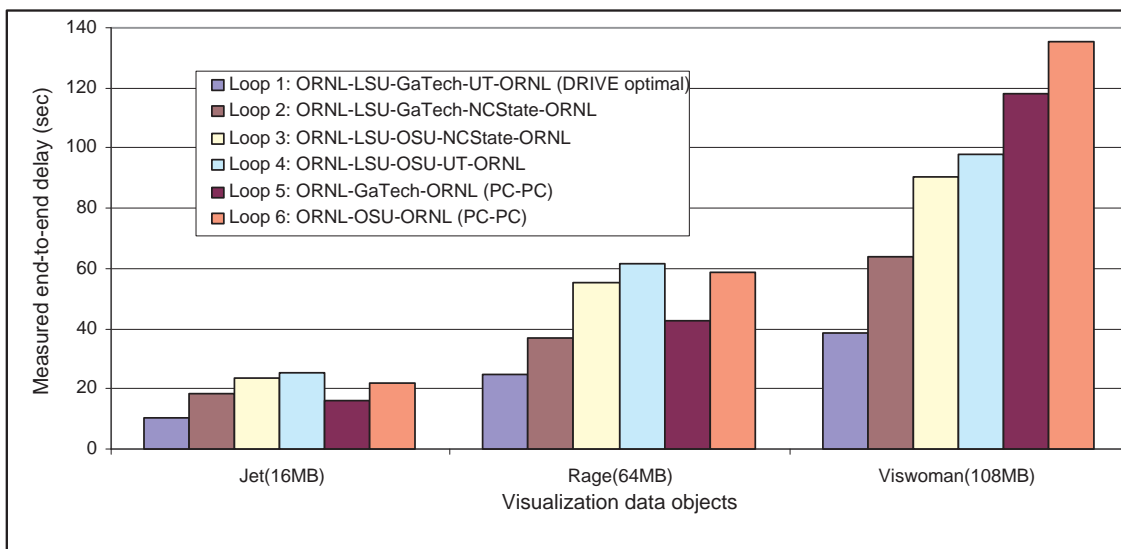


Figure 8.3: Performance comparisons between different visualization loops: Loop 1 along ORNL-LSU-GaTech-UT-ORNL (optimal), Loop 2 along ORNL-LSU-GaTech-NCState-ORNL, Loop 3 along ORNL-LSU-OSU-NCState-ORNL, Loop 4 along ORNL-LSU-OSU-UT-ORNL, Loop 5 along ORNL-GaTech-ORNL (PC-PC), and Loop 6 along ORNL-OSU-ORNL (PC-PC).

the routing table for each dataset. The optimal visualization pipeline GaTech-UT-ORNL is shown using solid lines in Fig. 8.1, wherein GaTech is used as a data storage node and UT is used as a computing service node. Together with the path ORNL-LSU-GaTech that carries the control messages, this solution forms the closed optimal loop ORNL-LSU-GaTech-UT-ORNL. The performance measurements for visualizing these three datasets using the isosurface extraction technique along this optimal loop are tabulated in Fig. 8.2.

As Fig. 8.2 shows, the overall estimation error of transport and computing times is within 5.0%, which demonstrates the accuracy of our performance models for network and visualization parts. We also observed that the system overhead is less than one second. This overhead consists of two components: setup time and loop time. The former is the time needed to compute a routing table and set up a visualization path, and the latter is the time spent in delivering control messages along the network loop for interactive visualization operations. Note that once a visualization path has been established, the system only has the loop time overhead, which is generally less than half second, if the routing table remains the same. This amount of overhead is almost negligible compared to the end-to-end delay on the order of dozens of seconds for large-scale remote visualization.

According to the experiment network configuration shown in Fig. 8.1, there are three other possible visualization loops using intermediate MPI-based visualization modules and two conventional PC-PC (client/server) visualization loops. For comparison purposes, we partitioned the same visualization pipeline in a similar way and mapped it onto each of these loops. Particularly, in the PC-PC experiments, since neither the GaTech nor the OSU host is equipped with a graphics card, we performed isosurface extraction on these two hosts acting as both a data source and a computing service node, and isosurface rendering on the ORNL host acting as both a client and a computing service node. The measurements of the end-to-end delay experienced by all these visualization loops using the isosurface extraction technique with an identical set of parameters are shown in Fig. 8.3

for a visual comparison.

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

It is interesting to point out that the advantage of utilizing an intermediate MPI module is not very obvious for small datasets because of the overhead incurred by data distributions and communications among cluster nodes. As a matter of fact, for datasets of several or dozens of MBytes, a simple PC-PC configuration with any type of server/client mode might be sufficient to deliver reasonable performances for remote visualization. However, for large-scale scientific datasets, parallel processing modules have become an indispensable tool supporting the visualization task. Hence, it also becomes increasingly important to select an appropriate set of processing nodes available in the Internet to map the visualization pipeline for the optimal performance. We only presented the performance evaluations on the isosurface extraction technique, whose actual run-time performance is much harder to predict than that of ray casting. The overall performances of our system employing other visualization methods are qualitatively similar.

8.3 Performance Comparisons with ParaView

In this dissertation, we propose that remote visualization systems can be implemented following a message-driven programming model and a state machine based method-

ology, so that self-adaptive pipeline configuration on intermediate nodes is facilitated. In addition, we have developed a framework to efficiently compute an optimal configuration using dynamic programming based on reliable underlying cost models. The entire dynamic programming process is very efficient (polynomial time), and primarily involve table lookups during the computation.

Although it would be time consuming for existing systems to extend the common client-server model into a network message driven model, our method to compute optimal pipeline configuration can be leveraged by any remote visualization system for initial system setup in a straightforward manner. Hence this allowed us to test how lightweighted our overall system is by running ParaView on the same optimal network configuration of the visualization pipeline as calculated by our system to for the same visualization job on identical datasets.

Specifically, our experiments involved running `pvdaserver` on the DS node at GaTech, `pvrenderserver` (executed by `mpirun` using the same four processing nodes) on the cluster-based CS node at UT, and `pvclient` at ORNL. Note that the CM node at LSU was not involved because ParaView does yet not employ such additional nodes. The end-to-end delay measurements using the same configuration for both ParaView and our system are illustrated together in Fig. 8.4.

We observed that for the same tasks, our system consistently achieved relatively better performances than ParaView. The performance differences may have been caused by higher processing and communication overhead incurred by visualization and network transfer functions used in ParaView. Instead of using third-party visualization packages, we developed and implemented our own lightweight visualization modules. However, it is not our intention to compare the efficiency of our visualization modules with those of other existing systems. Finally, we note that the mapping scheme from the visualization pipeline to the network nodes is manually performed as an initial setup in ParaView, and

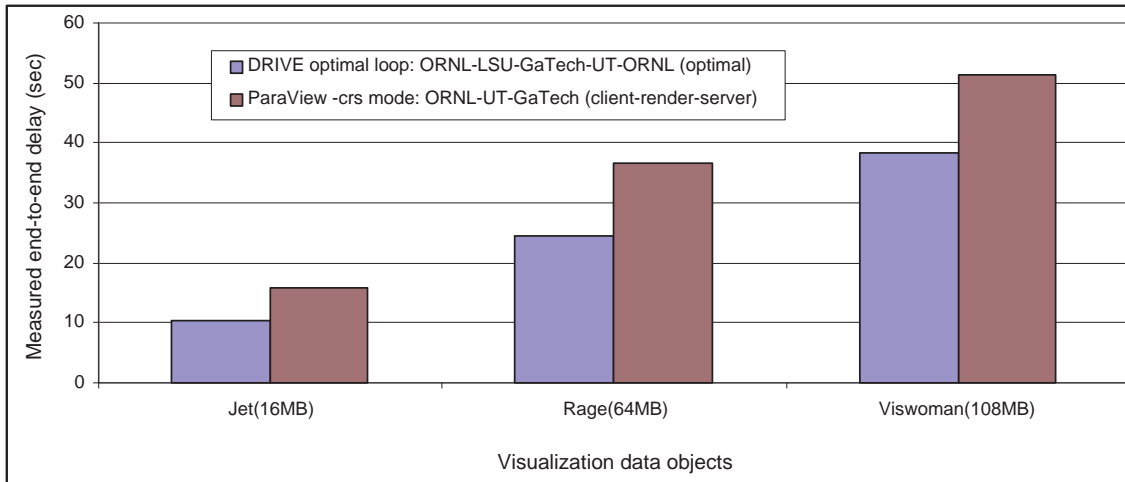


Figure 8.4: Performance comparisons between different visualization loops: optimal loop along ORNL-LSU-GaTech-UT-ORNL, ParaView -crs mode along ORNL-UT-GaTech.

the configuration was computed using dynamic programming by our CM node. In addition, the configuration in ParaView stayed constant throughout the whole visualization run, and for this test our system did not use runtime re-configuration either.

Chapter 9

Conclusion

In this dissertation, we proposed Distributed Remote Visualization System for visualizing large scientific data sets across networks. We presented the distributed visualization framework and mathematical model for automatically mapping a visualization pipeline to computer networks to minimize the total delay and maximize the frame rate of the visualization system. Domain-specific techniques were used to estimate the link transport and node processing times. We proposed a dynamic programming-based approach to compute optimal schemes for grouping and mapping the visualization modules onto a computer network.

Besides our theoretical contribution to high performance distributed computing, we also implemented a highly efficient and error tolerant visualization platform, which can handle scientific data ranging from archival data, dynamic simulation data to real time sensor data. Our computational monitoring and steering system allows direct interaction of human with simulation code, which provides researchers with a convenient visualization tool to monitor simulation process and restrain any stray simulation parameters. Computation resource such as supercomputer occupation time and development time can be saved. Remote visualization system on WSN enables safe environmental monitor and event detection, which has great potential in many civil and military application. Protocols and algorithms were proposed for WSN visualization system to address the data buffer and dispatch for periodic monitor application, and decision fusion for event detection application.

Instead of using commercial or open-source visualization package such as VTK, we developed our own parallel and serial visualization code aiming for a lightweight visualization system. Experimental comparisons with Paraview demonstrated that our system

achieve better performance despite the setup overhead on routing table and network channel.

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. To enable large scale web service functionality in our system, we also plan to encode our message and data results in XML to allow general public access. AI techniques such as neural networks may serve as ideal methods to estimate computing time for various visualization algorithms.

Besides in the Internet, we plan to deploy the distributed visualization system over dedicated networks, such as DOE UltraScience Net [6], for experimental testing. As the dataset 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 [65] [64] 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.

Bibliography

- [1] <http://retina.umh.es/Webvision/VisualCortex.html>.
- [2] Aspect. <http://www.aspect-sdm.org/>.
- [3] Circuit-switched high-speed end-to-end transport architecture. <http://cheetah.cs.virginia.edu/>.
- [4] Common data format. <http://nssdc.gsfc.nasa.gov/cdf>.
- [5] Computational engineering international. <http://www.ceintl.com/products/ensight.html>.
- [6] Doe ultrasciencenet. <http://www.csm.ornl.gov/ultranet>.
- [7] Hierarchical data format. <http://hdf.ncsa.uiuc.edu>.
- [8] human brain. <http://web.mit.edu/newsoffice/tt/1997/jan08/43171.html>.
- [9] jwebdust. <http://www.jWebDust.com>.
- [10] Louisiana optical network initiative. <http://www.cct.lsu.edu/projects/loni/index.php>.
- [11] Mote-view. <http://www.Mote-View.com>.
- [12] network common data form. <http://my.unidata.ucar.edu/content/software/netcdf>.
- [13] Network weather service. <http://nws.cs.ucsb.edu>.
- [14] Optical network introduction. <http://www.mitretek.org/pubs/mitretekBsummaries/SugmaBPubs/OpticalBNetworkBpub.pdf>.
- [15] Paraview. <http://www.paraview.org/HTML/Index.html>.
- [16] raycasting sample. <http://www.cs.rug.nl/michael/FANTOM/FANTOM1b.pdf>.
- [17] Teraflps and petaflps supercomputer. <http://www.llnl.gov/asci>.
- [18] Terascale browser. <http://www.llnl.gov/icc/sdd/img/terascale.shtml>.
- [19] Terascale supernova initiative. <http://www.phy.ornl.gov/tsi>.
- [20] Tinydb. <http://www.tinyos.com>.
- [21] Vh-1. VH-1 user manual.

- [22] Vis5d+. <http://vis5d.sourceforge.net/>.
- [23] A.R.Reibman and L.W. Nolte. Design and performance comparison of distributed detection networks. *IEEE Trans. Aerosp. Electron. Syst.*, AES(23):789–797, Nov 1987.
- [24] S. H. Bokhari. Partition problems in parallel, pipelined, and distributed computing. *IEEE Trans. Computers*, 37(1):48–57, Jan 1988.
- [25] I. Bowman, J. Shalf, K. Ma, and W. Bethel. Performance modeling for 3d visualization in a heterogeneous computing environment. In *Technical Report No. 2005-3, Department of Computer Science, University of California at Davis*, 2005.
- [26] Z. chair and P.K. Varshney. Optimal data fusion in multiple sensor detection systems. *IEEE Trans. Aerosp. Electron. Syst.*, AES(22):98–101, Jan 1986.
- [27] J.G. Chen and N. Ansari. Adaptive fusion of correlated local decisions. *IEEE Trans. Systems. Man. Cybernetics*, 28(2):276–281, May 1998.
- [28] T.T. Elvins. A survey of algorithms for volume visualization. *Computer Graphics*, 26:194–201, 1992.
- [29] M. J. Bertin et al. *Pisot and Salem Numbers*. user Verlag, Berlin, 1992.
- [30] R. Viswanathan F.A. Thomopoulos⁸⁷ and D.C. Bougooulis. Optimal decision fusion in multiple sensor systems. *IEEE Trans. Aerosp. Electron. Syst.*, AES(23):644–653, Sept 1987.
- [31] R. Ng S. Ahern R. Frank P. Kirchner G. Humphreys, M. Houston and J.T. Klosowski. Chromium: a stream processing framework for interactive graphics on clusters of workstations. *ACM Transactions on Graphics*, 21(3):693–702, 2002.
- [32] C. Chang R. Ferreira U. Catalyurek T. Kurc A. Sussman H. Andrade, M. Beynon and J. Saltz. Processing Large-Scale Multidimensional Data in Parallel and Distributed Environments. *Parallel Computing*, 28(5):827–859, 2002.
- [33] M. Beck S. Liu T. Moore J. Ding, J. Huang and S. Soltesz. Remote visualization by browsing image based databases with logistical networking. In *Proceedings of Supercomputing*, 2003.
- [34] H.W. Shen J. Gao, J. Huang and J. A. Kohl. Visibility Culling Using Plenoptic Opacity Functions for Large Data Visualization. In *Proceedings of IEEE Visualization '03*, pages 341–348, 2003.

- [35] J. Gallop M. Sagar K. Brodlie, D. Duce, J. Walton, and J. Wood. Visualization in grid computing environments. In *Proceedings of IEEE Visualization*, pages 155–162, 2004.
- [36] C. Ernst K. Engel, O. Sommer and T. Ertl. Remote 3d visualization using image-streaming techniques. In *ISIMADE'99*, pages 91–96, 1999.
- [37] O. Sommer K. Engel and T. Ertl. An interactive hardware accelerated remote 3d-visualization framework. In *Proceedings of Data Visualisation*, pages 167–177, 2000.
- [38] A. Kaufman. *Trends in visualization and volume graphics, Scientific Visualization Advances and Challenges*. IEEE Computer Society Press, 1994.
- [39] C. Hansen K.L Ma, J. painter and M. Krogh. Parallel volume rendering using binary-swap compositing. *IEEE Computer Graphics and Applications*, 4(14):59–68, 1994.
- [40] D. K. Knuth. *The T_EXbook*. Addison-Wesley, 1984.
- [41] L. Lamport. *L^AT_EX: A document preparation system*. Addison-Wesley, 2nd edition, 1994.
- [42] M. Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, 1990.
- [43] Y. Livnat. and C. Hansen. View dependent isosurface extraction. In *IEEE Visualization '98*, pages 175–180, 1998.
- [44] J. Llinas. Fusion-based methods for target identification in the absence of quantitative classifier confidence. In *SPIE Signal Processing, Sensor Fusion, and Target Recognition Conference*, Orlando, FL, April 1997.
- [45] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *Computer Graphics*, 21(4):163–169, July 1987.
- [46] E.J. Luke and C.D. Hansen. Semotus visum: a flexible remote visualization framework. In *Proceedings of IEEE Visualization*, pages 61–68, 2002.
- [47] F. Mittelbach M. Goosens and A. Samarin. *The L^AT_EX Companion*. Addison-Wesley, 1994.
- [48] K. L. Ma and D. M. Camp. High performance visualization of time-varying volume data over a wide-area network. In *Proceedings of Supercomputing*, 2000.

- [49] A. Neeman., P. Sulatycke., and K. Ghose. Fast remote isosurface visualization with chessboarding. In *Proceedings of Parallel Graphics and Visualization*, pages 75–82, 2004.
- [50] O. Virtajoki P. Frnti and T. Kaukoranta. Branch-and-bound technique for solving optimal clustering. In *Int. Conf. on Pattern Recognition*, pages 232–235, 2002.
- [51] A. J. V. D. Poorten. Some problems of recurrent interest. Technical Report 81-0037, School of Mathematics and Physics, Macquarie University, North Ryde, Australia 2113, August 1981.
- [52] L. Sobierajsk R. Avila and A. Kaufman. Towards a comprehensive volume visualization system. In *Proceedings of IEEE Visualization*, pages 13–20, 1992.
- [53] M. Moore R. Niu, P. Varshney and D. Klamer. Decision fusion in a wireless sensor network with a large number of sensors. In *The 7th International Conference on Information Fusion*, pages 21–27, Stockholm,Sweden, Jun 2004.
- [54] N. S. V. Rao. Distributed decision fusion using empirical estimation. *IEEE Transactions on Aerospace and Electronic Systems*, 33(4):1106–1114, 1996.
- [55] A.R. Reibman and L.W. Nolte. Optimal detection and performance of distributed sensor systems. *IEEE Trans. Aerosp. Electron. Syst.*, AES(23):24–30, Jan 1987.
- [56] N.R. Sandell R.R. Tenney and Jr. Detection with distributed sensors. *IEEE Trans. Aerosp. Electron. Syst.*, AES(17):501–510, July 1981.
- [57] M. Magallon S. Stegmaier and T. Ertl. A generic solution for hardware-accelerated remote visualization. In *Proceedings of Data Visualisation*, pages 87–95, 2002.
- [58] F.A. Sadjadi. Hypothesis testing in a distributed environment. *IEEE Trans. Aerosp. Electron. Syst.*, AES(22):134–137, March 1986.
- [59] M. Spivak. *The joy of T_EX*. American Mathematical Society, Providence, R.I., 2nd edition, 1990.
- [60] R. Srinivasan. Distributed radar detection theory. *Proc. Inst. Elec. Eng.*, 133:55–60, 1986.
- [61] P. Varshney. *Distributed detection and data fusion*. Springer-Verlag, 1997.
- [62] J. Lee D. Gunter W. Bethel, B. Tierney and S. Lau. Using high-speed wans and network data caches to enable remote and distributed visualization. In *Proceedings of Supercomputing*, 2000.

- [63] L. Westover. Footprint evaluation for volume rendering. *Computer Graphics*, 24(4):367–376, 1990.
- [64] Q. Wu. *Control of transport dynamics in overlay networks*. PhD thesis, Dept of Computer Science, Louisiana State University, 2003.
- [65] Q. Wu and N.S.V. Rao. A class of reliable udp-based transport protocols based on stochastic approximation. In *Proc. of the 24th IEEE INFOCOM*, Miami, Florida, March 13-17, 2005.
- [66] Q. Wu, N.S.V. Rao, and S.S. Iyengar. On transport daemons for small collaborative applications over wide-area networks. In *Proc. of the 24th IEEE International Performance Computing and Communications Conference*, Phoenix, Arizona, April 7-9, 2005.
- [67] A. Finkelstein Z. Liu and K. Li. Progressive view-dependent isosurface propagation. In *Vissym'01*, 2001.
- [68] M. Zhu, N.S.V. Rao Q. Wu, and S.S. Iyengar. Adaptive visualization pipeline decomposition and mapping onto computer networks. In *Proc. of the IEEE International Conference on Image and Graphics*, Hong Kong, China, December 18-20, 2004.

Vita

Mengxia Zhu is currently a doctoral student at Computer Science Department at Louisiana State University. She received the Bachelor of Engineering degree in biomedical engineering from the Zhejiang University in 1996. Her master's degree is in system science from Computer Science Department at Louisiana State University in 2002. She has also been working as research associate at Computer Science and Mathematics Division at Oak Ridge National laboratory. Her research interest include remote visualization and distributed sensor networks.