2009

Methods and design issues for next generation network-aware applications

Andrei Hutanu
Louisiana State University and Agricultural and Mechanical College

METHODS AND DESIGN ISSUES FOR NEXT GENERATION NETWORK-AWARE APPLICATIONS

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
Andrei Huţanu
Computer Engineering Diploma,
Politehnica University Bucharest, 2002
December 2009

Acknowledgments

This work would not be possible without the help and support of my current and former colleagues at the Center for Computation & Technology (Louisiana State University) and Zuse Institute Berlin, Germany.

I would like to thank all my collaborators from LONI, Masaryk University, MCNC, National Lambda Rail, Internet2, CESNET, the G-Lambda project team.

It is a pleasure to thank my advisory committee (Gabrielle Allen, Edward Seidel, Tevfik Kosar, Daniel Katz) and the *eaviv* team: Jinghua Ge for developing the parallel renderer and helping to integrate it in *eaviv* and Cornelius Toole for integrating tangible interaction in *eaviv* and for designing the optimization algorithm for remote data access.

I shall thank all my collaborators, in particular the following (not in any particular order): Ravi Paruchuri, Adam Yates, Steffen Prohaska, Andre Merzky, Jon MacLaren, Brygg Ullmer, Petr Holub, Miloš Liška, Erik Schnetter, Luděk Matyska, Peter Diener, Robert Kooima, Mehmet Balman, Kenneth Welshons, Lonnie Leger, Gigi Karmous-Edwards, Jon Vollbrecht, Rajesh Sankaran, Thomas Sterling, Tomohiro Kudoh, Brian Cashman, Shalini Venkataraman, Jarek Nabrzyski, Steven R. Thorpe, Yufeng Xin, John Moore, Stephan Hirmer, Hartmut Kaiser, Stephen David Beck, Archit Kulshrestha, Sam White, Marc Steinbach, Hans-Christian Hege, Ralf Kähler.

I thank Radu Chişleag (see [Chi03]) and the ARSIP program[1] for providing me the early opportunity to start my international career.

This work was supported by the EU GridLab project, the NSF Enlightened and CyberTools (NSF award #EPS-0701491) projects, the *eaviv* project (NSF award #OCI 0947825) and by the Center for Computation & Technology at LSU.

This project has been supported by a research intent "Optical Network of National Research and Its New Applications" (MŠM 6383917201) and "Parallel and Distributed Systems" (MŠM 0021622419).

[1] `http://www.arsip.com/`

Portions of this research were conducted with high performance computational resources provided by the Louisiana Optical Network Initiative (http://www.loni.org/).

Finally, I would like to thank my family and friends for their support throughout this journey.

Table of Contents

List of Tables

List of Figures

Abstract

Networks are becoming an essential component of modern cyberinfrastructure and this work describes methods of designing distributed applications for high-speed networks to improve application scalability, performance and capabilities. As the amount of data generated by scientific applications continues to grow, to be able to handle and process it, applications should be designed to use parallel, distributed resources and high-speed networks. For scalable application design developers should move away from the current component-based approach and implement instead an integrated, non-layered architecture where applications can use specialized low-level interfaces.

The main focus of this research is on interactive, collaborative visualization of large datasets. This work describes how a visualization application can be improved through using distributed resources and high-speed network links to interactively visualize tens of gigabytes of data and handle terabyte datasets while maintaining high quality. The application supports interactive frame rates, high resolution, collaborative visualization and sustains remote I/O bandwidths of several Gbps (up to 30 times faster than local I/O).

Motivated by the distributed visualization application, this work also researches remote data access systems. Because wide-area networks may have a high latency, the remote I/O system uses an architecture that effectively hides latency. Five remote data access architectures are analyzed and the results show that an architecture that combines bulk and pipeline processing is the best solution for high-throughput remote data access. The resulting system, also supporting high-speed transport protocols and configurable remote operations, is up to 400 times faster than a comparable existing remote data access system.

Transport protocols are compared to understand which protocol can best utilize high-speed network connections, concluding that a rate-based protocol is the best solution, being 8 times faster than standard TCP.

An HD-based remote teaching application experiment is conducted, illustrating the potential of network-aware applications in a production environment.

Future research areas are presented, with emphasis on network-aware optimization, execution and deployment scenarios.

Chapter 1
Introduction

With high-end distributed computational infrastructure, experimental devices and large data storage now becoming connected through new high-speed networks with the capacity to transport over one Gigabyte of data each second, networks should be considered an essential component of modern cyberinfrastructure.

One such infrastructure is the Louisiana Optical Network Initiative (LONI) [LON09]. LONI connects Louisiana and Mississippi research universities with a 10 Gbps research network, providing over 85 teraflops of computational capacity distributed across the network. On a national level, the NSF TeraGrid connects 11 sites, providing researchers with more than a petaflop of distributed computing capacity connected by high-speed networks [Ter09].

The work described in this thesis investigates fundamental issues and methodologies for how these networks can be used to build a new generation of applications designed to provide new capabilities by using existing and emerging cyberinfrastructure.

One pressing issue facing today's researchers is data. The amount of data consumed, manipulated and produced by large scale applications in science and engineering domains is increasing rapidly. This is because scientific instruments such as the Laser Interferometer Gravitational Wave Observatory (LIGO), or image acquisition devices such as computer tomography have increased their resolution and are collecting more data but also because local and national infrastructure for computation such as the LONI and the TeraGrid are expanding. Simulations in fields such as astronomy or climate modeling today routinely generate tens of terabytes of data per simulation as described in the DOE Office of Science report [DOE04]. The rate of data production increases every year, and the growth of digital data has been called a "data deluge" [HT03].

Applications are reaching their scalability limits in dealing with this data. One of the fundamental issues is the fact that we are close to hitting the limits of hardware scalability (single CPU performance), network performance (backbone capacity and utilization), disk, I/O and software scalability.

As the performance of these different systems improves we see a diminishing ability for applications to take advantage of all these resources at their maximum capacity.

A problem lies within the current approach of designing systems and applications which is to carefully separate the functionality of systems into independent and interacting components, each of the components being generally developed and optimized in separation, and only as a last step be integrated into complete systems.

These components then interact with each other using well defined interfaces that facilitate combining components easily into new types of systems and products. This approach has been a great enabler of discovery and innovation, and constant improvements in hardware performance has pushed back on the idea of using a different approach to build better, faster applications given the belief that the next, faster generation of hardware will achieve the desired performance improvement even if the application design process is unchanged. The result is that applications today do not adequately take into account improvements in parallel and distributed systems or networks, and as a consequence are not able to solve challenging problems such as the interactive visualization of terabytes of data, or provide a path for future scalability. This thesis describes a different approach for developing applications, one that uses an integrated, non-layered architecture.

Most codes today remain sequential, and the vast majority (around 90% [ENS05]) of programmers do not feel confident in writing parallel programs. However, single CPU performance has now hit the limit, and in consequence, the future is in multi-core, multi-processor systems [HP07] and parallel applications and not in sequential applications. While a small fraction of system designers are now able to use parallel computing, the resulting applications that can only use single parallel resources have a limited scalability potential.

A potential solution for further improving scalability is given by the grid [FKT01] and distributed computing. Distributed applications can use multiple clusters at various geographical locations to access more compute power than available from a single cluster. To be able to handle and process the large data being produced today, and to provide a scalability path for the future, it is crucial for applications to be able to utilize both parallel and distributed resources.

Networks represent the next frontier for distributed application development. The design philosophy [Cla88] of the Internet was to offer simple, robust and cost-effective access to all users and applications. The design has been an enormous success in that it has simplified network deployment and has facilitated ease of utilization for a wide range of applications. TCP/IP is currently the dominant network design and the system of choice for the Internet. The design principle for TCP/IP is based on the layered approach where the complexity of network operations is completely hidden from applications. Despite its advantages, such as ease of use and implementation, the layered approach hinders optimizations and the development of advanced applications. Information about network bandwidth, latency or capacity is not available to applications that use the TCP/IP system. The layered approach only allows communicating parties to attempt to transfer data between them, not knowing in advance how fast the data will be transferred, and does not allow the communicating parties to make optimization decisions (such as choosing the best network route) in order to transfer data faster. This is a significant shortcoming, one that is difficult or impossible to mitigate.

With a focus on interactive visualization, this work will show how by taking into account networks as first class resources, and using an integrated, non-layered approach along with parallel and distributed resources we can now design applications that provide better performance, scalability and improved capabilities over existing applications.

Next we look at usability and performance issues that appear in two types of networks: the public, commercial Internet infrastructure and research networks, the second being the main focus of the research presented in this work.

1.1 Commercial Internet

We look first at the commercial Internet as issues seen here will also have relevance for research and scientific applications. This section provides a few examples of what will happen in the future if the existing issues are ignored.

In the commercial world, architectural and historical reasons have produced an economy based on paying for network access rather than the actual cost of utilization. In fact, the existing network architecture does not currently support cost accountability [Bri07] making a cost-based economy

impossible to implement. As a result of the pay for access model, applications that have been more successful in utilizing the network capacity, such as file sharing using parallel and peer-to-peer transfers, video streaming, spam or tools crawling for information for search engines have succeeded and have done so at the cost of other, less adept applications such as web browsing.

Increases in the bandwidth of access networks have now moved the network bottleneck, from the "first-mile" (connection to the end-user) to the "middle-mile" [Lei09] (or the Internet backbone) and has led to various conflicts in the utilization of the congested network resources. A similar situation is now seen in cellular networks [Wor09]. We are currently witnessing conflicts in the Internet between Internet Service Providers and the applications that can successfully use the network where, in order to resolve conflict network providers are limiting access for particular users [Com09b] or applications such as bulk data transfer, peer to peer, software updates and newsgroups [Com09d]. These actions are changing the way the Internet is functioning, moving it away from its original design philosophy.

This conflict will probably lead to regulatory actions such as the 2008 decision on the Comcast Peer-to-Peer case [Com08a, Com08b]. Specifically, in this case Comcast has used special network devices that would investigate the data in a customers connections, and if there were too many connections from a particular type of file sharing application they were terminated by transmitting reset packets to both ends of the connection channel. The Federal Communications Commission (FCC) decision was to order termination of the Comcast practice. These regulatory actions will inevitably change the way the Internet works for users. For example Comcast's newest (FCC compliant) protocol-agnostic congestion scheme that resulted from the regulatory actions degrades the quality of VoIP applications just as much as it degrades any other applications, creating a difference between the quality of VoIP applications running over the Internet and those running over dedicated telephony circuits [Com09c]. This shows that any method of solving the congestion issue will have possibly unexpected consequences.

Internet consumers are now limited in the way in which they are able to utilize network services and capacity; and usage limits enforced by ISP's are common [Com07b, Com09a].

Congress hearings by the Federal Trade Commission and the FCC on "Network neutrality" may result in legislation [ea08] that will dramatically change the way the Internet functions as well as how

we view and use the Internet in the future. Unfortunately the technical issues that are creating the economical issues that we have to deal with today are not well understood and the arguments that are made by one side [Cer07] or the other [Coh06] seldomly tackle the underlying technical issues.

As computer scientists we need to facilitate a better understanding of the technical issues and of architectural options to influence future policy decisions so that they are made for the right reasons. The following references provide additional informative and balanced views on this issue [Com07a, Yoo08, For08, AW06, Ou08, vS08, Far08, Wal08, vSF09].

This section has illustrated some of the issues of the commercial Internet, indicating that changes in the way users and applications will use the network are probable in the future and showing the need for application developers as well as for network providers to consider new ways of thinking about the network. The current architecture is showing its limits and the growing conflicts between application and network providers indicate that a closer cooperation between them is needed in the future. Such cooperation is also needed between research network providers and scientific application developers and users.

1.2 Research Networks; Scheduling

In high-end computing the explosion of data generated by simulations and scientific experiments is increasing the data transfer volumes to the point that they cannot be managed using regular Internet services.

To move towards providing sufficient network capacity for scientific applications a number of high capacity networks have been deployed for the use of the scientific community. These are regional (such as LONI in Louisiana or NCREN [MCN09] in North Carolina), national (Internet2 [Int09] or NLR [NLR09] in the United States, CESNET [CES09] in the Czech Republic, PIONIER [PIO09] in Poland, DFN [DFN09] in Germany and JGN2 [JGN09] in Japan) and transnational (GÉANT2 [GEA09] across Europe).

The network capacity however is only one part of the problem experienced by applications. Increased capacity, from the tens of Mbps in the case of Internet, to tens of Gbps in the case of research networks also increases expectations that users have on the performance of their tools. Existing tools

and applications based on the current Internet architecture and protocols fail to provide the performance that scientists expect. For example, in Chapter 4, this thesis will show that applications using standard Internet transport protocols can only use a fraction of the available network capacity.

Another issue is the non-deterministic nature of shared infrastructure. In the compute world most applications are run in a time-share mode where each application has exclusive use of allocated resources for a period of time (managed by batch schedulers). This is a consequence of the unpredictable behavior that would be seen if applications run in a space-share mode, even though such use might lead to improved resource usage. For network resources a similar method, using dedicated, scheduled networks (see Section 1.5), can similarly be used to provide better performance for applications.

This thesis will show how applications can be developed using a different approach in dealing with the network, achieving a closer integration of the application and the network by: using experimental transport protocols; taking into account the properties of the network (such as bandwidth and latency); and relying on deterministic network services.

Distributed applications require the concurrent use of multiple resources. Existing infrastructure, such as LONI, although implemented as a distributed infrastructure does not routinely support execution of distributed applications. There are both technical and policy issues that prevent such usage.

On the technical side, management mechanisms for compute resources (job schedulers) were not designed to fundamentally support distributed applications, but to optimize the utilization of the local resource.

On the policy side, the decisions implemented by resource providers often inhibit the coordinated use of multiple resources. For example on LONI advance reservation is currently limited to a single node on each machine, making execution of distributed applications requiring more than one node on each machine possible only with administrator intervention. The majority of compute resources continue to be managed locally in all their aspects, and are controlled by a local scheduler, implementing local optimization policies. To utilize multiple resources in distributed applications, management systems and policies need to be changed so that multiple resources are managed together, not in separation.

Solutions to the technical issues are available, common mechanisms for data management, job management, security and accounting, information and monitoring are being developed and standardized by a strong community (OGF) and teams around the world (e.g. Globus, EGEE, NAREGI). With some effort, it should be possible to routinely use these tools in production compute infrastructure environments.

Local schedulers such as LoadLeveler, Torque (with Maui), LSF, PBSPro have been improved to support advance reservation mechanisms. Using advance reservation, a distributed application can be executed by reserving all the resources it needs in advance. Deployment of co-allocation tools is still experimental, for example in TeraGrid (see [Gro08]), however plans are underway that will hopefully enable TeraGrid co-allocation as a fundamental grid service.

1.3 Applications

An important issue today is that many application developers are relying on optimizations and feature enhancement in hardware or software subsystem implementations, without considering the need of changing the interfaces that they are using to interact with each individual subsystem. Because generic interfaces limit further improvements, applications that were written and designed to be run on current infrastructure have to be restructured in order to take advantage of next generation infrastructure (for example UNIX *cp* is not a good application for remote data copy).

A fundamental engineering trade-off exists between generic interfaces and specialized, high performance interfaces. Blocking I/O, sockets, OpenGL are examples of standard, generic interfaces that are suitable for applications that do not push the limits of the individual subsystems (storage, network or graphics). As we move towards using remote storage, high-speed networks and complex graphics hardware these interfaces are showing their design limits. Applications that need to take full advantage of hardware power should use more complex interfaces such as: asynchronous I/O, low level network provisioning services, and toolkits for parallel application development on the GPU (such as NVIDIA CUDA, OpenCL). Examples from distributed computing show the complexity of attempting to hide new semantics under old interfaces, e.g. RPC (attempting to execute remote operations as local) or Parrot [TL05] (attempting to make remote I/O look like local I/O).

The trade-off between generic, simple to use and inefficient APIs on one side, and specialized, complex but efficient APIs on the other side is becoming clear. Applications that are designed to use generic interfaces are inevitably designed to not be able to fully take advantage of existing hardware capabilities. To benefit from the continuous hardware improvements we need to design and use other, specialized high-performance interfaces.

1.4 Research Contribution

Although the research infrastructure combining high-speed networks and computational resources (such as LONI and TeraGrid) is available, the number of distributed applications that can fully take advantage of these combined resources is very small (see Section 1.5).

This thesis bridges this gap between network and compute resources, and analyses implementation and algorithmic approaches for the coordinated and integrated use of network and distributed compute resources for scientific applications. My research in designing a new class of distributed applications has led to the following theses:

- A network-aware interactive visualization application provides more capabilities, better performance and can handle larger data than existing visualization applications. The visualization application can actually be improved when using distributed resources.

- Taking into account network latency and network bandwidth utilization from the design stages of a remote data access system leads to a pipeline-based asynchronous architecture that enables higher operations throughput and faster data transfer than existing remote data access systems.

- High-speed networks can be efficiently utilized if a user-defined rate-based data transport protocol without congestion control is used.

- Other network-aware applications can use high-speed network services to improve overall performance, for example to improve frame-rate and resolution as demonstrated in a distributed high-definition classroom environment.

In short, this research has the following components:

- The design and implementation of a three-way distributed visualization system that uses storage, network, compute, rendering, display and interaction resources in a coordinated way and takes advantage of networks to improve I/O and rendering performance (Chapter 2).

- The research and design of a new remote data access architecture that enables efficient coupling of the data and rendering stages of the distributed visualization application. The implementation of a high-throughput remote data access system following this design (Chapter 3).

- Analysis of experimental benchmarks of appropriate transport protocols for dedicated networks connecting the data access and rendering stages of the distributed visualization application (Chapter 4). Analysis of experiments carried out for a network intensive application of high definition video for remote teaching (Chapter 5).

This research will help in defining future network services and new ways for applications to interact with networks, as well as contributing to an understanding of how network and compute service providers need to cooperate to create future cyberinfrastructures.

1.5 Related Work

Only a limited number of applications have been designed specifically for high-speed networks.

OptIPuter [SCD+03] is a large project that has built an advanced infrastructure connecting computational infrastructure with high-speed "lambda" networks to create virtual distributed metacomputers. The OptIPuter project has produced a wide range of high-speed network tools, such as the SAGE [RJH+09] video distribution system, the LambdaRAM [VZL08] distributed network cache, the LambdaStream [XLH+05] data transport protocol and the OptiStore data management system [Zha08]. OptIPuter technologies are being applied to scientific applications in areas such as microbiology [SGP+09] and climate analysis [VBLS09].

The Sector/Sphere system [GG08] has been designed to support scalable distributed data storage (sector) and distributed processing (sphere) using high-speed networks.

The CineGrid community [dLH09] is developing tools that enable the production, use and exchange of high-quality digital media over high-speed networks, for example uncompressed 4K video streaming [SKF+09].

The CoUniverse framework is focused on automatic organization of collaborative environments, adaptation on changing network conditions, and high-quality video streaming tools supported by high-speed networks [LH08].

Lambda Station [BCD+06] is a tool that automatically allocates network services and steers traffic to them, as needed by large size data transfers.

Several other experiments with applications of high-speed networks have been performed, for example with distributed visualization [SBSdL06], simulation of the human arterial system [DIK+06], and computational astrophysics [MGYC06].

One method for tightening the integration of applications and networks is to use reserved, exclusive access to network resources controlled by the user. This requires a mechanism that allocates, reserves and separates network resources for different applications.

One promising approach, on-demand provisioning of lightpaths, has already been used in a series of experiments around the world (Phosphorus, G-Lambda, EnLIGHTened, described below). A different approach is to separate traffic on the same physical medium using Virtual LANs[1].

Some of the first networks that provided dedicated dynamic connections (with bandwidths of 50 Mbps to 10 Gbps) were DOE UltraScienceNet [RWCW05, N. 08] and NSF CHEETAH [ZVR+05].

The EnLIGHTened computing project [BHKE+07] designed an architectural framework that allows Grid applications to dynamically request (in advance or on-demand) any type of Grid resource: computers, storage, instruments, and deterministic network paths, including lightpaths and implemented co-allocation of compute and network resources.

The Phosphorus project [FCL+07] focused on delivering network services to Grid users and making applications aware of their complete Grid environment (including both computational and network resources). The project enabled the use of the heterogeneous network infrastructure across multi-domain or multivendor networks.

The goal of the G-lambda project [THN+06] was to define a standard interface (GNS-WSI) that could be used by grid services to allocate network resources provided by commercial network operators.

[1]Defined as a part of IEEE standards: `http://standards.ieee.org/getieee802/`

The StarPlane network enables dynamic provisioning of lightpaths for use in the DAS-3 distributed metacomputer [GMM+09] in the Netherlands.

The VIOLA project worked to improve the UNICORE middleware to enable co-allocation of compute and network resources in Germany [EWW+07].

Internet2 has recently introduced the ION service. ION is a new virtual circuit network service that allows users and applications to allocate on-demand network circuits that provide guaranteed, dedicated network bandwidth [ION09].

ESnet has recently activated a dynamic circuit network dedicated solely to scientific research, called the Science Data Network (SDN).

Various systems have been developed to support the above mentioned user or grid application control of network links, for example UCLP [BGI+03] or OSCARS [GRT+06]. The TeraPaths [KYGM07] project investigates creating end-to-end virtual paths with bandwidth guarantees. These are used to prioritize and protect network flows according to user requests. For more information on networks for grid systems see "Grid Networks" [TMKE06].

Chapter 2
Distributed Interactive Visualization

As data sets continue to increase in size, scientists need to address the growing issue of effectively visualizing and analyzing their data. This chapter describes how scalability in interactive visualization can be greatly improved by using networks to take advantage of diverse distributed resources.

One motivating scientific application producing large data sets requiring interactive visualization is the numerical modeling of relativistic astrophysical systems, such as the collision of black holes and the supernovae collapse of neutron stars [CBB+07]. These simulations routinely generate terabytes of data, where a single time-step for one variable has a resolution of 4096^3 data points or higher, and simulations involve tens of variables and thousands of time-steps. Future simulations of gamma-ray bursts are predicted to produce petabytes of data as early as 2011 [OSA+08].

A second scientific application area is the interactive visualization of image datasets. Tomography datasets acquired by x-ray scans of flame retardant solution have a size of 32 Gigabytes (2048^3 resolution), and a single experiment can generate 24 or more datasets [HHB08].

In both cases, it is common for the data to be stored remotely at the supercomputer or instrument where it is created. For researchers to progress in their scientific endeavors they need interactive and collaborative visualization tools that can be used from their offices and laboratories on university campuses to analyze this data.

The goal of this work is to design a visualization system that is interactive, collaborative and supports large data. Such a system must have the following characteristics: interactive frame rate (5 frames per second or higher), large data (tens of gigabytes per time-step, terabytes in total data size), high resolution images (1 million pixels and higher), fast data transport, and fast updates (less than one second/update) of visualization as data is being read into the system, good quality (no visualization artifacts, quick response to interaction), support for collaborative visualization (multiple, possibly distributed users are able to see and interact with the visualization) and use an approach that will support expanding future technology.

Such visualization is a challenging problem, the interactive, real-time nature of the process adding an additional degree of difficulty compared to non-real-time applications. This challenge motivates the use of new methods and ideas that leverage emerging technology in graphics processing units (GPUs), networks, clusters and compute architecture.

The numerical relativity use case provides a motivating scenario for this work, illustrated in Figure 2.1. The large data to be visualized is stored near the supercomputer where the simulation was run. The scientist is connected via a high-speed network infrastructure to the server holding the data. At various locations in the network; compute and graphics resources are available that can be used to provide interactive visualization to the scientist at her location. To enable the scientist to better collaborate with her colleagues the system supports collaborative use from multiple users at different locations.

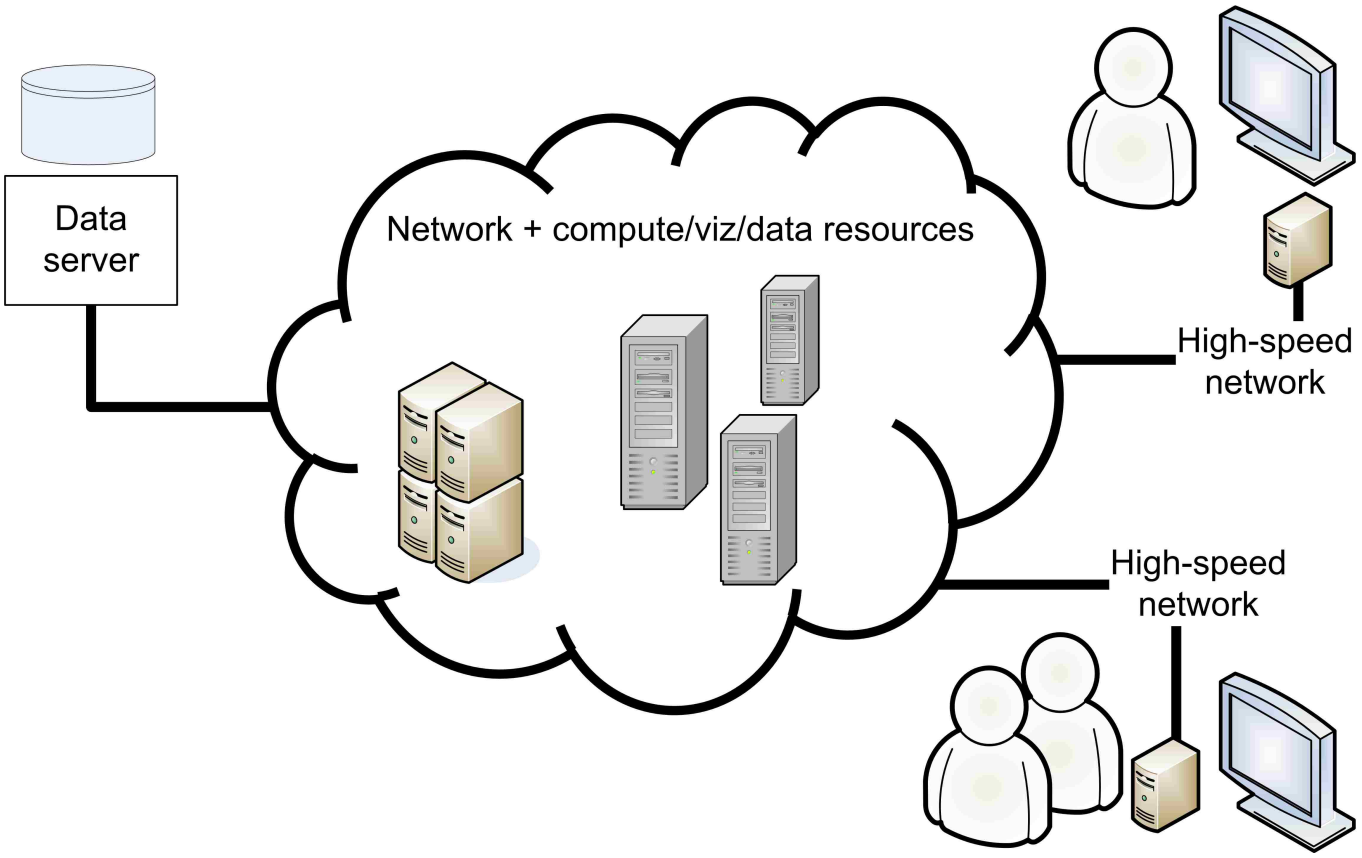

FIGURE 2.1: Illustration of a motivating visualization scenario with remote data, distributed visualization components, and collaboration over high speed network links.

In addition to the situation when the nature of the application (distributed data, distributed users) require using distributed technologies, there are real examples that perform better when using distributed resources than on a single local resource. Distributed applications can both use multiple, distributed resources, but also can access powerful remote resources that are not available locally. This defines a second motivating scenario for this work, distributing the visualization application because it brings a real benefit to the user, not because it is required logistically. High-speed networks provide the opportunity to use powerful remote resources that are not available locally, and to combine multiple distributed resources in a single visualization application. As dataset size increases it is crucial to take advantage of all available resources to construct and execute next-generation visualization applications.

In this work I describe how it is possible to build a distributed visualization system that is truly interactive, handles large remote data, produces high resolution images, and supports collaboration, using an integrated, network-aware, application design. The resulting system (called *eaviv*) has been built over the past four years and has provided a number of research challenges, not only in designing the visualization system (described in this chapter) but also in other areas such as remote data access, transport protocols and optimization. The work to address these challenges is described in the following chapters. I designed and implemented the complete visualization system except the parallel renderer and the integration of interaction devices. These components were developed by CCT collaborators. Where available, existing technologies such as UDT [GG07] and SAGE [RJH+09] were used to construct the system. The next section describes related and background work for the visualization system, Section 2.2 describes the overall architecture and design of the system and Section 2.3 presents implementation details, results and evaluation for two systems: an early implementation that was used for a distributed visualization experiment at iGrid 2005 and the current *eaviv* system.

2.1 Related Work

The visualization process, in particular when distributed, is described as a visualization pipeline (see Figure 2.2). The pipeline is composed of five stages: a data source (which may be either a disk or

memory), a data filter (for example selecting the data of interest), geometry generation (for example creating triangles), rendering, and finally image display. Rendering is the process that transforms visualization primitives (such as points, or triangles) into images.

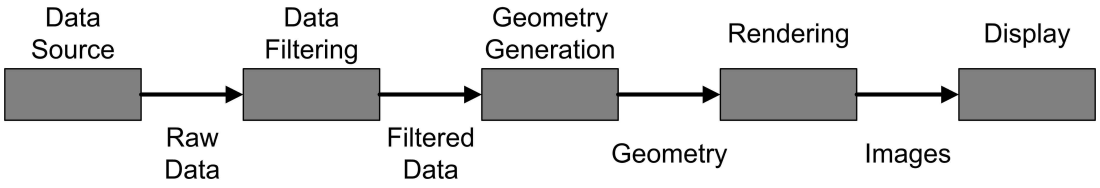

FIGURE 2.2: Visualization pipeline showing the five different stages

Desktop visualization tools provide a wealth of advanced visualization algorithms optimized for interactive control, however these are limited by the size of available memory, and do not scale to the large data sets researchers deal with today.

For large datasets, a common approach used to create visualization systems is to build distributed visualization pipelines. Some distributed visualization applications have been motivated by the idea of improving performance by taking advantage of distributed resources while other applications have been motivated by the need to facilitate visualization of remote data.

The existing distributed systems that have been motivated by performance improvements do not provide the level of performance needed for current scientific applications. For example the RAVE system is able to visualize a few megabytes of data at under 10 frames per second, rendering at 400x400 pixel resolution and unknown latency for interaction or data transfer speed [GAW09]. The pipeline optimization by Zhu et. al. [ZWRI07] resulted in a system that can visualize datasets smaller than 10 megabytes at a speed of one frame every 5 or more seconds. Other systems are described by their architecture but have as now no implementation. For example the distributed visualization architecture proposed by Shalf and Bethel [SB03] could support a variety of distributed visualization applications and inspired the development of the *eaviv* system.

The limitations of desktop-based visualization led to the development of parallel visualization systems and frameworks such as ImageVis3D [Ima09], Chromium [HHN+02] and Equalizer [EMP08] that can take advantage of computational clusters to visualize large datasets. Equalizer and Chromium are parallel rendering frameworks that can be used to build parallel rendering applications. ImageVis3D

is a parallel rendering tool for interactive volume rendering of large datasets. These and other tools and techniques are being developed (for example as part of the Institute for Ultra-Scale Visualization [MWY+09, MRH+07]) to be able to take advantage of parallel resources for visualization. Some of these tools focus only on parallelizing the rendering process and do not deal with the issue of data or image transfer over the network.

Other visualization systems such as ParaView [CGM+06] and VisIt [CBB+05] were designed to facilitate the visualization of remote data and, while they have the capability to transmit data and images over the network, they are not able to take advantage of the full capacity of high-speed networks and thus have low data transport performance, can suffer from a lack of interactivity and image quality, and do not support collaborative visualization.

ParaView is a parallel visualization application designed to handle large datasets. It supports two distribution modes: client–server and client–rendering server–data server. In the client–server mode the client connects to a parallel renderer running on a cluster. The rendering is performed either on the server (if data is too large) or locally, after transferring the data to the client. The system makes a decision based on configurable parameters, and video or data are transmitted from the server to the client for viewing or rendering respectively. In the client–rendering server–data server distribution, the server is separated into two components, a data server that is responsible for filtering operations and runs where the data is located and the rendering server which does only rendering, no data processing. These two components are usually executed within the same local area network.

VisIt is a visualization software designed to handle large datasets using client–server distribution of the visualization process. Similar to ParaView's client–server distribution, VisIt uses a parallel rendering server and a local viewer and interaction client. Visit's visualization pipeline can be either server-side rendering with image streaming to the client for visualization methods such as parallel ray-tracing, or server performing data processing with data being transferred to the client machine for interactive accelerated visualization. Most commonly, the server is as a stand-alone process that reads data from files. An alternative exists where a simulation code delivers data directly to VisIt, separating the server into two components. This allows for visualization and analysis of a live running simulation. In the current implementation, VisIt's server-side component and the simulation need to

be executed on the same cluster. VisIt has recently been shown to be able to render (although not interactively) more than 10^{12} cells [BJA+09].

On the TeraGrid [Ter09] visualization of remote data is supported through two different models: *Server provided visualization* where remote visualization sessions are launched using VNC [RSFWH98]. Within VNC, a wide variety of visualization applications can be executed. VNC transports the images from the remote machine to the local client and keyboard or mouse interaction commands from the client to the remote machine; *Web portal assisted client–server visualization* where the TeraGrid's visualization gateway[1] launches a ParaView server to which users can connect using a ParaView client running on a local machine.

Visapult [BS03] is a distributed, parallel, volume-rendering application. Visapult's processing pipeline has three components: a raw data source, a viewer, and a visapult back-end. The data source component, usually placed near a distributed parallel storage system, feeds the multi-process visapult back-end using multiple parallel data streams. Each process of the visapult back-end renderer feeds images to the viewer which combines them using an image-based rendering-assisted volume rendering (IBRAVR) [MSHC99].

Semotus Visum [LH02], is a framework for distributing the visualization pipeline into two components between the client and server machines. This framework allows several server-client configurations in which the visualization stages can either reside within the client, server or shared between both.

gViz [BDG+04] developed an XML format to describe the visualization process and grid enabling of modular visualization environments that can be combined in a flexible way. gViz is used in a system called eViz whose goal is to provide an adaptive infrastructure for distributed collaborative visualization [BBC+07].

VIRACOCHA [GHW+04] is a distributed post-processing and visualization tool that combines a parallel data filtering and processing system with parallel rendering, these two components being executed at possibly different locations in the network.

[1] `https://viz.teragrid.org/`

VLMIC [SPM06] combines image caching and distributed rendering into a system that can interactively browse multiresolution datasets, independent of the rendering algorithm that is utilized.

Many other distributed visualization systems have been proposed, however none of the existing systems provide the performance and quality (described at the beginning of the chapter) necessary to support the motivating scenarios.

As the number and quality of parallel rendering tools increases, taking into account networks to be able to take advantage of remote parallel rendering systems or high-speed storage systems will become an important part of the visualization application development process.

The visualization method that this work is focused on is volume rendering, a method that does not require intermediate geometry so the number of stages in the visualization pipeline is four. These are: data source (disk or memory), data filtering (selecting the area of interest, such as a sub-sampled version of a data volume, or a timestep), rendering (volume rendering) and display on the screen.

In initial background work on distributed visualization, we built two experimental systems for the visualization of remote data and collaborative visualization using video streaming respectively. These two systems represent initial stages in the *eaviv* application design, but are also representative of existing visualization systems that are not able to take advantage of high-speed networks.

2.1.1 Visualization of Remote Data

The design of the first experimental system was focused on investigating mechanisms to interactively visualize and explore large remote datasets. Here we separated the visualization pipeline in just two sections: a section containing the data stages and a section containing the rendering and display stages, with a network link connecting the two (Figure 2.3).

Only a low network capacity was available at the time for these experiments, and so the distributed visualization system [HHK+05, PHKH04, SMHS04] was designed to reduce the amount of data transferred over the network. The data source and filter components of the visualization pipeline were located on the server storing the data to be visualized, while the rendering and display were placed on a workstation local to the user.

The system used a progressive visualization approach which allowed us to meet the goal of supporting visualization updates every one or two seconds. The visualization was interactive (the frame

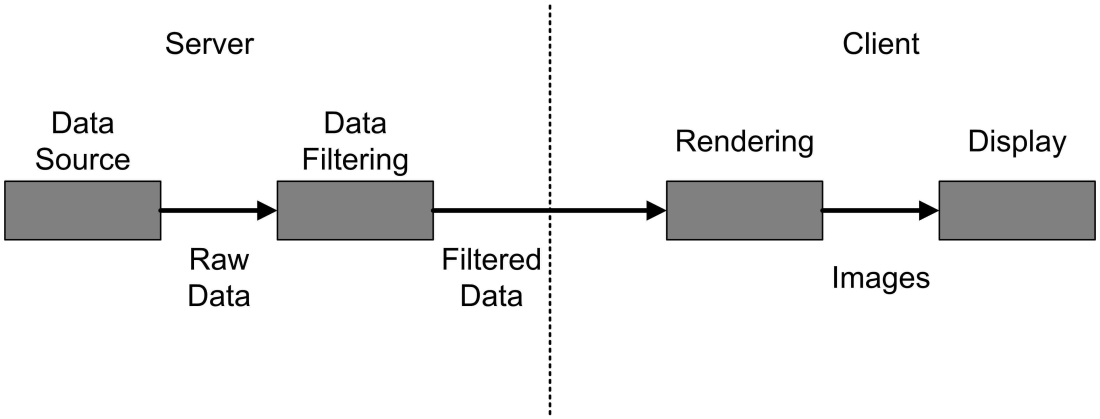

FIGURE 2.3: Visualization pipeline for remote data access: data server and filtering are on the server; rendering and display on the client

rate and resolution were controlled locally), however mainly because of network speed limitations, the total amount of data visualized at any time was limited. Another issue was that I/O operations were serialized, which led to network latency having a damaging effect the on overall application performance. Where possible, I/O operations that could be combined into a single remote operation, were executed together, an approach that was very effective in reducing the remote data access time. (by a factor of up to 30 [HHK+05]). The size of data that was visualized at any given time was limited by the capacity of the workstation used for the visualization client.

2.1.2 Video Streaming

Another method of building a distributed visualization application is to separate the visualization pipeline in two sections, the first section containing all the pipeline stages except display, this section being run on the server containing the data and the second section being the image display which is run on the client machine (Figure 2.4). Examples of popular systems supporting this architecture are VNC and SGI Vizserver [Sil05]. Image transport over the network between the two sections is called video streaming.

In the second experimental system leading up to *eaviv* this distribution approach was used to enable collaborative and remote visualizations [ZSH05, HBH+04, HHS+04]. The advantage of the video streaming approach is that multiple users can receive a copy of the video stream (by using multicasting), enabling collaborative remote visualization when multiple users have the necessary remote interaction mechanisms. Network speed limitations restricted the system to a low-bandwidth,

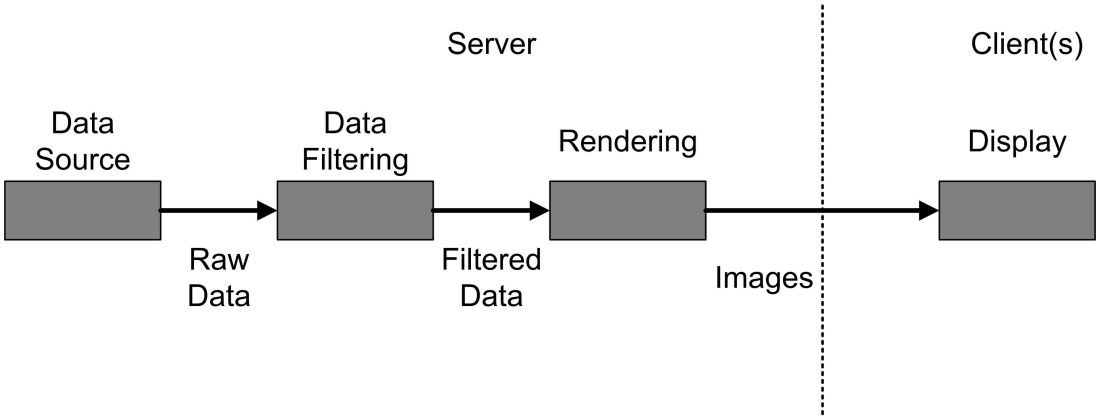

FIGURE 2.4: Visualization pipeline for video streaming, all the stages except for the display are carried out on the server; the display is carried out on the client

low resolution, thin client multicast solution. The sender was directly integrated as a streaming server in the visualization application (Amira [SWH05]). Remote interaction was available for a single user using the mouse and keyboard, the other participants being able to passively participate in the visualization session and interact with each other by other means (audio and videoconferencing). The system could achieve a high frame rate, but because of bandwidth limitation this was at the cost of using high compression (resulting in image artifacts) and low resolution.

This system illustrated the potential of collaborative visualization and thin client visualization based on video streaming. More specifically it highlighted the need to provide interaction devices for each user and the need for high-speed networks for good quality video.

2.2 System Architecture and Design

The *eaviv* system is designed to use distributed resources to create a visualization application that is better than applications using only local resources. The introduction of LONI in Louisiana in 2005 provided access to 10 Gbps network connections. With the improved network distributed resources could now be used to improve the capabilities of the visualization process itself, not because it is required by the application scenario. Three main features are made possible by the use of distributed resources and high-speed networks:

- The first improvement is to increase the I/O bandwidth of the visualization application to reduce the data loading time. The data-rendering separation described in Section 2.1.1 is used

to separate the visualization front-end from a distributed data access server with high-speed networks connecting the machines running these components.

- The second improvement of the visualization system is to use high-speed networks to enable high-quality collaborative visualization.

- The final improvement is to integrate parallel rendering methods in the distributed visualization system to take advantage of powerful graphics clusters located in the network. Remote, network-connected graphics resources are used to increase the amount of data that is interactively visualized.

The architecture of the three-way distributed *eaviv* system is provided in Figure 2.5.

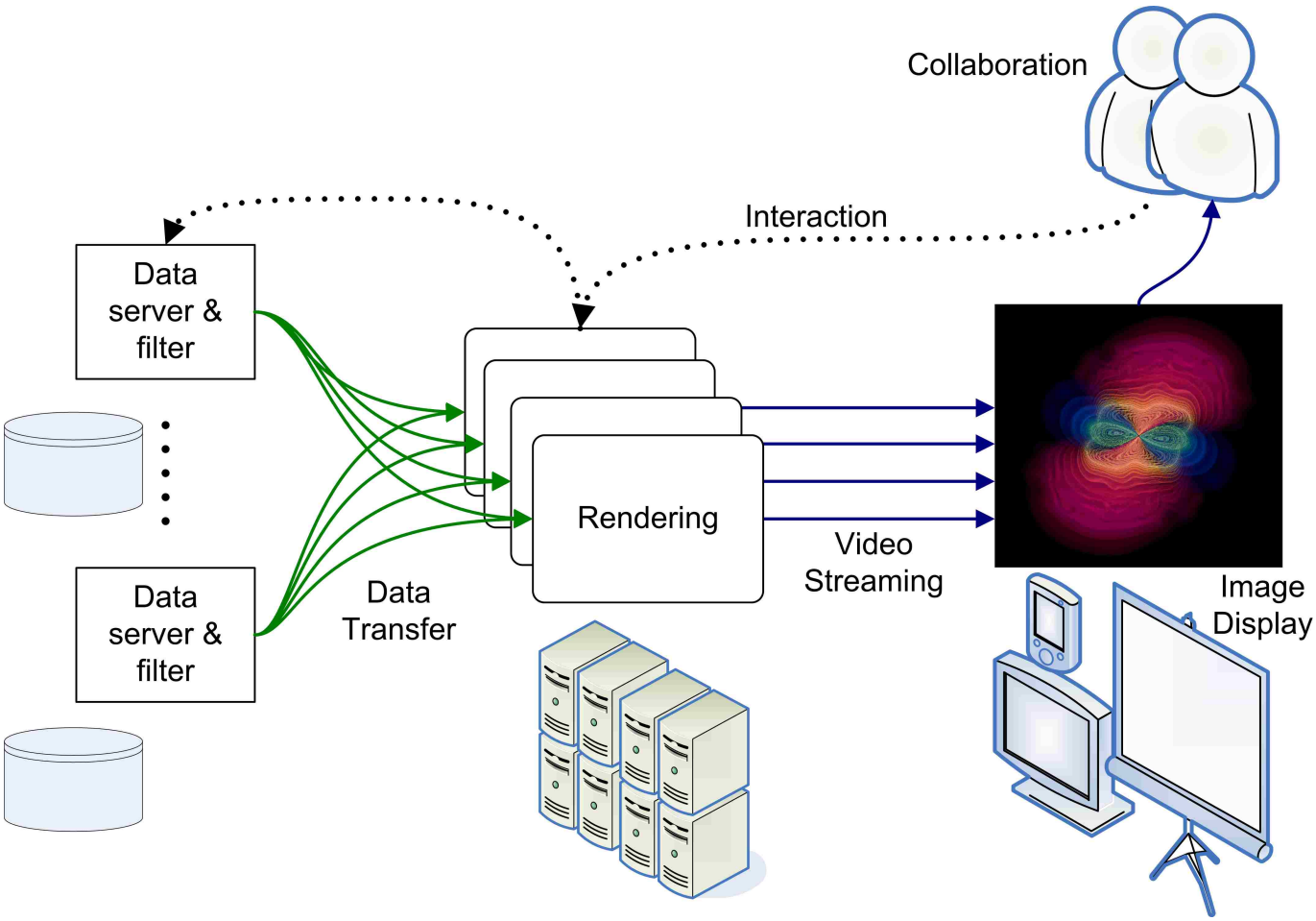

FIGURE 2.5: Architecture of the *eaviv* system providing three-way distributed visualization using video streaming and remote data access.

2.2.1 Remote I/O

The first component of the *eaviv* system is I/O, or remote data access. Using high-speed networks can improve the data transfer speed of the remote data access component, as described below.

Separating the data from the rendering components improves performance because the storage resources local to the rendering process can be a bottleneck for the data transfer rate of the application. Disk speeds lag behind the speed of network interfaces and by distributing the data into the network the load time experienced by the visualization application is reduced. A distributed data server can sustain higher data transfer rates than a single, local data source, and transferring data from the memory of remote machines is faster than transferring it from the local disk. This improves the responsiveness of the application as seen by the user.

In effect, the system can use a large pool of memory distributed over multiple remote computers, similar to the approach in LambdaRAM/Optiputer [ZLD+03]. The amount of data for which the high network bandwidth can be sustained when using distributed resources is higher than the capacity of the local main memory. The idea of utilizing network RAM for application speed-up is derived from the concept of virtual memory and its roots can be traced back to the 90's [CG90].

With distributed resources network latency can become an issue for the application. The I/O system needs to be restructured to take advantage of the fast network speeds. *eaviv* uses a pipeline remote data access architecture that allows for data requests and responses to be transmitted and processed in parallel, thus supporting fast execution of a large number of remote data access operations (high operation throughput). This is used to support the many operations generated by the progressive visualization process described next in Section 2.2.2. An important feature of the data access system is that it is non-blocking. This ensures that the interactive usage is not degraded and the application does not freeze while reading in the data. Issues related to the remote data access system are described in full detail in Chapter 3.

High-performance data transmission over wide-area networks is difficult to achieve. One of the main factors influencing data transport performance is the network transport protocol. Using unsuitable protocols on a wide area network can result in very poor performance, for example 10 Mbps on a

dedicated 10 Gbps long-distance (200 ms round-trip-time) network connection using TCP (Transmission Control Protocol). This issue and possible solutions are described in Chapter 4.

The *eaviv* system uses high-speed transport protocols such as UDT that support a high network throughput on long-distance and high-capacity network links.

Figure 2.6 illustrates the interactions between the components of the visualization system as triggered by a user request for new data to be visualized. When a user requests that a new portion of the dataset should be visualized, the visualization application determines which section of the data needs to be supplied by each server and communicates the individual selection to the servers. Upon receiving the requests the servers start delivering their data in parallel to the visualization application.

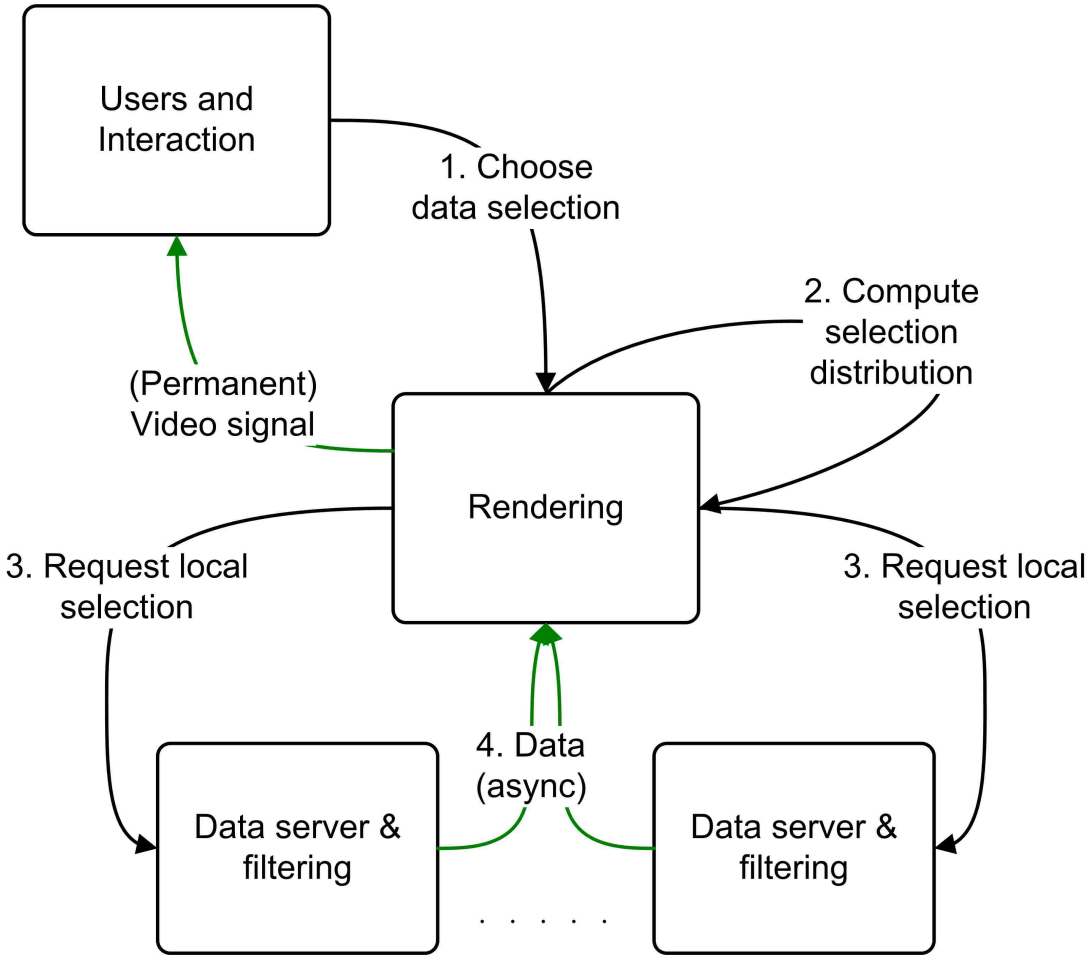

FIGURE 2.6: Remote data access in the *eaviv* visualization system

User interactions that do not require any modifications to the visualization data, such as viewpoint changes (rotations) do not trigger any communication between the visualization client and the data servers.

2.2.2 Rendering

The second component of the system is rendering, the visualization pipeline stage that transforms raw data into images.

As discussed in the results section (2.3.4.1), because of fundamental scalability issues of parallel rendering even the capabilities of the most powerful graphics clusters today will not be able to interactively render data in the terabyte or petabyte range, so in *eaviv* only particular sections of interest (for example a single time-step) that can be rendered interactively are transferred to the rendering machine (and not the entire entire file at once). When the data is received, the visualization is updated and the user can move to another section of interest, interactively exploring the dataset [PHKH04].

Loading data into the application is a lengthy process, particularly for large data sizes. To address this, progressive visualization was implemented in *eaviv*. Each data request is split into multiple smaller requests so that when a subset of data has been transferred the visualization can already be updated, and the user does not have to wait for the data object to be transferred completely before seeing an update to the visualization. This approach is illustrated in Figure 2.7.

The number of updates that can be handled by the visualization application is limited by the rendering frame rate.

2.2.3 Video Streaming

The third component of the system is video streaming. Images generated by the remote rendering process need to be transported to the local client for viewing by the user.

High-speed network connections used in conjunction with the remote rendering architecture presented in Section 2.1.2 also enable collaborative visualization at high resolution, high-frame rate and no compression. By using video distribution, each user of the system can receive a copy of the video stream and by using appropriate interaction mechanisms (described in Section 2.2.4) the users can steer and control the visualization process.

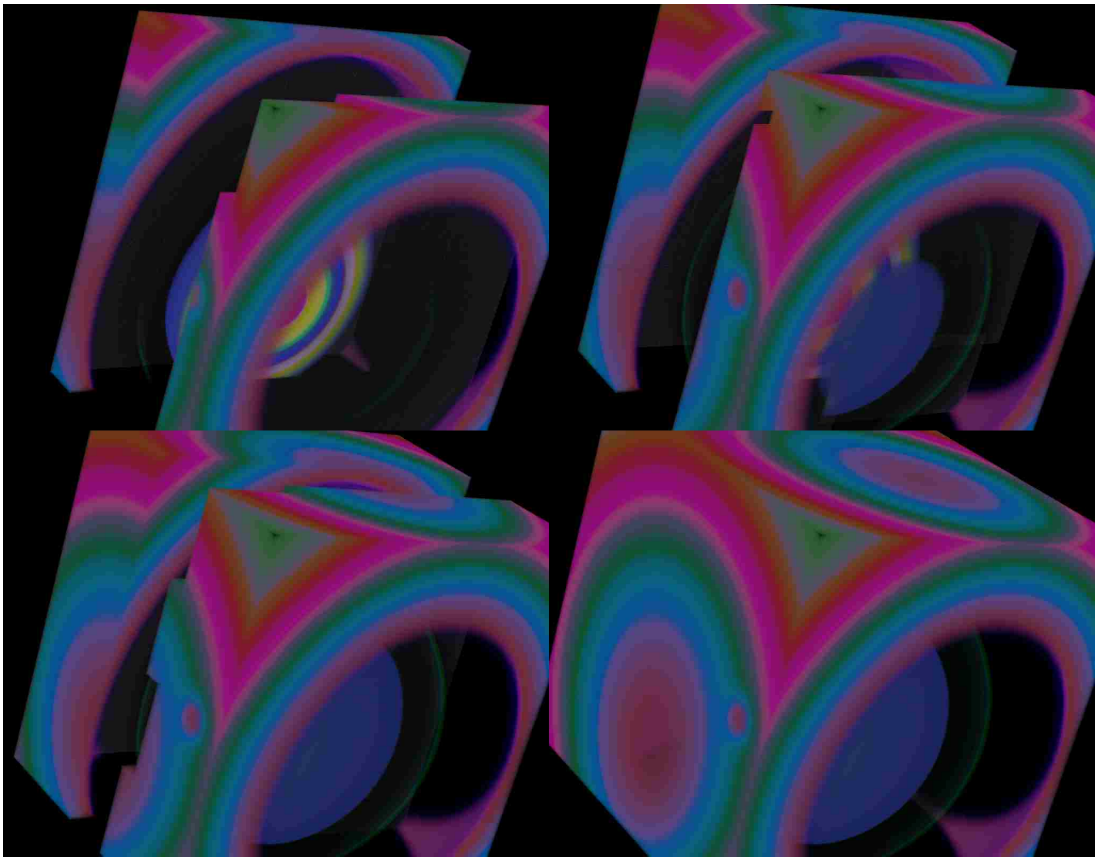

FIGURE 2.7: Four stages in the progressive visualization process of a single dataset

Compression of video streams is useful for reducing the data rate [BPS+03] but it comes at the cost of inducing additional latency and having to deal with the issue of quality degradation due to data loss in the network. Using low resolution videoconferencing technologies may require separation of the interactive and collaborative part from the high-resolution visualization [KPB+03].

The solution adopted for *eaviv* is to use uncompressed video transmission. While having higher bandwidth requirements, uncompressed video transmission supports high quality video transmission as well as low latency in the interaction loop, an important feature for interactive visualization.

2.2.4 Interaction

As the rendering is separated from the user, a remote interaction system is necessary for the user to connect to and steer the visualization. Interaction with the remote renderer is used to modify visualization parameters such as the viewing direction or the level of zoom and to move between different regions of interest in the data.

For interaction, in the initial stages of developing *eaviv*, we saw how remote mouse control (e.g., via the Synergy[2] program) can grow practically unusable over high-latency (> 1 second) image-streaming pipes. Even with lower latency, there are major practical challenges in allowing numerous users to collaboratively manipulate a shared visualization via mouse-based interaction. In response, we made experimental use of specialized physical interaction devices called "viz tangibles". These devices [USJ+08], developed by the Tangible Interaction group at CCT[3] support both local and remote collaborative use, providing a key functionality enabling collaborative use of *eaviv*.

2.2.5 Deployment (Grid Computing and Co-Allocation)

The final issue concerning the *eaviv* system is the system deployment and execution. Regarding the data servers, there is an option to execute them as always running services on remote machines, however if the data selection/filtering operations are non-trivial (i.e. they are CPU intensive) or if any type of caching is used on the server side, as it is the case for *eaviv* the clean solution adopted in the *eaviv* system is to schedule their execution as a regular job.

To execute the distributed visualization application, a component is needed that can co-allocate the required compute and network resources. To this end, the HARC [Mac07] (The Highly-Available Resource Co-allocator) framework was utilized. HARC is a system that is able to reliably co-allocate both network and compute resources.

HARC uses the advance reservation mechanisms provided by the local compute schedulers to co-allocate nodes across all the compute resources that are used by the application. HARC also implements a basic network resource allocation module that uses underlying network provisioning mechanisms. After co-allocation, the application is initiated by submitting compute jobs to the advance reservations created by HARC, to be executed at the time when the reservations start.

2.3 Results and Discussion

This section describes details of two visualization systems designed using the proposed architecture: an early prototype termed the "iGrid 2005 system", and the current *eaviv* system. The iGrid system

[2] `http://synergy2.sourceforge.net/`
[3] devices integrated in *eaviv* by Cornelius Toole

includes video conferencing and integrated use of grid deployment technologies while the *eaviv* system supports parallel rendering on GPU clusters. Results of each system are presented and discussed.

2.3.1 iGrid 2005 System

The first experiments using distributed resources and high-speed networks to improve the performance and features of a visualization application were performed during the iGrid 2005 international conference [iGr05] and later at the Supercomputing 2005 conference.

The iGrid experiment used a three-way distribution of the visualization pipeline: data, rendering and display by combining the remote data access and remote rendering methods described in Sections 2.1.1 and 2.1.2.

As described in Section 2.2.1, for remote data access, compute resources connected with high-speed networks to the rendering machine were used to cache data in their main memory to improve the data transfer speed of the application.

To limit the effect of latency on the visualization system, a remote data access system that separates data selection from data transport was used [KPHH05, PH05]. This allowed pipelining and asynchronous execution and reduced the overhead of executing a remote data access operation to a maximum of one network Round-Trip Time (or RTT). The iGrid system however did not support high-speed transport protocols, and used standard TCP for data transport.

Image streaming was handled in the iGrid system [HAB+06] (see Figure 2.8) by using video and audio transport software and hardware to connect three sites in a video conference session. One site (LSU) served as the host for the visualization application, whose output (rendered images) was directly connected to the video conference using specialized video conversion hardware (Doremi XDVI 20s). A solution based on uncompressed high-definition video [HML+06] was used for video transport.

The videoconferencing system that was used captures video with full 1080i resolution (1920×1080 image, 60 fps interlaced) and sends the data over the network resulting in a bandwidth requirement of 1.5 Gbps per each video stream. For three video streams (one visualization, two video conference) this totals 4.5 Gbps required at each of the participating sites. The video data was distributed to the other participating sites using UDP packet reflector technology [HHD04]. The total bandwidth capacity required by this setup is equal to the number of participants $\times$ number of video streams $\times$

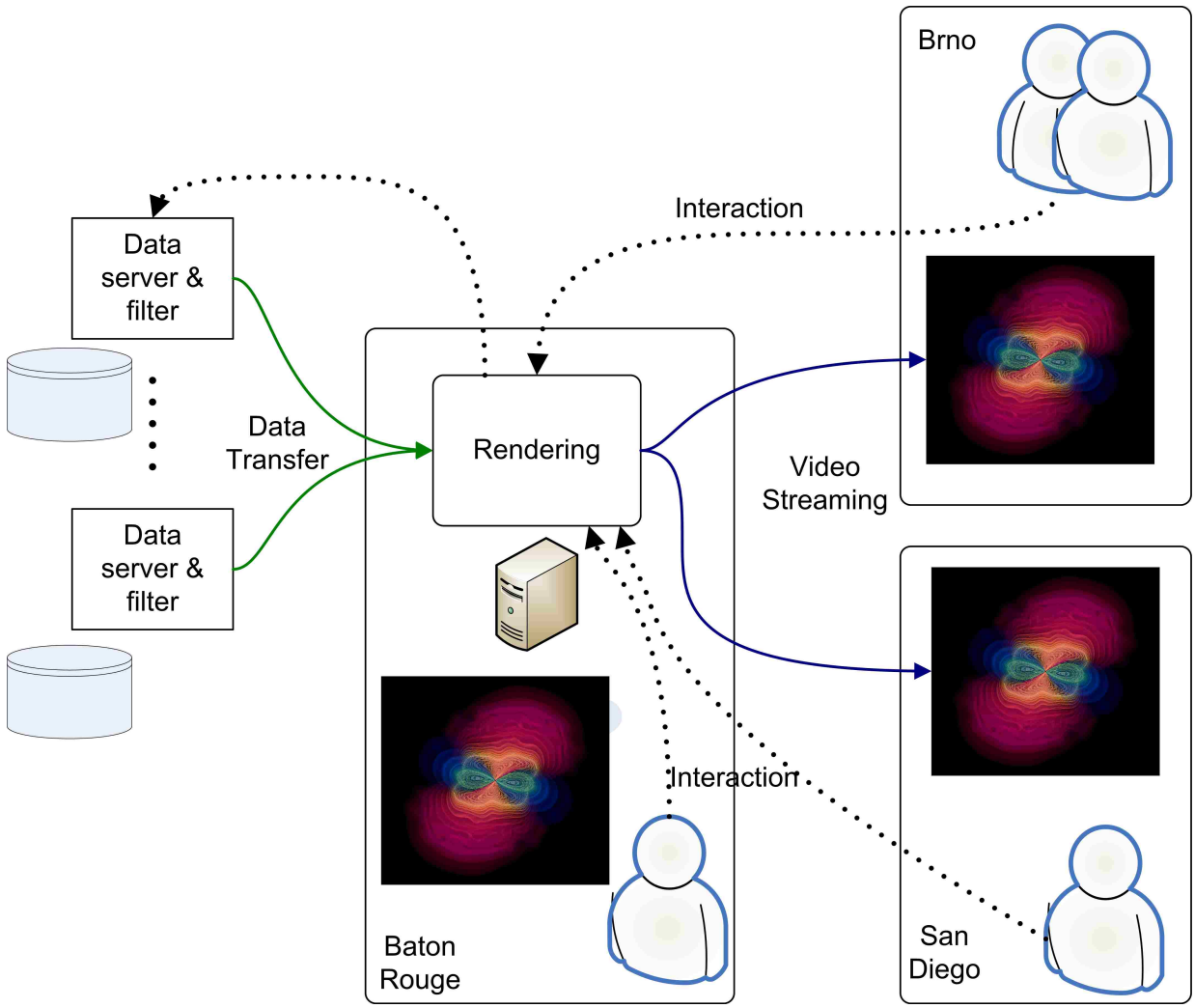

FIGURE 2.8: Illustration of the *eaviv* architecture used for the visualization server-based collaborative environment at iGrid 2005 (Brno, Baton Rouge, and San Diego)

1.5 Gbps. For three sites and three video streams this adds up to 13.5 Gbps network usage showing the need of high-speed networks to support high-quality collaborative visualization.

Interaction devices were deployed at all sites and, together with high-resolution uncompressed videoconferencing, allowed to add high-quality remote and collaborative capabilities to the visualization application.

For the rendering component, the Amira visualization package running on a single visualization workstation was used. Parallel rendering was not supported for the iGrid experiment and progressive rendering was not implemented.

Regarding deployment, HARC was utilized in the iGrid system to co-allocate the compute resources needed to execute the remote data servers. After successful co-allocation, the Grid Application Toolkit

(GAT) [ADG+05], which provides a simple generic job-submission interface, was used to submit the jobs to the compute resource reservations, through the Globus GRAM resource management system and the PBSPro compute scheduler.

2.3.2 iGrid 2005 Results

The distributed visualization system showed that using a pool of networked memory can improve the data transfer rate of the visualization application. The measurements showed a reduction in load time from over 5 seconds when using a single locally-mounted file system to 1.4–1.5 seconds per timestep (256 Mbyte timesteps) when using the distributed cache. The end-to-end bandwidth observed by the application (including network transfer, data request, endian conversions) was approximately 1.2 Gbps.

The latency induced by the video system was approximately 200 ms. Even with network round-trip times of up to 150 ms for the transatlantic connection to Brno (aerial distance between Baton Rouge and Brno is approximately 5400 miles, network path distance is longer) the distributed collaborative environment remained interactive.

A visualization application meeting many of the requirements described at the beginning of the chapter (high frame rate, good response time, high resolution, good video quality and collaborative visualization) was thus created.

The remaining issue is to increase the size of data that can be visualized, where the iGrid system is still limited by the rendering capacity of a single visualization workstation and, indirectly, also by network protocol transport performance. Also, using a fixed-resolution videoconferencing system, as that used by the iGrid system to enable collaborative visualization is an effective, convenient method of adding collaboration capabilities to an existing visualization system however it is limited in resolution by the video capture and conversion equipment.

2.3.3 *eaviv* System

Building on the previous experience, the current *eaviv* system retains and enhances many of the features used in the iGrid experiment such as: pipeline remote data access, using distributed compute resources to improve data transfer speed, tangible interaction and uncompressed video transmission.

A major improvement over the iGrid system is the integration of parallel rendering methods in the visualization system to take advantage of powerful graphics clusters. A second improvement is the integration of progressive rendering methods.

2.3.3.1 Parallel Rendering

eaviv uses a GPU-based parallel volume renderer[4]. Each node renders a portion of data and the resulting images are collected and composited together for a single 3D view of the complete dataset.

2.3.3.2 Progressive Rendering

The rendering system has been designed to support progressive visualization, as data transfer takes place the 3D volume texture (stored on the graphics card) is continuously and asynchronously updated allowing the user to continue to interact with the visualization while the data is loading.

2.3.3.3 Video Streaming

In *eaviv*, the images generated by the rendering nodes are streamed to the viewer using the SAGE software-based video streaming system.

SAGE supports both TCP and UDP-based video streaming, making it suitable for high-speed long-distance networks as well as high-resolution and tiled displays. Excepting network latency, this system supports a video quality similar to that produced by a local rendering system.

Each rendering process uses SAGE to stream its section of the final image directly to the viewing client(s), avoiding the bottleneck of sending images through a single node in the parallel rendering system. This improves rendering performance and enables the system to use parallel network links for a higher video throughput. SAGE also supports video distribution to multiple users using SAGE bridges, thus providing an essential feature supporting collaborative visualization.

The *eaviv* system has been designed to support collaborative visualization, however this feature has not yet been tested. Regarding deployment, the same co-allocation and remote execution mechanisms used at iGrid (HARC and Globus) can be used for *eaviv*, however this remains to be tested in the future.

[4]Implemented by Jinghua Ge

2.3.4 *eaviv* Results

eaviv's performance was tested and compared using a sample dataset with a resolution of 4096^3 bytes with a total size of 64 GB (generated by Cactus [GAL+03]).

The first benchmark analyzes rendering speed and I/O speed as the data size and number of rendering processes is increased. A second benchmark analyzes the relationship between rendered image resolution and rendering speed. The final analysis compares *eaviv* with two other distributed visualization systems: ParaView and VisIt.

The benchmarks were performed on an 8-node visualization cluster, each node having two Quad-core Intel Xeon E5430 processors (2.66 GHz), 16 GB RAM, 1 Gbps network interface cards. The system has four NVidia Tesla S1070 graphic units. Each Tesla contains 4 GPUs, has 16 GB video memory and services two rendering nodes, each node thus having access to two GPU units and 8 GB video memory. The disk system of the cluster is a 22 Terabyte RAID array connected to the rendering nodes via NFS. The cluster interconnect is 4x Infiniband and the software was compiled and executed using MPICH2, version 1.1.1p1 using IP emulation over Infiniband.

2.3.4.1 Data Throughput and Rendering Scalability

The rendering frame rate was measured and local throughput was compared with remote (network) throughput for three scenarios: rendering 15 GB data using 8 processes, rendering 30 GB data using 16 processes (two processes per node), and rendering 60 GB data using 32 processes (four processes per node, two processes per GPU).

The network data servers were deployed on two LONI clusters, using up to 32 distributed compute nodes to store data in the main memory. The network protocol used for data transfer was UDT.

For reference, the performance of the system when running on a single workstation was measured (workstation specifications: Intel Core2 CPU X6800, 2.93 GHz, 4 GB RAM, graphics: GeForce 8800 GTX, 1 GB video memory, 1 Gbps network interface). The rendering resolution for the benchmark is 1024x800 pixels.

The results are shown in Table 2.1. We can see that as we increase the number of rendering processes we can render more data, however the frame rate is decreasing. At 32 processes the system is not able to achieve more than 5 frames per second (requirement for interactivity). This reduction

in speed is expected because the communication overhead increases with the number of processes. The effect is a reduction in frame rate, showing a fundamental issue with parallel rendering: at some point as the data size (and thus number of processes required to render it) increases, the frame rate drops to a level below the point of interactivity. However the results show that the system is able to utilize the rendering cluster to interactively render 35 times more data than a typical workstation, and maintain an acceptable level of interactivity while rendering more than 70 times more data than on the workstation.

TABLE 2.1: Data throughput and rendering scalability results.

# processes	Data size	Frame rate (fps)	Local speed	Network speed
1 (workstation)	0.8 GB	30	0.68 Gbps	0.8 Gbps
8 (cluster)	15 GB	15-21 (18 avg)	0.11 Gbps	6.6 Gbps
16 (cluster)	30 GB	11-13 (12 avg)	0.12 Gbps	5.3 Gbps
32 (cluster)	60 GB	4-5 (4.5 avg)	0.2 Gbps	4.3 Gbps

Regarding data speed, we see a big advantage when using network I/O on the cluster, proving the value of the proposed approach of designing the system to be able to take advantage of high-speed networks. The system achieves 6.6 Gbps throughput over the LONI wide-area network (the limit being the network interfaces on the cluster) when using 8 processes. As we increase the number of processes the network speed decreases slightly because of the increased contention on the network interface on the same node (analyzed in Chapter 4). The difference in speed on the workstation is not as large, because of the lower network capacity of the workstation, and because the visualization cluster disk system is relatively slow.

The results show that the system is able to successfully sustain interactive frame rates when rendering 30 GB data and nearly interactive frame rates for 60 GB as well as achieve high data transfer rates when using the LONI network.

2.3.4.2 Resolution Scalability and Video Streaming Needs

The performance of the system as the rendering resolution is increased was measured. The important result is the the bandwidth requirement for video streaming which is computed by multiplying the number of pixels by the frame rate and by 24 bits of color information per pixel. This gives

an indication of the network service needed to support remote visualization using the system, without having to reduce the video quality or reduce the frame rate. The rendering speed for different screen size configurations was measured by running experiments on the 8-node cluster for a 2 GB dataset using 8 rendering processes. The results in Table 2.2 show that the bandwidth required for video streaming is between 500-600 Mbps with a slight increase in the bandwidth requirement as the resolution is increased.

TABLE 2.2: Resolution effect on frame rate and video streaming bandwidth requirements

Resolution	Frame rate	Streaming requirements
1024x800	28 fps	525 Mbps
1920x1080	11.5 fps	546 Mbps
2048x2048	6 fps	576 Mbps

The results illustrate the need for high-speed network services to sustain the bandwidth requirements for full-quality video streaming.

2.3.4.3 Comparison with Other Systems

To better understand the features and the trade-offs of *eaviv* a comparison with alternative visualization systems was made. Two appropriate comparison systems were identified, ParaView (version 3.6.1) and VisIt (version 1.12.0).

The comparison was made in four different areas: data input; parallel rendering; video streaming and interaction, and miscellaneous items. Both qualitative and quantitative items were analyzed. Slightly different data sizes were used due to the different modes of selecting the section of interest in each system.

Starting with data input, the first feature of interest is data loading style. *eaviv* uses progressive visualization which, for data input, means that large data read operations are split into multiple smaller operations. *eaviv* also uses asynchronous data loading, so that data operations are executed in the background and do not block the visualization system. ParaView loads the entire data at once, and blocks the system while the load operation is executed. Data protocols are used to transfer data between the data servers and rendering. *eaviv* supports multiple data protocols allowing it to take advantage of high-speed networks. The benchmark executed on the rendering cluster shows how *eaviv*

TABLE 2.3: Comparison of visualization systems features and performance: I/O methods, rendering, interaction, streaming and other items

Feature	*eaviv*	ParaView	VisIt
Data loading	Progressive, Asynchronous	Single operation, Blocking	Single operation, Asynchronous
Data protocols	UDT, TCP, fully configurable	TCP only	TCP only
Data servers	Distributed and parallel	Parallel only (MPI)	Parallel only, must be on same cluster
Data throughput	*5.3 Gbps Network (30 GB data)*	*0.12 Gbps Local (32 GB data)*	*0.12 Gbps Local (32 GB data)*
High-speed data limit	*Yes: Main memory*	*No: Disk size*	*No: Disk size*
Parallel volume rendering	GPU	CPU	CPU
Frame rate	*11-12 fps (30 GB)*	*0.5-1 fps (32 GB)*	*0.28-0.35 fps (32 GB)*
Render size limit	*60 GB (GPU memory)*	*120 GB (CPU memory)*	*120 GB (CPU memory)*
Time to first image	*5 s*	*35 minutes (load time 30 GB)*	*35 minutes (load time 30 GB)*
Visualization updates	*0.1 s (frame rate)*	*N.A.*	*N.A.*
Video streaming	Parallel (SAGE)	Serial	Serial
Video transmission	TCP, UDP (SAGE)	TCP only	TCP only
Interaction	Tangible devices (mouse & keyboard with VNC)	Mouse & keyboard	Mouse & keyboard
Collaborative support	Yes: SAGE video distribution, tangible devices	No	No
Direct simulation connectivity	No	No	Yes
Fully-featured visualization application	No (Prototype)	Yes	Yes
Programming effort	High	Lower	Lower
Execution complexity	High	Low	Low

can take advantage of the high-speed network to achieve a high data throughput. This throughput can however only be sustained for an amount of data equal to the main memory size available in the network. Both ParaView and VisIt throughput is limited by disk speed.

The second area of interest is the parallel rendering component. *eaviv* uses a GPU-based parallel renderer, allowing it to take advantage of graphics acceleration for volume rendering and enabling high frame rate. ParaView and VisIt do not currently support parallel GPU acceleration, and in consequence the frame rate that they can achieve is below 1 frame per second. For VisIt the ray-casting

parallel rendering method was used for comparison. GPU-based rendering is however limited in the data size that it can render by the amount of video memory of the graphics cards. CPU-based rendering can usually render more data, as the amount of main memory in a system is generally higher than that of video memory. The benefits of progressive visualization are clearly seen in the time needed for the system to produce the first visual image which for the *eaviv* system, when being set-up to render 30 GB of data is only about 5 seconds, equaling the set-up time and the time needed for the first data to be transferred. The visualized data is updated continuously as data is being loaded into the system, at the update rate limited by the rendering frame rate. ParaView and VisIt do not support progressive visualization, so the wait time for the first image and each other data update is equal to the data load time (35 minutes).

Parallel video streaming is a feature supported by *eaviv*'s use of the SAGE system. Each rendering node, after generating a section of the final image can transmit it directly to the viewer client. In contrast, VisIt and ParaView rendering processes transmit their results first to the master node which combines them and transmits the complete image to the client. Serial video streaming introduces additional overhead and latency into the system. The video transmission protocols influence the video throughput, and support for UDP video streaming is essential for using the system in wide-area networks. The *eaviv* prototype has integrated support for tangible interaction devices while allowing mouse and keyboard interaction through the use of third-party software, such as VNC. The use of SAGE and tangible interaction devices enables direct support of collaborative visualization, where multiple users, potentially at different locations around the world can simultaneously interact and collaborate using the visualization system. SAGE bridges can be used to multicast the video stream from the application to multiple users, and interaction devices deployed at each user location can be used to interact with the visualization.

One of the important missing features of *eaviv* is the current lack of support for direct connectivity to simulations, in order to visualize live data, as it is being generated (this feature is already supported by VisIt). *eaviv* is designed as a prototype to explore the possibilities of network-aware distributed visualization, it only supports volume rendering of uniform scalar data, and has only a small fraction of the features of complete visualization systems such as VisIt and ParaView.

The programming effort to implement the *eaviv* system to take advantage of high-speed networks, asynchronous visualization updates, GPU rendering and specialized interaction devices is considerable and starting and running the system, with its dependence on distributed grid resources, networks and specialized hardware is a complex process. VisIt and ParaView are thus much easier to use. As grids, network services and software matures this process should become simpler and accessible for a wide range of users.

2.4 Conclusions and Future Work

This chapter has shown how the proposed integrated approach in designing visualization applications helps to decrease the gap between the ever increasing data sizes generated by scientific applications and the data handling capabilities of interactive visualization systems. *eaviv* can take advantage of parallel rendering clusters to increase the amount of data that is visualized while keeping the interaction quality to an acceptable level. *eaviv* can use remote parallel rendering without reduction in video quality using high-speed networks and the SAGE streaming system.

eaviv can take advantage of high-speed networks to improve its data throughput and reduce data load time. At the same time it uses techniques such as progressive visualization and asynchronous data loading to improve the user experience when dealing with large datasets. Using tangible interaction devices and video distribution supports collaborative use by distributed users. An important next step is to conduct an experiment where *eaviv* is used for collaborative visualization.

The *eaviv* system is a prototype that shows visible performance improvements over existing, more complex distributed visualization systems; potentially guiding the development approach and directions of existing and future visualization systems. Although past experience using the *eaviv* architecture in existing systems such as Amira and Equalizer have shown that fundamental issues may appear that are hard to solve, it may be possible to integrate the *eaviv* system architecture in existing visualization systems such as ParaView and Visit.

The path for further scaling the size of the data rendered interactively will not likely be to increase the number of rendering nodes as that may decrease the frame rate below an acceptable rate for interaction. To further increase the data size a system that uses multiple loosely synchronized 3D

views of different sections of the data (as opposed to a single 3D view of the complete dataset) or of different datasets is a possible future solution. Multiple clusters at different locations in the network can be used to support such a scenario.

An important question when instantiating the distributed visualization application is, assuming that a selection of resources is possible, what resources should be selected, and how should the visualization application be instantiated. This is a part of future work, possible ideas being discussed in Chapter 6.

A particular characteristic of the distributed visualization application is that it requires coordinated, parallel use of multiple resources. A system that allocates all these resources for executing a single application is needed. If only the network resources, the data servers, or the visualization resources are allocated alone the application cannot be executed. Reintegrating use of co-allocation mechanisms in *eaviv* remains an important step for the future.

Chapter 3
Remote Data Access

An important component of distributed, network-based applications is data communication between application modules.

The work described in this chapter was mainly motivated by the *eaviv* application, described in detail in Chapter 2. In the *eaviv* architecture the rendering component uses remote data access to connect to the data servers (see Figure 2.5).

In the *eaviv* visualization scenario the user selects a region of interest to be visualized, and the rendering component splits the data request that describes this region and distributes it to the data servers (See section 2.3.1). The data is then progressively rendered by each node as it is continuously transferred over the network. The visualization system defines the requirements for the remote data access system: high speed, high operations throughput, non-blocking execution and ease of configuration, as follows.

- **Speed**.

  The main requirement of the remote data access system is that data needs to be transferred as fast as possible from the data servers to the rendering processes. As described later in Chapter 4, in order to achieve maximum speed the system needs to be aware of the network situation and use the appropriate data transfer system for the particular network conditions in which it is executed. To meet this requirement the following two design principles are proposed. First, the remote data access system should support parallelism: a single data consumer (or in the case of the visualization application a single rendering process) should be able to retrieve data from multiple data servers in parallel. Second, the remote data access system should support high-speed data transport protocols. As will be shown, on high-speed optical networks, the standard TCP protocol is not suitable for fast data transfer so the remote data access system needs to be able to use other, more suitable protocols.

- **Operations Throughput**.

  To improve interactiveness, the visualization application uses progressive rendering. This is achieved by splitting large data transfer operations into multiple smaller operations, so that when a subset of data has been transferred the visualization can already be updated, and the user does not have to wait for the data object to be transferred completely before seeing an update in his visualization. An important question is how many (sub-)operations should remote data access operations be split into. The answer to this question, for the visualization scenario is: as many operations as the visualization application can handle. The number of operations, or updates that can be handled by the visualization application is limited by the rendering frame rate. For example, if the visualization application is working at 100 frames per second, the number of data updates should also be 100 each second (meaning the remote data access system will need to support 100 remote data operations per second)[1].

  The *eaviv* system, depending on the rendered data size achieves a frame rate of 5–30 frames per second. In general, a rate of tens of frames per second for a visualization system is a reasonable generalization. This means that the remote data access systems needs to support the execution of a minimum of tens of operations per second. For generality to other applications however, the remote data access system should be designed to achieve the highest possible operations throughput. The following section (Section 3.1) analyzes and compares implementation architectures for remote data access systems focusing on operations throughput.

- **Non-Blocking, Non-Serialized Execution**.

  When multiple remote data operations are executed, for example in the visualization scenario where the visualization needs to be updated as often as possible, if blocking or serialized I/O is used, network latencies are added together, thus limiting the overall performance of the remote I/O system. For example, on a 250 ms RTT, 10 Gbps network link, using serialized I/O at most four remote operations can be executed each second. If the operations have a size

[1] If we define a remote data access operation as that of transferring a contiguous sequence of bytes from a remote data object, it should be noted that sometimes, for example if every second byte of a dataset is requested the data requests of the visualization application are composed of multiple separate sequences of bytes. In effect, single visualization requests can be considered to already be defined as multiple remote data access operations thus further increasing the number of operations that the system needs to support. Section 3.1 will show that combining multiple operations into a single one is an effective method of increasing operation throughput

of 1 byte, the resulted throughput is limited to 4 bytes per second, independent of transport protocol performance. Clearly, one solution to this is to increase the operation size. However, a system that serializes operations creates an artificial upper limit on the achievable throughput because serialized execution introduces breaks in network transfer every time the client sends its requests to the server (during that time the server does not transmit any data to the client). The goal is to eliminate this limit, and to design a system that enables high-throughput by allowing multiple operations to be executed at the same time. For quality interaction it is also important that the visualization does not block while I/O operations are executed. The system needs to remain responsive to user interaction even when data transfer operations are executed. To achieve this the proposed remote data access supports non-blocking, non-serialized execution of operations.

- **Configurability**.

  A visualization system should support a variety of file formats and data models, as well as various modes of interaction defined by the user. A variety of data access patterns need to be supported, the choice of which depends on the type of analysis required and the type of data involved. For example, the visualization application may request the transfer of different sections of the remote dataset to the rendering process. To support this, a remote data access mechanism that supports the encoding of arbitrary requests is needed. A remote data access system that only supports simple file operations (where operations are defined by file offset and data size) is not suitable for the *eaviv* system.

- **Library Implementation**.

  The data is delivered to a live running visualization application and it should be possible to extend the system to support live data streaming from a running simulation this describing the final requirement for our system: the ability to integrate both the client and the server side in independent applications, thus it should be constructed as a library.

The proposed system (called *eavivdata*) while mainly designed to support the visualization application can also be used in other applications requiring configurable, high-performance and high-throughput remote data access.

This chapter describes the *eavivdata* remote data access system, starting with an analysis of implementation architectures that can support high operation throughput and continuing with the description of the *eavivdata* architecture and system evaluation showing how it enables high data transfer speed, high throughput, user-defined remote operations, non-blocking execution and parallelism while being implemented as a library, unlike any other system available today.

The next section (3.1) analyzes and compares remote data access implementation architectures, and the following section (3.2) describes and evaluates the complete remote data access system.

3.1 Implementation Architectures for Remote Data Access

Communication-intensive applications should be designed to minimize the overhead of network communication between application components. If we define a message to be an arbitrary sized unit of information used for communication between application components, then the communication overhead will be larger when an application uses a large number of messages, large message sizes or both.

This section analyzes methods of reducing the communication overhead incurred when an application uses a large number of paired request/response messages, or more specifically a large number of remote data access operations as is the case of the motivating distributed visualization application.

Network latency is ofter overlooked when designing a distributed system although it is an important factor influencing application performance, in particular when executed over wide-area networks [Smi09]. This section is focused on analyzing implementation architectures that help reduce the damaging effect of network latency on application performance (and because network latency can not be reduced this reduction is named "latency hiding") and increase the operations throughput, or the number of executed operations in a given unit of time.

We look at five remote data access architectures: a simple synchronous architecture included for reference and four existing approaches for reducing the effect of network latency/increasing throughput:

a threaded architecture which creates a new thread to execute each operation; a pool architecture that uses a fixed number of threads; a bulk architecture which bundles multiple operations together in a single remote operation; and a pipeline architecture which executes remote operations in a pipeline-parallel mode.

This section briefly introduces the various programming architectures used by RPC (remote procedure call) systems to reduce the effect of network latency and shows that there currently appears to be no clear understanding on which one of them should be preferred over the others. RPC is used as a starting point because remote data access operations are seen as special types of RPC operations.

Next, the relevant implementation architectures are described in detail starting with the synchronous architecture. It is generally agreed that for communication-intensive applications, computation and communication should be overlapped whenever possible and as the visualization scenario also requires non-blocking execution the three architectures supporting non-blocking execution (threaded, pool and pipeline) are described next. Another method for reducing communication time is to combine multiple operations into a single remote operation. This mechanism is included in the analysis as a fourth architecture (bulk).

The various architectures were implemented and their performance measured on networks of both low and of high latency. These measurements were made on deterministic, dedicated networks. Such a network infrastructure is required for deterministic results. The parameters and technical requirements of each architecture are analyzed and general properties that guide the design of the *eavivdata* system are identified.

3.1.1 Related Work

Much work has been carried out in reducing the overhead and latency of one-way message passing and the latency of a single remote call [BALL89, vECGS92]. The work presented here evaluates methods of reducing the effect of network latency for multiple remote operations. This is referred to by many authors as obtaining "high-throughput" as opposed to obtaining "low-latency". This work actually shows that these two goals are conflicting as the methods that increase total throughput when executing a large number of operations do so at the cost of increasing the latency of each individual operation.

RPC implementations appeared around twenty years ago [BN84] and since then, with CORBA [OMG91] and more recently Web Services (SOAP) [BEK+00] architectures, RPC can be used in the abstraction of computer architectures, operating systems and programming languages.

The synchronous nature of the RPC model was recognized as a major limitation soon after its introduction and much work has been done trying to overcome this by using asynchronous semantics. A notable early adopter of asynchronous RPC is the X Window system [SG86].

Multilisp [RHH85] introduced the "future" concept of asynchronous method evaluation. "Promises" [LS88] and "futures" [WFN90] extended the concept to distributed systems and asynchronous remote procedure calls. The early implementations used heavy-weight processes, but later the multithreaded model proved its value for improving the performance of communicating processes [FM92] and for hiding communication latency [BR92] and was adopted by some RPC implementations.

One method of reducing the overhead in the thread creation for each remote operation is to use thread pools [PSH04, Sch97]. Another mechanism used to reduce the overhead associated with remote procedure calls is to aggregate multiple operations in batches or bulk operations [BL94]. Other systems, such as HTTP [NGBS+97] and CORBA [AOS+00] take advantage of the pipeline nature of the network to improve performance. This approach was also considered for SOAP-based implementations of GridRPC [SNS02].

There is no good understanding of what programming method should be utilized to address the issue of network latency. This becomes apparent when looking at some of the more recent RPC implementations and specifications.

CORBA did not include support for asynchronous method invocation until late in the standardization process, when it was introduced in such a way as to not break backwards compatibility [DSOB02]. The SOAP specification includes asynchronous method invocation [Gro03] but toolkit implementations do not generally support the advanced two-way asynchronous method scenarios. It can be argued however that SOAP was never designed to facilitate high-throughput method invocation [BD05].

From the implementations of the more recent GridRPC [SNM+02] standard: Ninf-G [TTNS04] aggregates multiple operations in a single one, and OmniRPC [SHTS01] executes RPC methods in individual threads. The initial version of the new RPC system Babel RMI (Remote Method

Invocation) did not include asynchronous invocation [KLE07] however implementing non-blocking RMI in Babel is a part of the development plan.

Although surveys of RPC and asynchronous implementations exist, no quantitative and qualitative comparison of the various implementation methods is available. This work, extending and revisiting the previously published results of Hutanu et al. [HHAM06] addresses this, providing guidelines for the development and design of *eavivdata* as well as information to aid middleware and application developers.

3.1.2 Architectures Description

In the following we refer to a client as the entity that initiates the communication, and the server as the side that responds to the communication. In this analysis a single client/server pairing is considered and for simplicity it is assumed that if there are dependencies (hazards) between operations, then none of the optimized implementation architectures (threaded, pool, bulk and pipeline) can be applied. Since read-only remote data access operations (as those used by the visualization application) do not produce hazards, this assumption does not have an influence on the conclusions of this analysis. The following benchmarks use equally sized messages and identical operations, but this can easily be generalized.

Figure 3.1 illustrates how all architectures are positioned in relation to each other. Three dimensions are used to categorize the architecture in question: the number of communication threads used to process requests; the number of remote operations initiated by the client (client operations); and the number of remote requests these client operations are encoded into when using the various architectures.

In the following the number of communication threads is considered equal to the number of communication channels[2] that are established between the client and the server. This reasoning is applied since it makes little sense for a thread to establish more than a single communication channel and if multiple threads share a communication channel their accesses need to be serialized thus effectively combining them into a single communication thread. The total number of threads serving

[2]considered broadly: some communication protocols may actually use multiple low-level communication links to implement a communication channel, for example a UDP channel for bulk data transmission and a TCP channel for control or multiple TCP channels for striped transfer. From the remote data access system perspective we can safely consider this to be a single communication channel

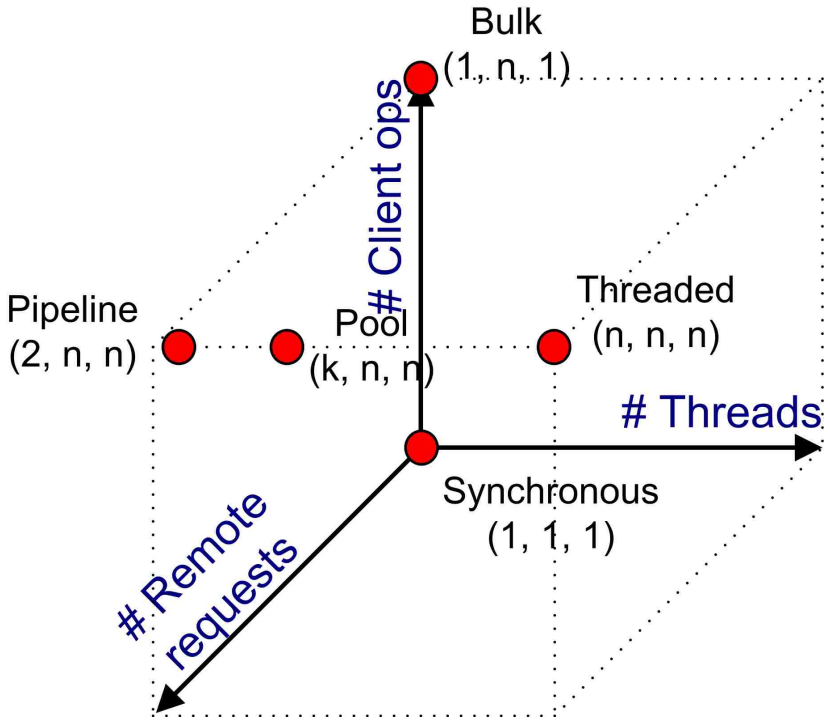

FIGURE 3.1: The five implementation architectures positioned in a phase space with axes (number of threads, number of client operations, number of remote requests).

the communication channels is actually double (each thread on the client side is precisely matched by a thread on the server side) but this is omitted for simplicity.

The synchronous architecture executes one single operation (e.g. a file access request) through one remote request using one single thread or process at any time. Hence, it is located at the intersection of the 3 axes: $(1, 1, 1)$. The bulk architecture executes n client operations through one single remote request, also using one thread: $(1, n, 1)$. The threaded architecture executes n client operations through n remote requests, invoking them in parallel, using n threads: (n, n, n). The pool architecture executes n client operations through n remote requests, invoking them in parallel using $1 \leq k \leq n$ threads: (k, n, n). The number of threads in the pool can be larger than the number of operations n, however in that case only $k = n$ threads from the pool will be activated. Finally, the pipeline architecture executes n client operations through n remote operations, using 2 communication threads (one for sending and one for receiving), its position $(2, n, n)$ is thus close to $(1, n, n)$.

The following text discusses these different approaches, outlining their requirements, advantages and drawbacks.

3.1.2.1 The Synchronous Architecture

The synchronous architecture (Figure 3.2) is, of course, not suitable for high-throughput execution as it serializes all operations.

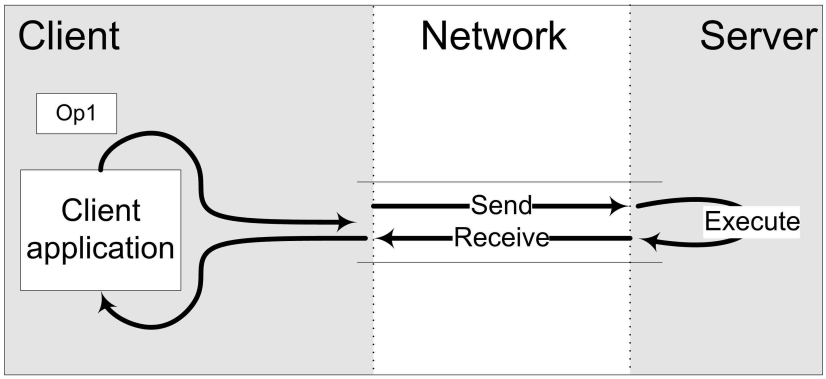

FIGURE 3.2: Synchronous implementation architecture. All operations are completely serialized

Sequential execution of operations is a common programming paradigm, utilized by all major programming languages: each operation is executed after the previous one has completed. Only one single thread of execution exists and only one operation is active at any given point of time. The advantages of this architecture are the low overhead which makes it suitable for low latency invocation of single operations and low implementation effort.

3.1.2.2 The Bulk Architecture

The bulk architecture (Figure 3.3) clusters multiple operations into one single remote invocation.

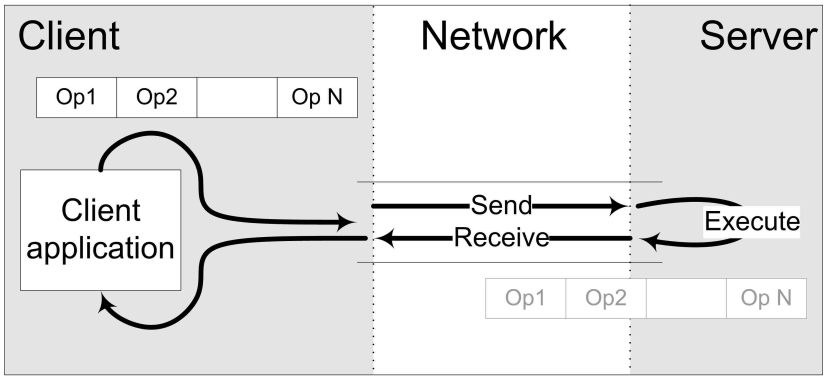

FIGURE 3.3: Bulk implementation architecture. Similar to synchronous architecture except multiple operations are processed together

These operations must be started and packed together on the client side. The implementation can ensure this by offering bulk interfaces to the application programmer, or in some cases by

automatically combining multiple remote methods into a single remote invocation [YK03]. On the server-side, bulk interfaces are required for this approach to be applicable; these are offered by various Grid middleware frameworks such as gLite [BKS05].

3.1.2.3 The Threaded Architecture

The threaded architecture (Figure 3.4) uses multithreaded execution to hide remote operation latencies: the application spawns a new thread for every remote operation (each client-side operation is executed through exactly one remote operation). Since threaded operations are non-blocking, the main application continues execution while the time-consuming remote operation is executed in a separate thread.

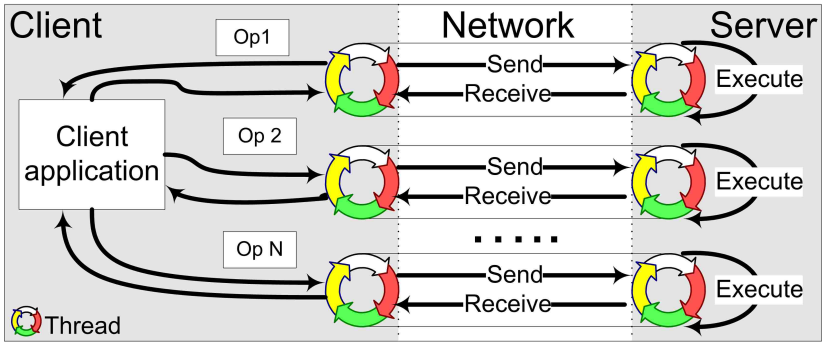

FIGURE 3.4: Threaded/pool implementation architecture. As many threads as there are operations (threaded architecture) or a fixed number of threads (pool architecture)

This architecture raises the usual multithreaded application issues, such as race conditions, deadlocks, and data (in)consistency. Application programmers must be aware that execution order is not guaranteed, unless the middleware programmer ensures operation serialization. Further, the server should be able to handle multiple requests in parallel to make this architecture work effectively [WFN90].

3.1.2.4 The Pool Architecture

The pool architecture is very similar to the threaded architecture, but with a fixed number of threads used to invoke remote operations. This significantly reduces the overhead of executing multiple operations (but can increase the overhead of executing few operations) and ensures that the system resources are not depleted. Since the total number of threads is limited this architecture also introduces

a queue of operations that are waiting for a processing thread to become available for them to be processed.

3.1.2.5 The Pipeline Architecture

The pipeline architecture (Figure 3.5) considers the client-server system to be composed of three segments: the "send command" segment (client → server), the "execution" segment on the server, and the "receive response" segment (server → client).

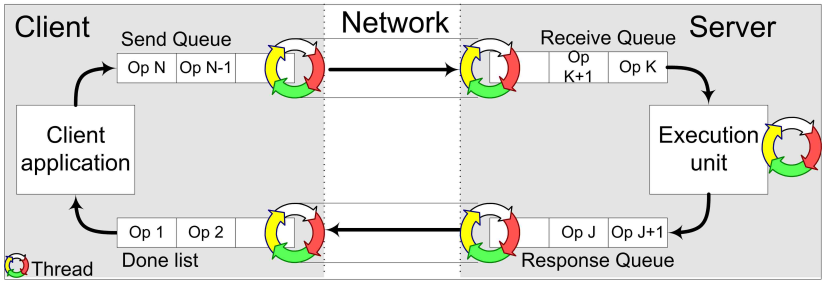

FIGURE 3.5: Pipeline implementation architecture. Showing threads for sending and receiving data and a separate thread for server-side processing

All the operations initiated on the client are queued and sent to the server by a dedicated thread over a single persistent connection. On the server side, a receiving thread queues the incoming requests for processing by the execution unit, which in turn adds its results to the response queue, to be sent back to the client. Another thread on the client side receives the results and notifies the client application.

This architecture has complex requirements for both the client and server implementation. On the client side, similar to the threaded and pool architectures, the application must support a non-blocking execution model and be able to allow multiple operations to be executed in parallel. This architecture also prescribes the server implementation details, ordering of operations is, however, implicitly preserved, with no additional overhead.

3.1.3 Benchmarks and Performance Analysis

A series of experiments were performed to compare the different architectures and understand how various factors influence execution time.

All benchmarks use self-implemented middleware emulations using the five execution architectures. The significant portions of the code (C++) use the boost thread library, the BSD sockets

implementation and the UDT library. The operations used for these benchmarks are simple file access operations (defined by file offset and read size), suitable for the analysis of the implementation architectures. The benchmarks were constructed to be network intensive, network being the important factor in this analysis.

The first benchmark measures the time needed to execute a fixed number of remote operations, using the various implementation architectures and when all operations are initiated at the same time. This is clearly an advantage for architectures that can optimize execution time when multiple operations are active, but it helps us gain an understanding on how effective all these architectures are by eliminating other factors from the measurements.

The second benchmark measures the time needed to execute a single operation using each architecture. This indicates the overhead added by each architecture and how this increases the latency of a single operation.

Additional benchmarks were made for individual architectures to understand some of their specific details such as the effect of changing the number of threads or changing the number of active operations on overall throughput.

3.1.3.1 Benchmark Setup

The benchmarks were performed using dual quad-core Intel(R) Xeon(R) machines, CPU frequency 2.66 GHz, 16 GB RAM located at LSU, Baton Rouge and on dual double-core AMD Opteron(tm) machines, CPU frequency 2.6 GHz, 4 GB RAM machines in Brno, Czech Republic. All machines were running Linux (kernel 2.6.*) and the code was compiled using the GNU compiler collection. The code was executed in two network environments: (1) a local fiber connection between the LSU machines: RTT (round-trip-time measured with the ping utility) < 0.1 ms, named "LAN", and (2) a long distance dedicated fiber connection between LSU and Brno: RTT 149.3 ms named "WAN".

For these benchmarks, the size of the "request" and "response" messages are equal to 24 bytes each.

TCP throughput over wide area networks is highly dependent on factors such as RTT, buffer settings on the end hosts and message size. To reduce the effect of network protocol performance variance, the performance of the system was also measured when using a fixed rate transmission

protocol implemented under the UDT library, a protocol that has less performance variance under different network conditions.

Table 3.1 and Figure 3.6 show data transfer time and computed bandwidth for various message sizes when using two different transfer protocols: TCP, and rate-based data transmission using the UDT library.

TABLE 3.1: Transfer time and computed throughput depending on message size using UDT and TCP. UDT refers to fixed rate data transmission using the UDT library (The rate was set at 3 Gbps). TCP is the standard Linux TCP implementation. Mbps = Megabits/second. 1Megabit = 1048576 bits = 131072 bytes. speed = throughput. Time is one-way transmission time and throughput is computed as size divided by time

size(bytes)	time(UDT)	speed(UDT)	time(TCP)	speed(TCP)
100000	0.003 s	254 Mbps	0.3 s	2.5 Mbps
1000000	0.019 s	401 Mbps	1.2 s	6.3 Mbps
5000000	0.085 s	448 Mbps	1.8 s	21.2 Mbps
10000000	0.12 s	635 Mbps	2.15 s	354 Mbps
50000000	0.26 s	1467 Mbps	4.39 s	869 Mbps
100000000	0.59 s	1616 Mbps	7.1 s	1074 Mbps
500000000	1.95 s	1956 Mbps	29.07 s	1312 Mbps
1000000000	3.69 s	2067 Mbps	56.9 s	1340 Mbps

The results show how the throughput increases with increased data size, and how the variance is much higher for TCP than for rate-based transmission with UDT. Also, for small data sizes the transport protocol performance is quite low meaning that for a small number of operations we should expect a lower performance than for a large number of operations.

3.1.3.2 Operations Throughput

The main goal of this analysis is to find out what implementation architecture(s) are well suited for hiding network latency and to achieve high operations throughput. Table 3.2 and Figure 3.7 show the maximum throughput that can be achieved using the different implementation architectures. The results clearly show that the bulk and pipeline architectures are those that can best increase operation throughput.

Both bulk and pipeline architecture achieve higher performance as the number of operations is increased. However, the number of operations needed to achieve maximum throughput in the bulk and pipeline cases are different. In the case shown here, for example for the bulk case in WAN

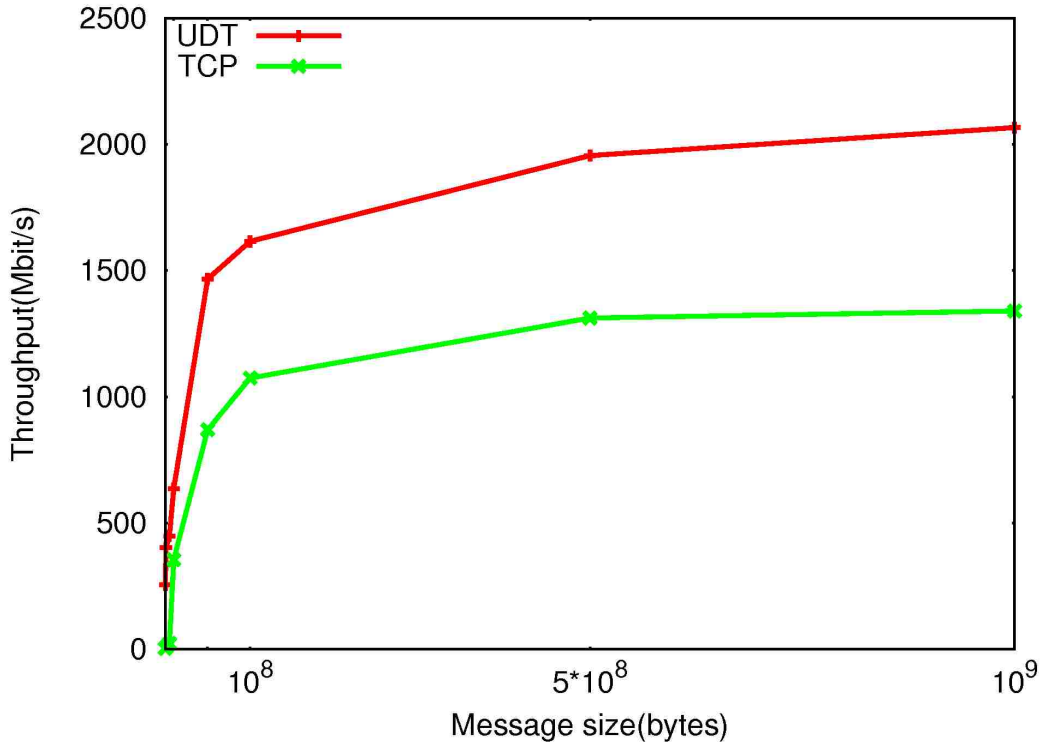

FIGURE 3.6: Computed throughput of TCP and UDT depending on message size

TABLE 3.2: Operations throughput when multiple operations are executed using the five implementation architectures. Italics numbers indicate the best performing architectures

Network	Synchronous	*Bulk*	Threaded	Pool	*Pipeline*
WAN TCP	6.6 op/s	*125000* op/s	305 op/s	3311 op/s	*125000* op/s
WAN UDT	6.6 op/s	*35714* op/s	111 op/s	3215 op/s	*32258* op/s
LAN TCP	18518 op/s	*333333* op/s	9259 op/s	83333 op/s	*125000* op/s
LAN UDT	14.6 op/s	*50000* op/s	372 op/s	27777 op/s	*40000* op/s

when using TCP, over 800000 operations are needed to reach the maximum throughput of 125000 operations per second, whereas for the pipeline about 50000 operations are sufficient to reach the same throughput.

The number of operations used for the threaded architecture in this benchmark was 100, limited to avoid thrashing (discussed in detail later), the number of operations for the pool architecture is 300000 and the number of threads in the pool is 500.

The LAN results show a higher throughput not only because of the lower network latency but also because of the slightly better performance of the LSU machines used exclusively for the LAN measurements over the Brno machines involved in the WAN benchmarks.

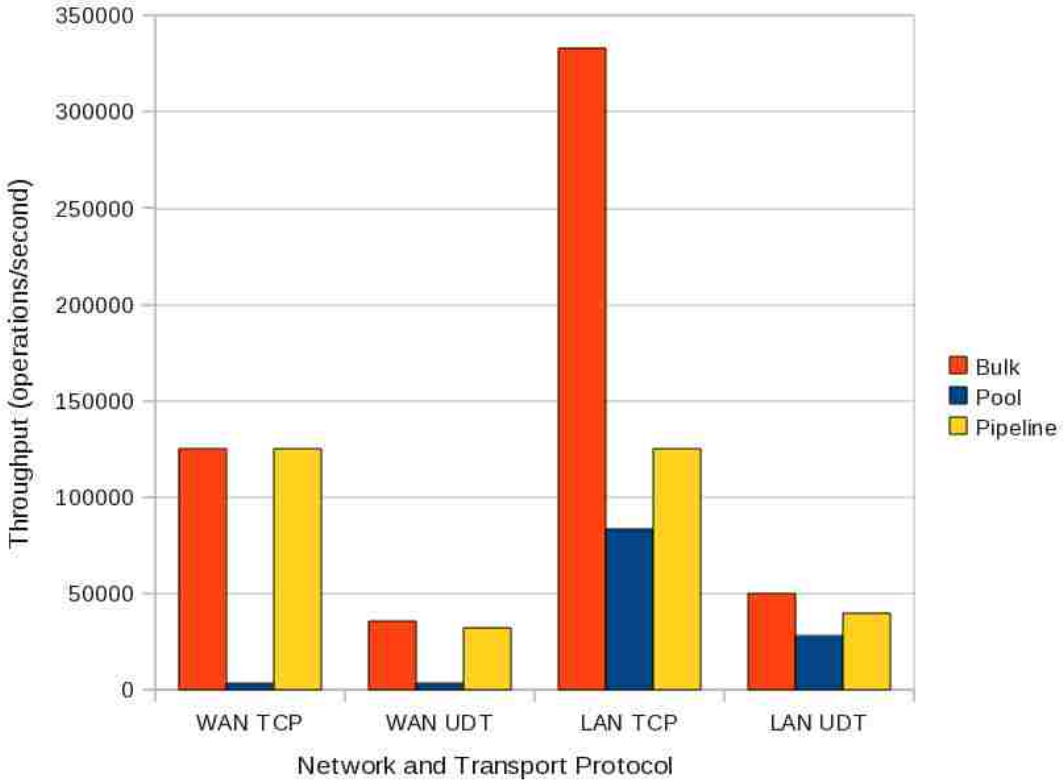

FIGURE 3.7: Maximum throughput (operations per second) when multiple operations are executed using the three implementation architectures with the highest throughput (Bulk, Pool and Pipeline). The number of operations for which this throughput is achieved is different for the three architectures

3.1.3.3 Single Operation Overhead

The second benchmark analyzes the overhead of each programming architecture when executing a single operation. The results in Table 3.3 and Figure 3.8 show that the synchronous architecture, which is not optimized for high throughput, has the lowest per-operation overhead. This is because of the complexity and additional processing steps needed by all the other architectures. The results in the WAN case are, as expected, dominated by the network latency. The LAN measurements illustrate the programming architecture overhead more clearly. For both throughput and overhead benchmarks teb measurements were made for each experiment, the results showing the average execution time. Standard deviation was not significant.

The bulk architecture has the second-lowest overhead, and the pipeline the third-lowest. The bulk architecture has a higher overhead than the synchronous architecture because of the additional encoding and decoding of the bulk operation, for example encoding the number of operations composing the bulk (needed even for a single operation, for consistency). Similar to pipelines in CPU architectures,

TABLE 3.3: Overhead per operation when a single operation is executed using the five implementation architectures. μs = microseconds. Italics indicate the architectures with the lowest overhead

Network	Synchronous	Bulk	Threaded	Pool	Pipeline
WAN TCP	151040 μs	151064 μs	302488 μs	151257 μs	151084 μs
WAN UDT	151059 μs	151854 μs	492458 μs	151344 μs	151120 μs
LAN TCP	*45 μs*	*70 μs*	199 μs	449 μs	*107 μs*
LAN UDT	*126 μs*	*161 μs*	58603 μs	3904 μs	*210 μs*

the pipeline architecture has an even higher overhead, caused by the multiple segments a single operation needs to pass through. The pool and threaded architectures have very high overheads, and in the case of the threaded architecture the overhead is extremely high because in addition to starting a new thread, a new connection needs to be established for each operation. This can easily be seen even in the network-latency dominated WAN results.

In the following we take a closer look at the individual implementation architectures, with a particular focus on two architectures of interest: bulk and pipeline, as the previous results show that these architectures have the highest operations throughput as well as the lowest overhead of the architectures that enable high throughput.

3.1.3.4 Synchronous Architecture

The synchronous approach can be used for distributed applications but has no latency hiding meaning that remote operations can easily have a damaging effect on overall application performance. On the other hand, the synchronous architecture is trivial to implement and use, and may be a valid option if remote execution performance is negligible for the overall application performance, or if concurrency constraints inhibit communication and computation to overlap.

The main advantage of the synchronous architecture is that it has the least overhead of all the five proposed systems, which makes it the best choice for applications that don't need high operation throughput, but rather require low overhead for single operations.

As all the operations are serialized, the time required to execute n identical operations is exactly n times the time needed to execute a single operation.

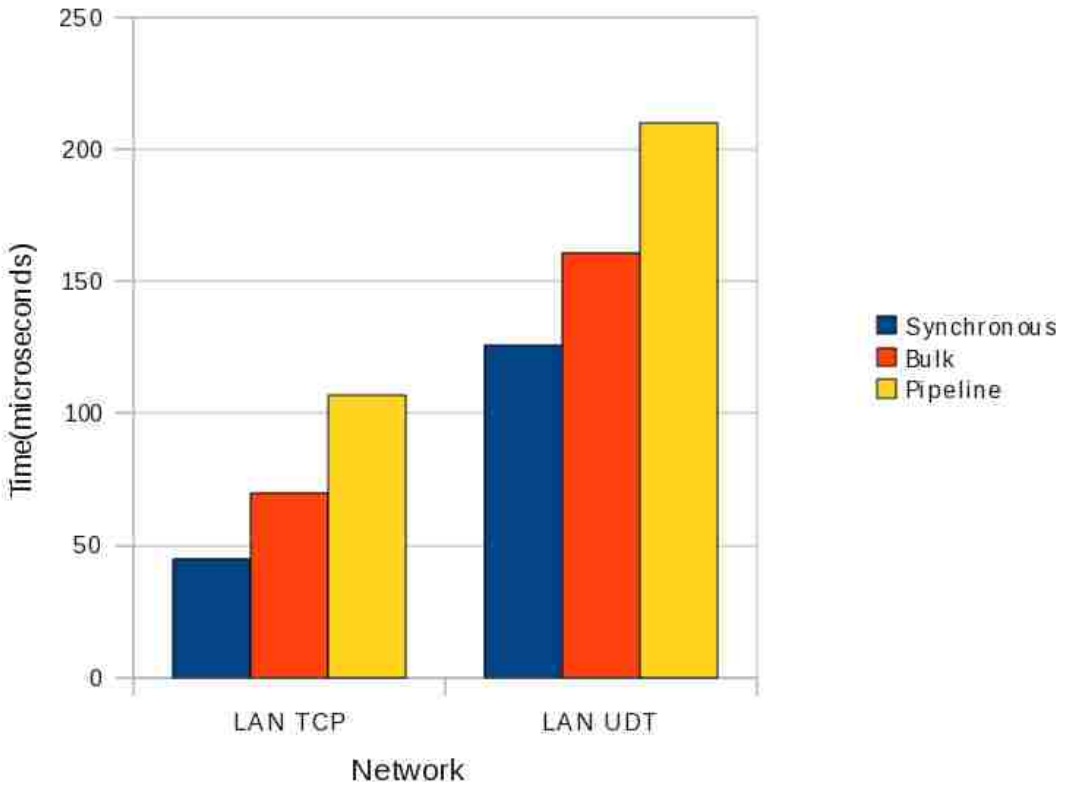

FIGURE 3.8: Overhead per operation when a single operation is executed using the three implementation architectures with the lowest overhead (Synchronous, Bulk and Pipeline)

3.1.3.5 Bulk Architecture

In the case of the bulk architecture, latency is "hidden" since it is accounted only once for multiple operations. The higher the number of operations executed, the more operations is the one-time network overhead split into thus reducing the effect of the network overhead on each operation.

The results for the LAN show that the execution time of a single bulk operation can be as low as 3 microseconds. This can be considered to be equal with the per-operation overhead of parsing and processing.

On the WAN the situation changes because execution time is dominated by the network data transfer time. Data transfer time variability on the total data size mentioned earlier becomes a factor. In consequence, the average execution time/operation depends on the number of executed operations. The higher the number of operations executed, the lower the execution time/operation.

The execution time/operation cannot be decreased indefinitely as a minimum per-operation overhead cannot be eliminated.

Performance improvements over the synchronous architecture occur mainly due to the fact that only one remote request is necessary to invoke n remote operations, thus drastically reducing overall communication latency.

3.1.3.6 Threaded Architecture

The threaded architecture adds two additional factors to the execution time. The first factor is the time needed to establish a new connection for each operation. The second added factor is the time needed to spawn a new thread. The time spent for these two additional operations is considered an integral part of the execution time of a remote operation.

Table 3.4 shows how the execution time/operation decreases as the number of operations executed at a given time is increased. However, the number of operations that are executed at any given time using this architecture must be limited. Allowing the number of spawned threads to increase indefinitely creates thrashing and the program may even crash. This happens when an excessive number of threads are started at the same time and compete for resources. In this benchmark the number of active operations was limited to 100 as further increases of the number of active operations actually degraded throughput. As shown earlier, we can see that the minimum execution time is much worse (higher) than that which is achieved by the bulk and the pipeline systems.

TABLE 3.4: Execution time/operation of the asynchronous architecture with varying number of operations active at the same time, using TCP over WAN

# operations	time per operation
1	318513 μs
2	151351 μs
3	100987 μs
4	75803 μs
5	60697 μs
10	30474 μs
30	10326 μs
50	6301 μs
70	4572 μs
100	3275 μs

3.1.3.7 Pool Architecture

In this architecture, the number of threads and communication channels is fixed, thus amortizing their set-up cost over multiple operations. This cost can be considered a part of the set-up time but it can substantial for large number of threads (see Table 3.5). This however could be optimized by allowing multiple connections to be set-up in parallel, while taking care to avoid thrashing.

TABLE 3.5: Connection set-up time for pool system

number of threads	connection time (LAN UDT)	connection time (WAN UDT)
50	0.065 s	15.15 s
500	0.841 s	151.91 s

For the pool architecture, in addition to the dependency on the number of operations, where in general the higher the number of operations, the lower the execution time/operation, the minimal execution time also depends on the number of threads in the pool.

As the number of threads increases, the execution time for each operation in the WAN decreases (see Figure 3.9), however the total number of threads needs to be limited so that system resources are not depleted. In this benchmark, it was limited to 500. The decrease in the execution time with the number of threads is attributed to the increased parallelism in executing the operations. We can see however that the parallelism of the pool system, limited by the number of threads in the pool is not as high as that of the pipeline system where the parallelism is only limited by the capacity of the network pipes.

3.1.3.8 Pipeline Architecture

In the case of the pipeline architecture it is useful to first attempt to understand its performance.

For pipeline execution, we consider the generic case of a pipeline with k segments. The overall execution time is determined by the time the operations spend in the slowest pipeline segment (that element will always be busy), plus the time to process the complete pipeline once:

$$t_{pipeline}(n) = (n-1)\, max_{i=1..k}(t_{segm}[i]) + \sum_{i:=1}^{k} t_{segm}[i] \tag{3.1}$$

where $t_{segm}[i]$ is the time needed for one operation to go through the i'th segment of the pipeline, k is the number of segments and n the number of operations.

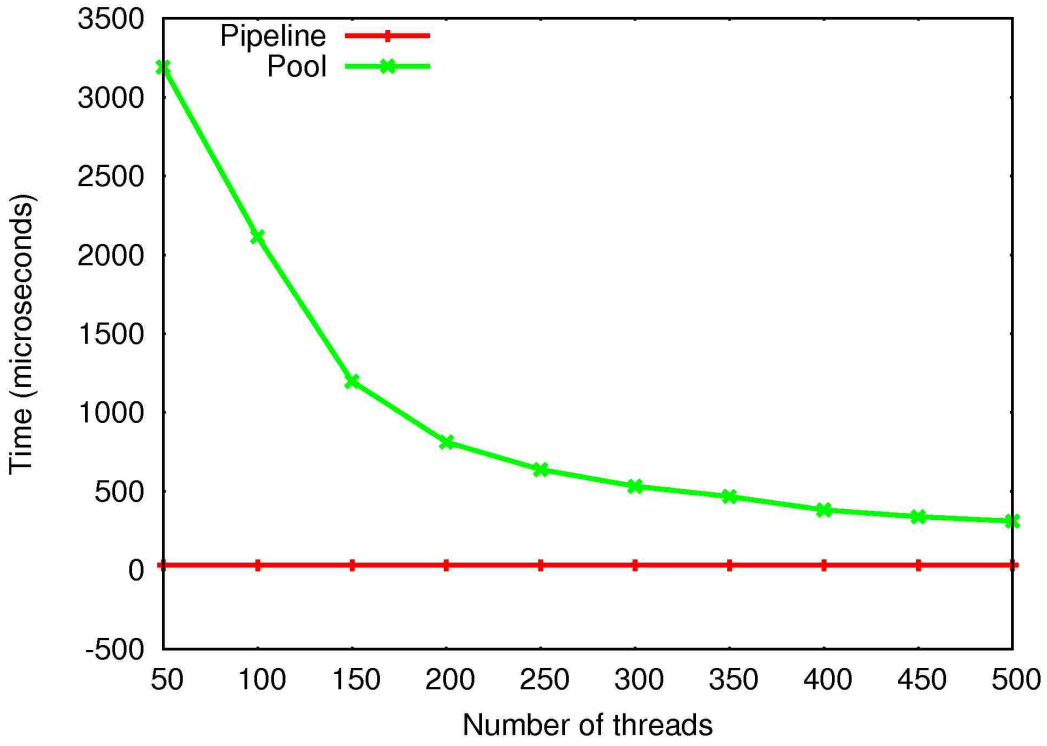

FIGURE 3.9: Execution time/operation on the WAN when executing 300000 operations using the pool system, UDT protocol, with varying number of threads. Pipeline (31 μs) included for reference

For the remote data access system we may consider the pipeline to have three segments: send, receive and server processing. However this would not be an accurate model since it doesn't account for the fact that more than one operation can be active in the send or in the receive channels at any given time. We need to consider the pipeline system to have a larger number of segments for the send and receive channels. The number of segments is equal to the maximum number of messages that can be active in a network channel at any time. This (in the case of equal size messages) is determined by the size of the request and response messages and the network link capacity:

$$\#segments_{send} = \frac{network\_capacity}{request\_size} \tag{3.2}$$

$$\#segments_{receive} = \frac{network\_capacity}{response\_size} \tag{3.3}$$

In addition to the data transfer segments we have additional overhead segments for placing data to- and for retrieving data from the network as well as the server processing segment.

The performance model tells us that if the pipeline is full, then the execution time/operation will be determined by the slowest segment. To fill the pipeline both network channels: the send pipe and the receive pipe have to be filled. If the pipeline is not full the execution time is given by a combination of the number of segments that are not occupied plus the time of the slowest segment from those occupied.

This performance model could be refined to predict the execution time for any number of operations, however this comparative analysis does not require such predictions, as the goal of the analysis is only to compare and understand the performance of the relevant architectures to make a decision for the eaviv system.

To understand the relation between the performance of the pipeline system and the number of operations in the pipeline the execution time/operation for variable numbers of operations in the pipeline was measured. The results for WAN (Figure 3.10) show how the execution time decreases as the number of operations in the pipeline increases.

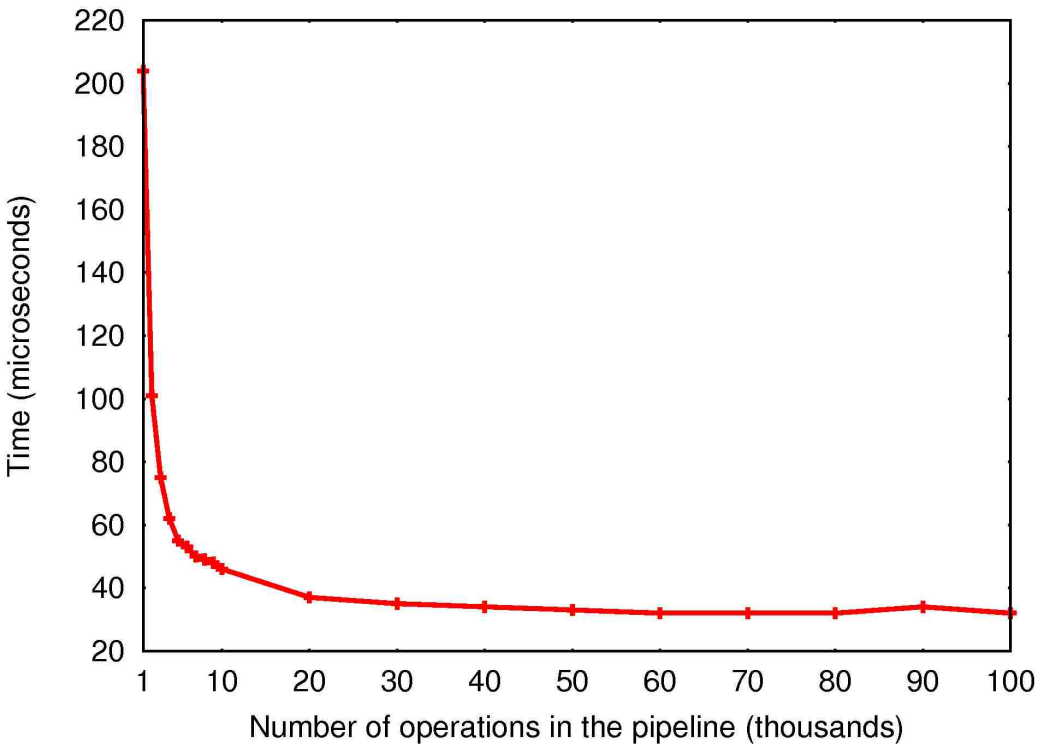

FIGURE 3.10: Execution time/operation when using the pipeline system over WAN, UDT protocol, and varying the number of operations in the pipeline

Starting from a certain number of operations (in this case around 50000) the execution time remains constant. This indicates that starting with that number of operations, the data transfer segment is not the slowest one, and the execution time is determined by the overhead segments. In the LAN, because of the smaller capacity of the network channel (because of short distance) this number is only 2000 operations. Even with a single operation in the pipeline though, the execution time is smaller than that of the threaded or pool systems, as seen in Table 3.3.

3.1.4 Conclusions

The results clearly show that the bulk and pipeline methods are well suited to hide network latency, while the threaded and pool systems fail to provide good performance. The results also show that the threaded and pool systems have a high overhead, making them less suitable than the bulk and pipeline systems for any kind of optimization.

The bulk architecture has the highest throughput of all five proposed systems and has the second-lowest overhead however it doesn't overlap communication and execution. One disadvantage of the "pure" bulk system is that bulk operations are serialized, subsequent sets of operations thus needing to wait for the first set to complete.

The pipeline architecture is the best of those that do not require blocking on the previous operation however its overhead is slightly higher than that of the bulk architecture. The cause of this is likely the high cost of many context switches for each operation.

From the two other implementation architectures that allow for parallelism, the pool system shows potential high overhead in connections and threads set-up time and it cannot reach the level of parallelism of the pipeline. This is because the pipeline allows more operations to be executed in parallel than the thread pool. The pool only allows as many operations to be executed in parallel as there are threads in the pool.

The overhead of executing a single operation using each architecture gives a direct indication of the system load of each system. The higher the measured overhead, the higher the load caused by the architecture. The threaded and pool systems, with their high number of threads will have the highest system load. The pipeline has a higher load than the bulk and synchronous systems because of the multiple threads.

The goal was to find the best architecture for high-throughput remote data access, and the final conclusion of this analysis is that the pipeline architecture is the best architecture for executing multiple non-blocking operations while the bulk architecture offers the highest throughput when multiple operations are executed at the same time. The remote data access system architecture will thus be designed to support a *combination of the pipeline and bulk architectures*, meaning that when possible, multiple operations will be encoded into a single bulk, and these bulk operations will be executed using a pipeline system. The configurability requirement outlined at the beginning of this chapter provides a crucial capability for supporting bulk operations in conjunction with pipeline execution. A remote data access where operations are configurable can easily be adapted to include bulk operations.

The alternative of a simple non-blocking bulk architecture without pipeline support would not be suitable for *eaviv* as it combines multiple operations in a bulk, whereas in *eaviv*'s progressive visualization we need to have multiple asynchronous updates of the data rather than a single, large update. The proposed architecture is illustrated in Figure 3.11.

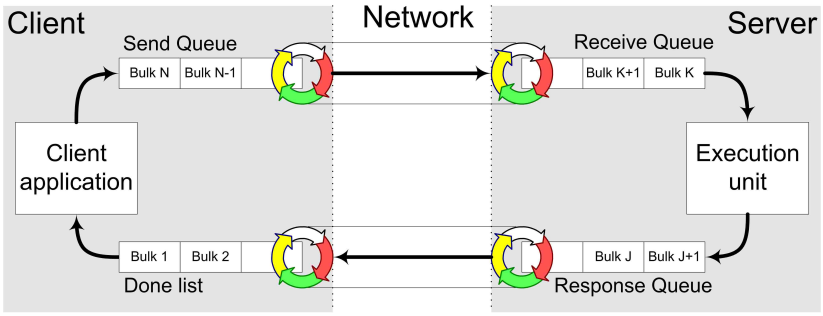

FIGURE 3.11: Proposed system architecture combining the best features of the bulk and pipeline architectures

3.2 *eavivdata* System Architecture

We have so far analyzed the available options for implementing high throughput, non-blocking remote data access. The conclusion is that the remote data access system needs to support pipeline execution as well as bulk operations. As described at the beginning of the chapter, the remote data access system has three other requirements: speed, configurability and needs to be implemented as a library. This

section describes the complete system design, architecture and implementation supporting all the required features, analyzes its performance and compares it with relevant existing work.

3.2.1 Related Work

An alternative to remote data access is file staging, or copying of the entire file locally [STBK08], an approach that is of limited use for this system seeking to use high-speed networks to improve I/O speed.

The German TIKSL [BHM+00] and GriKSL [HHK+05] projects developed technologies for remote data access in Grid environments to provide visualization tools for numerical relativity applications. Based on GridFTP [ABKL05b] and the HDF5 library[3], these projects prototyped a number of remote data access scenarios on which this work is based.

Remote Procedure Call systems such as CORBA[4], SOAP[5], Java RMI[6] provide good options for encoding requests because they are designed for flexible and simple encoding of complex operations. They are also well suited for integration in independent applications. Their data transfer options and performance are however limited. RPC systems usually have integrated transport mechanisms (for example TCP-based HTTP for SOAP) which limit their performance on high-speed networks.

While not specifically designed for remote data access, one the most utilized data transfer systems for grid and distributed applications today is GridFTP. GridFTP can be utilized for remote data access operations using extended retrieve (ERET) operations. With ERET, the client can encode its specific request (as a string), the request being interpreted on the server by a server-side processing plug-in which feeds the data to the client. GridFTP can use various transport protocols through the XIO [ABKL05a] library, with a recent integration of the UDT protocol [BLKF09], supports parallel data streams and has an efficient implementation. GridFTP satisfies for the most part the requirements of supporting of arbitrary data requests and efficient transport protocols. High-throughput execution of multiple non-blocking operations is currently not well supported by GridFTP. Asynchronous execution of single operations is well supported and recently, pipeline support has been

[3] http://www.hdfgroup.org/HDF5/
[4] http://www.omg.org/gettingstarted/corbafaq.htm
[5] http://www.w3.org/TR/soap/
[6] http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp

added to the protocol to enable efficient execution of multiple asynchronous operations [7]. This is however only available for transfers between two GridFTP servers and is not supported in the client library therefore it cannot be used for *eaviv*.

Parrot [TL05] performs remote I/O by intercepting local application I/O calls and redirecting them to a remote system. Parrot can directly attach to an application binary without any changes to the application, making it very convenient for use. The throughput and data transfer performance are however limited and Parrot does not support user-defined remote operations.

FUSE [FUS09] is a library that can be used to implement access to remote file systems. FUSE is used by many remote file system implementations, however it only supports a limited set of remote operations.

Nallipogu et al. [NOL02] show how pipelining provides significant performance improvement in SRB [BMRW98] remote data access, however not addressing transport protocol performance or user-defined remote data access operations.

iRODS[8] is a data and file management system for grids. iRODS supports user-defined operations by enabling the implementation of "micro-operations" on the server, supports parallel data transfer and the high-speed RBUDP protocol [Dav08]. iRODS does not support non-blocking execution of multiple operations, requiring additional work to enable high throughput over wide-area networks [BAK09].

The MPI-IO standard provides asynchronous I/O interfaces and non-contiguous file reads which allow for efficient, high throughput implementations. RIO (Remote I/O) [FDKKM97] implements pipelining of MPI-IO operations, and RFS uses buffering, bulk operations, and a threaded architecture to improve throughput [LRT+04]. The ADIO layer of MPI-IO enables implementations to support efficient remote file access for example by using Logistical Networking [LRA+06] or GridFTP [BW04]. MPI-IO's support for user-defined I/O operations is however limited to file views and user-defined data types and current implementation support for high-performance wide area transport protocols is limited.

[7] See [BLK+07] although the architecture and implementation of the feature are not well documented
[8] `https://www.irods.org/`

DataCutter [BKC+01] has introduced an architecture for data processing based on user-defined filters and pipeline execution.

The Network File System (now at version 4.1) is a widely used protocol for remote file access, however with limited performance over wide area networks.

The Internet Backplane Protocol (IBP) is a middleware for managing and using remote storage that provides a uniform, application independent interface to storage in the network and supports optimization and scheduling of data transfers based on a model that takes into account the network physical resources [BBM+03].

Many other remote data access systems, including Kangaroo [TBSL01], Condor I/O [TTL05], Lustre [Lus09], GPFS [GPF09], DPSS [TLC+99], HPSS [WC95] have been proposed, however no system today supports all the features required by the *eaviv* application: high-speed protocols, high throughput and non-blocking execution, configurable operations and availability as a library.

Much work has been done on the improvement of remote data access throughput via tuning and optimization of the underlying transport protocol. This is discussed in Chapter 4.

3.2.2 System Design

This section describes the *eavivdata* system design including flexible operation encoding and support for transport protocols and the following section (Section 3.2.3) describes the overall system architecture and implementation.

3.2.2.1 Flexible Operations Encoding (Control Channel)

The *eavivdata* system needs to support a wide range of user requests. Client requests can take many forms depending on the data model and the particular application scenario. Simple requests can be represented as "(file, offset, size)" for basic remote file access but can be as complex as "(hyperslab selection of remote Cactus HDF5 file)" for the more complex remote data access and visualization scenario described in Chapter 2. The system needs the ability to represent complex queries and we need the possibility to extend the system capabilities for as yet unknown future types of requests.

The solution I propose to support encoding of arbitrary operations is to use RPC encoding. RPC systems already provide the mechanisms for encoding complex data types and method invocations but are limited in their data transport performance. For *eavivdata* this problem is solved by using

only the encoding and decoding capabilities offered by RPC systems, data transfer being handled separately in order to support high-performance data transfer.

Similar to FTP and GridFTP, two separate channels are used by *eavivdata*: a control channel for request and response communication and a separate data channel for bulk data transmission. RPC systems generally provide their own transport for requests and responses however in *eavivdata* the RPC encoding and decoding is separated from the transport mechanisms. This enables the system to support high-speed protocols allowing high performance data transmission even over the control channel.

The control channel is utilized to implement the send request stage of the remote data access pipeline that is at the core of the *eavivdata* architecture. Conceptually, the RPC response processing is considered to be a part of the control channel as illustrated in Figure 3.12. The encoders and decoders translate the request between the internal data format of the *eavivdata* system and the network format used to transmit the requests over the wire. The control channel is also used by the client to communicate the required endianess of the data to the server, in case there is a difference between the native endianess of the machines running the server and the client.

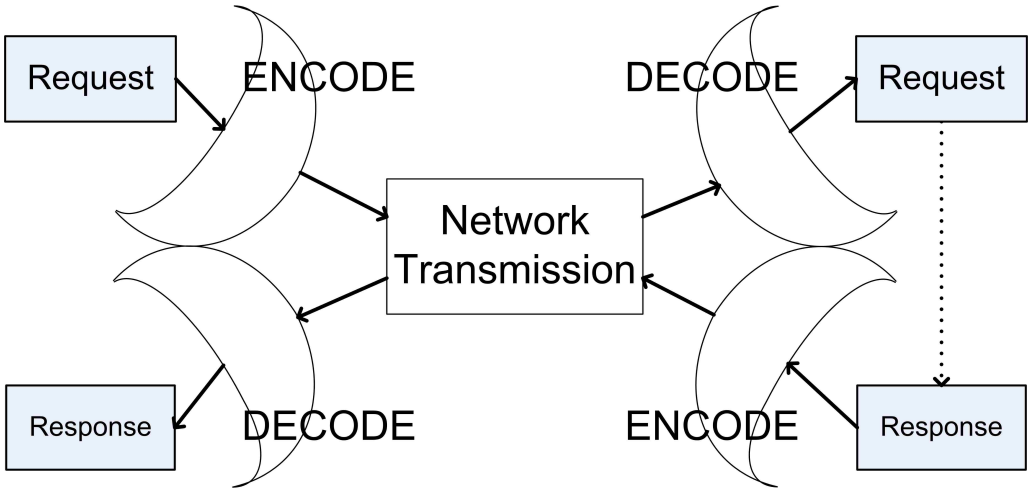

FIGURE 3.12: Control channel design including encoding and decoding of the RPC request as well as encoding and decoding of the RPC response

There are many available RPC systems today, and *eavivdata* uses XML-RPC since XML encoding provides a clean, standard interface and format and allows easy debugging. A possible drawback is that XML processing can limit the performance of the system (maximum number of operations per

second). The XML-RPC[9] for C and C++ library implementation was chosen. This library provides XML encoding and decoding methods that are independent of the underlying transport mechanism, an essential feature for *eavivdata*.

3.2.2.2 Configurable Transport Protocols

This section addresses the requirement of providing maximum performance for data transfers.

To enable high-performance data transfers *eavivdata* does not encode raw data as text and does not perform unnecessary per-byte processing (for example copying between buffers).

For high data throughput, the system is designed to support transport protocols that are able to saturate the network link between the server and client. The approach used for *eavivdata* is to define a data transport interface independent of the transport protocol that is actually used. This interface is implemented using various protocols or libraries. The interface between the *eavivdata* library and the transport protocol implementation is the following:

```
AsyncResultRef beginSend (BlockRef iData,\
   AsyncResultCallback cb);
void endSend (AsyncResultRef handle)

AsyncResultRef beginKnownReceive (BlockRef block,\
   AsyncResultCallback cb)
BlockRef endKnownReceive (AsyncResultRef handle)

AsyncResultRef beginUnknownReceive \
  (AsyncResultCallback cb)
BlockRef endUnknownReceive (AsyncResultRef handle)
```

There are a few notable characteristics of these interfaces. First, reference-counted pointers are used to enable simple management of data buffers without requiring expensive memory copy operations. Second, the interfaces are message-oriented, "BlockRef" being a reference-counted block of raw data.

[9] http://xmlrpc-c.sourceforge.net/

There are two reasons for this: (*i*) the *eavivdata* library operates only on data blocks or messages, so there is no reason to adopt a byte-oriented interface; (*ii*) message-oriented interfaces may enable in the future more efficient data transport protocols implementations that allow for byte reordering at the lower level.

Other characteristics are the callbacks used to notify the higher layers when an operation is finished ("AsyncResultCallback") and the operation handles ("AsyncResultRef") that can be called by the user to block and wait for the operations to finish ("endSend, endReceive").

Finally, there are two data transport methods supported by *eavivdata*. The first method (named "Unknown") is used when the receiver does not know in advance the size of the data to be received. In *eavivdata* configurable transport protocols are used for the control channel as well as for the data channel. The "Unknown" transport method is required for the control channel as the receiver does not apriori know the size of the message when receiving an RPC request or response. The size of the message is dependent on the actual operation, can have data structures with variable size, all this information being encoded inside the actual message. The "Unknown" data transport mode encodes the size of the message inside the data using the first 64 bits of the communication so that the receiver, after decoding the 64 bit header knows exactly how much data to expect.

The second mode (named "Known") is used for the data channel, since for bulk data transfers the receiver knows beforehand what the size of the data to be received is, based on the RPC response received in advance of the bulk data transfer.

The current version of *eavivdata* supports two data transport implementations: TCP sockets (BSD interface) and the UDT library (currently version 4.5). There is a separate set of interfaces from those described above for setting up and configuring connections, and most of the transport protocol tuning parameters (such as congestion control in UDT) are supported.

3.2.3 Integrated System Architecture

This section shows the overall system architecture (Figure 3.13), describes the steps a single operation goes through, top-level initialization of the system and technical details about what needs to be implemented in an application by a user of the *eavivdata* library.

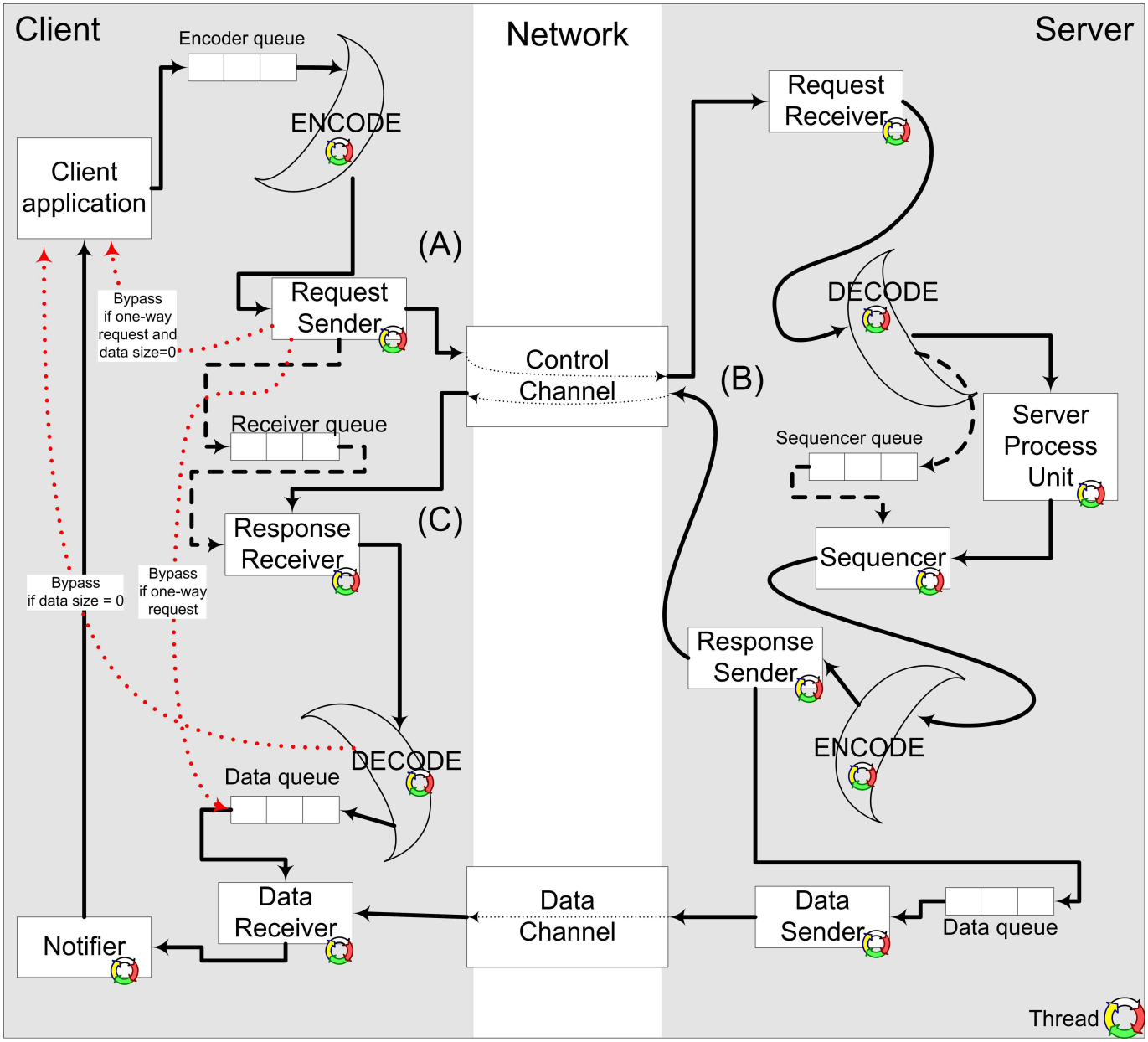

FIGURE 3.13: Complete *eavivdata* system diagram. Dotted lines indicate bypass options - when there is no RPC response message and/or no bulk data. Dashed lines on client and server indicate forwarding of operation information to a future pipeline stage. Solid lines indicate the flow of operations. (A), (B) and (C) indicate stages of execution for an operation (described in the text)

3.2.3.1 Execution Steps

To better understand the system this section describes the processing steps a single operation passes through, following the diagram in Figure 3.13 and the actions taken by the system as the operation is being processed. For clarity, the execution steps are separated in three stages (A, B, C).

(A): The operation is created by the client which then starts the execution process.

The operation is added to a queue that feeds the encoder thread. The encoder transforms the operation into a string by serializing the method name and its parameters and the string is passed to the request sender data thread.

The request sender simply sends the request over the network to the server and subsequently adds the operation to the queue of operations that need to be processed by the response receiver. The request sender has two bypass options. The first bypass option, activated when there is no RPC response to the operation (the operation is a one-way request not requiring a response), has the effect that the operation is directly passed to the bulk data receiver. The second bypass option is activated when there is no bulk data to be received for the operation and has the effect that the operation is finished and the client is notified (callback is called). If no bypass options are applicable and the operation does require an RPC response, the response receiver will receive this response (as a string) over the control channel.

(B): Between the time the RPC request is transmitted to the server and the arrival of the response back to the client, the operation processing takes place on the server side. After receiving the string-encoded request from the client, the request is passed to a decoder that parses the operation parameters and method name.

Using the method name, the appropriate server processing unit is chosen from a list and the operation is transferred to this unit. Because the server processing unit implementation (provided by the application developer) is not actually required to finalize the requests in the order they are received, the server reorders the operations using a sequencer. This is needed to maintain the order of operations as seen by the client[10].

The sequencer waits for operations to finish in the order they were received before passing them further to the response encoder and sender units. The encoder serializes the RPC response of the operation and passes the resulting string to the response sender. The response sender transmits this response to the client and adds the operation to the queue of bulk data transfers to be processed by the data sender.

[10] A possible improvement of the system for the unlikely situation where the sequencer is a bottleneck would be to implement reordering on the client side, this requiring a slightly more complex handling of the operations on both the server and client side

(C): Back to the client side, the RPC response is received and then parsed by the decoder unit. The operation is then added to the queue of bulk data transfers (receive) that is serviced by the data receiver. Another bypass option is available here, if the server responds that there is no data attached to the message then the operation is immediately finished. If the bypass is not activated, after the bulk data transfer is finished, the operation is finalized and the client application is notified.

Of note, there are no bypass options implemented on the server side. This is because if the operation is a one-way operation, the encoder response string will have a size of zero so the response sender will not have anything to send back to the client. If the size of the data transfer is zero, the data sender will in turn have nothing to send to the client.

Some of the operation queues were omitted from the diagram to reduce its complexity however every two connected stages (for example request encoder and request sender) communicate through a queue. There is no blocking in the system, other than waiting on an empty queue. Providing an option to limit the size of the queues may be useful in the future to prevent memory overload, however limiting the queues size produces the risk of reducing the overall system throughput.

3.2.3.2 Top Level Initialization; Tear-down

So far it was shown how the remote data access system works after is initialized, this section describing the initialization process for the pipeline system. One of the characteristics of the pipeline is that, to provide the expected benefit, the pipeline needs to remain connected after is initialized. If the pipeline is destroyed after each operation, no benefit will be seen.

In *eavivdata* the initialization of the system is done through a high level RPC dialog between the client and the server. The server is initially listening on a well defined port for connections from the client. The client creates a connection to the server (this connection is then reused for the control channel) and uses it to communicate the parameters to be used for establishing the data channel.

The data channel can be constructed either by the client connecting actively to the listening server or by the server connecting to the client (useful for traversing firewalls). The data channel options include connection parameters such as port and network interface address as well as transport protocol and tuning parameters (buffer sizes, congestion control, etc.).

The result of the initialization process is an object that is used by the client to execute remote operations. When this object is deleted, the network socket connections are removed. This is detected on the server that then automatically deletes its data structures associated with the connection.

3.2.3.3 Parallel Data Servers

In addition to operation parallelism supported by the pipeline system, *eavivdata* enables parallel data servers by facilitating a single client to use multiple servers at the same time. The servers can be executed on different machines or on the same machine. Running multiple servers on the same machine enables parallel data streams between a single server machine and the client.

This feature is implemented as an additional layer on the client side, on top of the pipeline remote data access system. This new layer manages multiple server connections and distributes the data requests to servers according to application-specific data distribution rules. These rules depend for example on the data that is actually available on each server, or on the speed of each server. No modifications are required on the server to support this type of parallelism.

3.2.3.4 Implementation Requirements for User-Defined Operations

One of the goals of the *eavivdata* library is to minimize the work needed to implement the methods required to support new user-defined operations. This section describes the minimal list of methods that need to be implemented for new operations.

On the client side, the application developer needs to implement a common interface providing the following four methods:

1. "getParams", a method that converts the parameters describing a remote request into a list of XML-RPC parameters

2. "getMethodName" returns the name of the server method that processes requests of this type

3. "oneWay" returns a boolean indicating if the remote operation requires an RPC response or not

4. "parseResponse" takes an XML-RPC response object received from the server and processes it

On the server side, for each type of remote operations a common interface providing the following two methods needs to be implemented:

1. "beginProcessing" takes the list of XML-RPC parameters received from the client and returns a server-side operation object. This new object stores the bulk data and the RPC outcome object that are sent back to the client at the end of the execution

2. the "doProcessing" method receives the server-side object created by "beginProcessing", executes the actual operation and then sets the bulk data and the RPC outcome objects to the appropriate values (bulk data response and RPC response)

3.2.4 Benchmarks and Results

The *eavivdata* library[11] was evaluated by performing a series of benchmarks and also by integrating it into the *eaviv* distributed visualization application. The benchmarks measure the technical performance of the library and the visualization application shows successful integration in a real usage scenario. Two benchmarks were performed, one that analyzes the implementation performance of data transport protocols within *eavivdata* and a second one that compares *eavivdata* with a representative remote data access system existing today: GridFTP (version 4.2.1).

For the benchmarks we used the same 10 Gbps network connection between LSU, Baton Rouge and Brno, Czech Republic (RTT of 149.3 ms), and the following machines:

In Brno: a Linux machine with two dual-core AMD Opteron processors (2.6 GHz) 4 GB RAM. In Baton Rouge: for the first benchmark a Linux machine with two dual-core AMD Opteron processors (2.6 GHz) 4 GB RAM and for the comparison with GridFTP (second benchmark) a dual quad-core Intel(R) Xeon(R) machine, CPU frequency 2.66 GHz, 16 GB RAM.

The first benchmark analyzes the data transport performance of the *eavivdata* library. This analysis is made by comparing the data transfer rate achieved by the *eavivdata* library with the transfer rate achieved by the data transport protocol alone. This illustrates the overhead added by the additional *eavivdata* library processing.

[11] available for download at `http://www.cct.lsu.edu/~ahutanu/codes/eavivdatalib.tgz`

The results of this benchmark, for a data size of 2 GB are the following: the stand-alone UDT library transfers the data in *10.05* seconds; the UDT library, when used within *eavivdata* transfers the data in *10.38* seconds.

The *10.38* seconds measured transfer time of the *eavivdata* library also includes the time needed to initiate the request by the client and the time needed by the server to respond to the data operation and to send this response back to the client. The difference of 0.3 seconds (twice the RTT) is attributed to this additional time that is needed to transfer the (acknowledged) XML request and response between the client and the server. With this consideration, the actual bulk data transfer performance is nearly identical and the *eavivdata* overhead minimal.

The *eavivdata* library was profiled (using callgrind[12]) for this benchmark and the results show that, as expected, most of the execution time is spent within the UDT library. An interesting note, the time used by UDT for memory copy reaches up to 60% of overall execution time, this showing the importance of avoiding memory copy and being a bottleneck for the data transport performance. There is no additional memory copy inside the *eavivdata* library.

The second benchmark analyzes the influence of the number of operations on the data performance and compares the performance of *eavivdata* with GridFTP. This analysis is based on measurements of the data throughput for transferring a 2 GB data block when split in a variable number of operations, using both *eavivdata* and GridFTP. The operations used for this benchmark are simple remote file read operations defined by file offset and data size. To eliminate file system influence from the measurements, the special /dev/zero file was used as the remote data file for both *eavivdata* and GridFTP. The size of 2 GB was chosen as it is representative for the amount of data that is transferred by the visualization application between a single data server and a single visualization node (see Section 3.2.5).

To analyze the influence of parallel data streams on transport performance measurements were taken for both single stream and parallel stream data transfers.

The results for a single data stream (Table 3.6 left, Figure 3.14) show that for *eavivdata* there is no significant decrease in throughput (computed as data size divided by execution time) when the

[12] `http://valgrind.org/info/tools.html`

number of operations is increased as long as the number of operations is not too large (not over 10000 operations). In this case the execution time is dominated by data transfer time.

For a larger number of operations, the throughput is reduced (Figure 3.14 right) because operation processing on the client and the server becomes the dominant factor in execution time.

TABLE 3.6: Average data throughput for *eavivdata* and GridFTP with variable number of operations, with single and parallel streams, total data size 2 GB. s = single stream, p = parallel streams

# operations	eavivdata (s)	GridFTP (s)	eavivdata (p)	GridFTP (p)
1	1959 Mbps	419 Mbps	4625 Mbps	1698 Mbps
10	1920 Mbps	404 Mbps	4608 Mbps	499 Mbps
100	2049 Mbps	311 Mbps	4818 Mbps	235 Mbps
500	2094 Mbps	87 Mbps	4595 Mbps	50 Mbps
1000	2186 Mbps	44 Mbps	4147 Mbps	26 Mbps
5000	2295 Mbps	9 Mbps	2478 Mbps	5 Mbps
10000	2151 Mbps	–	1760 Mbps	–
30000	1927 Mbps	–	777 Mbps	–
50000	1816 Mbps	–	464 Mbps	–
100000	1108 Mbps	–	328 Mbps	–
500000	269 Mbps	–	74 Mbps	–

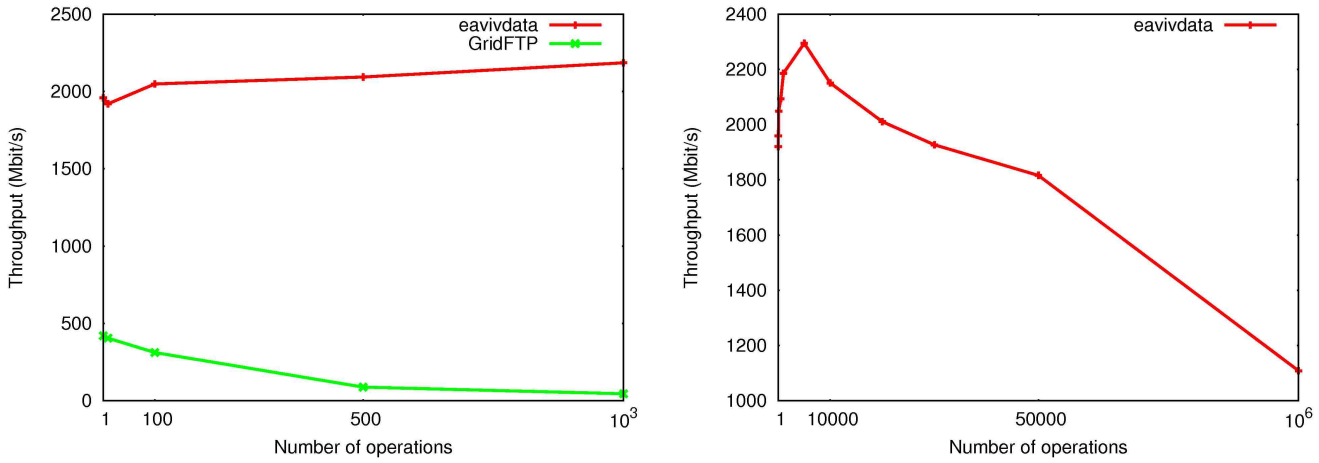

FIGURE 3.14: Average throughput for remote data access using single data streams and variable number of operations. Left: *eavivdata* and GridFTP comparison, centered on 500 operations. Right: *eavivdata* throughput evolution with a large number of operations.

Comparing the performance with GridFTP (Figure 3.14 left), we can see the decrease in throughput appears only for a much larger number of operations in *eavivdata* than when it appears for GridFTP. This shows the benefits of pipelining for *eavivdata*. Because of the serialization of remote operations

and the damaging effect of network latency the GridFTP throughput starts to decrease immediately as the number of operations is increased. The execution time for a large number of operations using GridFTP was not measured as it becomes very large and is completely dominated by network latency (150 ms/operation).

The results also show that *eavivdata* has a higher throughput than GridFTP even for a single operation. This is because of the additional transport protocol tuning options supported by *eavivdata*.

For the data transfers using parallel streams, for *eavivdata* the benchmarks were performed using the optimal numbers of 6 parallel rate based UDT streams. The analysis leading to this configuration is described in Chapter 4. For GridFTP, because it does not support rate based UDT, parallel TCP streams were used. The optimal number of parallel TCP streams for transferring the 2 GB data using GridFTP was determined to be 50 after a series of measurements. In consequence, for GridFTP the benchmarks were performed using 50 TCP streams.

The results (Table 3.6 right, Figure 3.15) show that although the data throughput increases when using parallel transfer, the operation throughput is slightly decreased compared to the single stream case. This is indicated by the more sudden decrease in data throughput as the number of operations increases. The reduced operation throughput is likely caused by the high system load generated by parallel data streaming. The high load reduces the operation processing speed of both the server and the client.

Somewhat surprising is the slight increase in data throughput of *eavivdata*, for both single and parallel streams as the number of operations is increased. Not only the throughput is not reduced by splitting the operation into multiple smaller operations, thus achieving the design goal of *eavivdata* but throughput is actually increased. The maximum throughput is reached at around 5000 operations in the single stream scenario and at 100 operations for parallel streams before starting to decrease. This is an unexpected benefit of pipeline parallelism.

3.2.5 Application Integration

To evaluate its effectiveness and ease of use for a real world scenario, the *eavivdata* library was integrated into the *eaviv* distributed visualization application which has motivated its development.

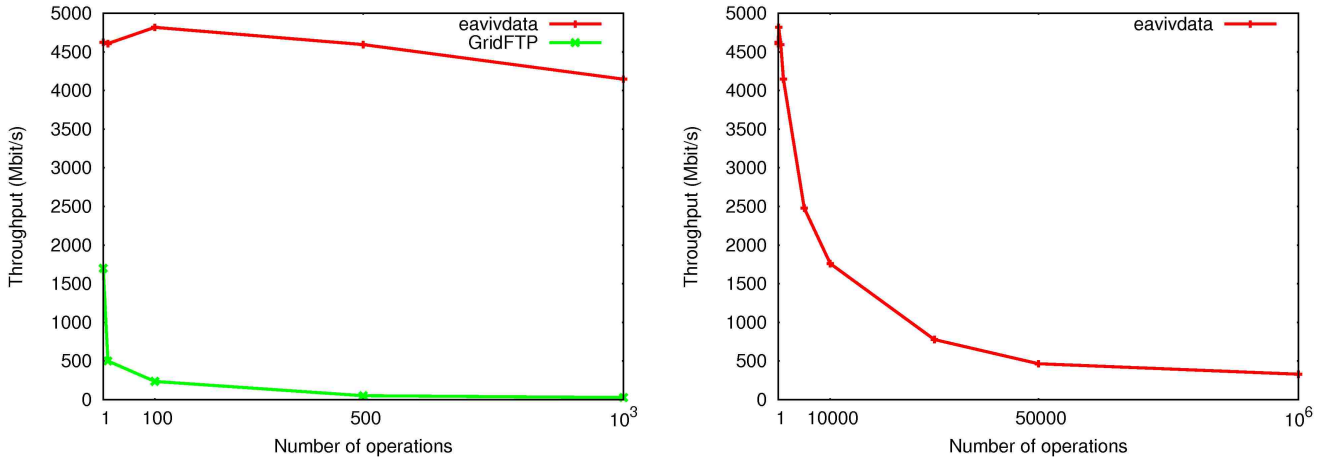

FIGURE 3.15: Average throughput for remote data access using parallel data streams and variable number of operations. Left: *eavivdata* and GridFTP comparison, centered on 500 operations. Right: *eavivdata* throughput evolution with a large number of operations.

The *eaviv* benchmarks in Chapter 2 have been performed with the integrated system using *eavivdata* remote access. The configurability feature of the *eavivdata* system was tested by implementing remote access to Cactus format HDF5 files. HDF5 data selection operations (simple "hyperslabs"[13]) were implemented, as well as remote caching methods used to instruct the *eavivdata* servers to pre-load data of interest in main memory. Data selections are defined as 4D arrays (three spacial dimensions plus time), with start coordinates, size, and strides[14]. Helper operations such as file open, cached read, file close were also implemented. Cached read is used by the visualization application to retrieve the data of interest from the server and file close releases the resources on the server side. The complexity of implementing these operations were not related to the *eavivdata* system, but to the inherent complexity of implementing a 4D cache. Integrating these operations in *eavivdata* was not a difficult task.

eavivdata was used by the visualization application to achieve up to 6.6 Gbps throughput on a 8 Gbps network link showing the real-world applicability of *eavivdata* for high-speed remote data access.

The visualization tests were run using up to 32 data servers, each of them serving a single rendering process. The *eavivdata* servers allow the visualization application to access datasets with multiple

[13] See `http://www.hdfgroup.org/HDF5/Tutor/select.html`
[14] strides = number of elements to be skipped in each direction

timesteps. This means that although an interactively rendered timestep can currently only be up to 60 GB in size, the total data size that can be visualized using *eavivdata* can be terabytes or more, the only limit being the storage capacity of the systems running the servers. The quantity of data that can be cached in main memory and transferred at maximum speed is however limited by the amount of available main memory.

High operation throughput, non-blocking execution and pipelining are well supported by *eavivdata* and used to facilitate progressive visualization. Splitting the remote data access operation of each visualization node in 500 sub-operations showed no reduction in overall transfer time, while substantially improving the user experience by offering visualization updates for each rendered frame (on average). The number of 500 operations, equaling the total number of updates, was chosen as the result of the multiplication of the overall data transfer time (approximately 50 seconds) by the frame rate (approximately 10 frames per second). For 16 rendering processes this comes to a total of 8000 remote data access operations executed for each data transfer.

3.3 Conclusions and Future Work

This chapter described a comprehensive approach to the remote data access problem. A remote data access system supporting high-speed, high operations throughput, non-blocking execution and user-defined operations, not available in this combination in any other system existing today was designed and implemented.

A detailed analysis of implementation architectures was made, showing the need to support both pipeline execution and bulk operations processing to enable high operations throughput.

The design and architecture of the *eavivdata* system allow the efficient execution of the large number of operations needed to support progressive visualization. The system enables the visualization application to split large data transfer operations into multiple smaller ones to improve responsiveness without suffering any performance degradation.

Flexible operation encoding is facilitated by the use of RPC libraries. The system supports configurable high-speed transport protocols enabling it to achieve a higher data and operations throughput than any other system available today.

For the *eaviv* application the current operation throughput is sufficient, however if future case scenarios require it, binary encoding options can be added to eliminate the XML overhead. One area of development for the visualization system is to use the data servers as clients for other servers (chaining of servers). The system was designed with this capability in mind, as this feature will enable significant data transfer optimizations with the application controlling the routing and flow of data in the network.

A further planned usage scenario is to integrate the server-side of the *eavivdata* library into simulation code (such as the numerical relativity application described in Chapter 2) to enable live visualization of generated data.

Although the system was motivated by the interactive visualization of remote data the design is applicable to a wide range of applications and the resulting library can be used to construct data transfer services or by other data processing applications.

Parallelism should be considered in all the components of a distributed system and the solution proposed here can be used as a model for other applications. The *eavivdata* implementation can directly be used today in other applications that require its capabilities.

Chapter 4
Transport Protocols

Computer networks use a layered organization for data transmission. The lowest layer (or layer 1) – the physical layer – transmits data between two network entities connected by a single network link, for example a workstation or a laptop and the router it connects to. The next layer (layer 2) is the data link layer that transmits data packets from node to node based on its physical address (MAC). The next layer (layer 3) is the network layer, which creates network paths between communicating partners by selecting a series of links over which the data flow is directed. A network path is composed by multiple links and usually there are multiple possible paths that data can take, the network layer being responsible for choosing a particular path for the data. The next layer (layer 4) is the transport layer, responsible for delivering data to the appropriate application processes on the host computers. The transport layer can provide reliable or unreliable data transport. Data can get lost or delayed in the network, and to achieve reliable, lossless data communication *reliable transport protocols* are used. They define the sequence of messages and actions (such as retransmission or receipt acknowledgments) the communicating partners need to take to ensure the complete data is transmitted from the source to its destination.

For distributed visualization, remote data access, and other applications a reliable transport protocol that is able to utilize as much as possible of the network capacity is needed[1]. There are big performance differences between different reliable transport protocols. Network performance measurements have shown that the Transmission Control Protocol (TCP), the standard reliable transport protocol in TCP/IP networks such as the Internet has limited performance when used on high-speed networks, and cannot take full advantage of the available network bandwidth. The goal of the work presented here is to understand the options available for transport protocols and to select a protocol suitable for the *eaviv* application.

[1]For certain visualization scenarios, unreliable data transport that may result in errors in the data may actually be acceptable, however reliable data transfer can be used for any visualization application

This chapter discusses transport protocols (Section 4.1), options that are available for applications and users (Section 4.2), compares the performance of a selected set of transport protocols (Section 4.3) leading to a final selection and conclusions for the *eaviv* system (Section 4.4).

The evaluation made here is necessary because while other transport protocol evaluations exist (e.g. [LAQ+08, KHJ08, AAB05, BCHJ03, GACHJ05]), most of them are motivated by simple file copy applications or are made for long lived data transfers (minutes, hours). The analysis presented here is made from the perspective of the *eaviv* distributed visualization application which involves relatively short-lived data transfers (tens of seconds) which have different requirements, and will lead to different results.

4.1 TCP and Alternatives

Traditionally, distributed applications have been designed to use TCP for reliable data communication. TCP provides reliable, ordered, bidirectional, connection oriented data transport. An important feature of TCP is that it implements congestion avoidance. Congestion may appear when multiple concurrent streams travel through the same link or network element (router) or when a stream traverses a low capacity link. If congestion occurs, the overall utility of the network is reduced because of the capacity wasted with retransmissions and transmission of data that eventually is dropped.

In particular, TCP's congestion control characteristics, reducing the window to half when congestion is detected and the slow increase of the congestion window afterward (see Figure 4.1) create performance issues on high-capacity wide area networks.

The congestion window size used by the transport protocol determines how much data is in the process of being transmitted over the network and directly influences the performance of the protocol. TCP's response to a congestion event means that on a network link with no concurrent traffic TCP will not be able to sustain utilization of the full link capacity. The slow increase of the congestion window during the congestion avoidance stage when at most one segment is added to the congestion window each round-trip time means that on long-distance, high-capacity links the time needed to increase the congestion window to reach the link capacity can be measured in hours [Flo03]. Also in practice, bandwidth utilization as a fraction of the total capacity is lower on higher capacity networks

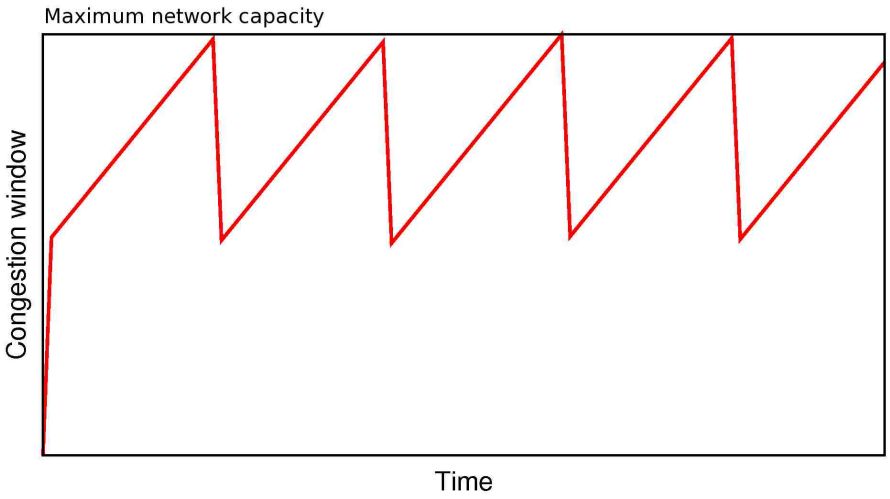

FIGURE 4.1: TCP congestion control showing initial rapid increase of the congestion window during the "slow start" phase, then slow congestion avoidance increase of the window with at most one segment each RTT followed by congestion response, reducing the window to half

since the probability of a single packet loss occurrence that is not caused by congestion is higher when the number of packets traversing the network increases.

Another related issue of the TCP protocol is the unfair distribution of bandwidth between long-distance and short-distance flows to the advantage of short-distance flows. This and other limitations have been shown to appear [KTO07] even in the newer TCP variants that are mentioned below.

In the remainder of this section I give a brief overview of the most common alternative transport protocols, several of which are used for benchmarks.

Many variants of the TCP protocol have been proposed, and most of them introduce modified congestion avoidance algorithms. Congestion control algorithms can differ in how congestion is detected, how congestion avoidance is implemented (what actions are taken to mitigate congestion), or what is communicated between the sender and the receiver.

Scalable TCP [Kel03] and HighSpeed TCP [Flo03] differ from TCP in that they use a different congestion response function.

While TCP detects congestion based on unacknowledged packets, TCP Vegas [BOP94], FAST TCP [WJLH06] and Compound TCP [KTS06] detect congestion by measuring the time delay of transmitting individual packets.

TCP changes its congestion windows size by constantly increasing it as long as congestion is not detected and abruptly reducing when congestion is detected while protocols such as BI-TCP [XHR04] or CUBIC [RX05] use a search method for determining the congestion window size. A similar approach is used by TCP Westwood and TCP Westwood+ [MGF+04].

UDP, or User Datagram Protocol is a standard, connectionless transport protocol for unreliable data transfer. Reliable protocols are implemented on top of UDP by retransmitting the packets that are lost in the network.

Another set of protocols, such as LambdaStream [XLH+05] and UDT [GG07], instead of using a congestion window like TCP use a rate-based congestion control algorithm, thus directly manipulating the transfer rate.

While TCP and other congestion control algorithms are design to transfer data between a single sender and a single receiver, the Group Transport Protocol or GTP [WC04] optimizes congestion control for a multiple senders–single receiver scenario.

Another possibility for data transport, particularly useful for dedicated networks with no concurrent traffic is to avoid protocols that implement congestion control. The alternative is to use data transport protocols that, instead of automatically setting the transmission rate to match the congestion level (like UDT or LambdaStream), allow the transmission rate to be set by the application or the user. Allowing the user to set the transmission rate effectively means that the protocol does not implement congestion control. Reliable Blast UDP, or RBUDP [HLYD02] is an example such protocol where the user sets the data transmission rate.

While most protocols use only the sender and the receiver, other protocols, like XCP [KHR02] have proposed to use network elements (routers) assistance to enable fast data transport.

Applications and users can utilize these protocols and tweak their parameters to improve throughput, for example for TCP parameters such as buffer size and number of parallel streams can be tuned [SBG00, LGT+01, LQD05]. Parallel streams are an effective method of circumventing the

limitations of transport protocols, and it has been shown [YSK08] that the optimal number of streams for peak performance using TCP can be predicted using mathematical modeling.

Solutions that do not use the IP protocol (in TCP/IP networks, IP represents the network layer below TCP), such as wide-area extensions to Infiniband [RYW+08] have also been proposed as a possible solution to the transport protocol problem.

4.2 Application Options

Many of the protocols that have been proposed as alternatives to TCP are not generally available for use in applications, having been designed only as a prototype or just simulated using a network emulator. Some TCP variants are however available as patches for the Linux kernel and since kernel 2.6.7 they are a part of it and can be enabled or disabled. Continuously switching TCP variants in the kernel on a production resource such as a compute cluster is currently not feasible, although theoretically it may be possible.

Clearly, for an application to be able to use a non-standard transport protocol, a user space implementation that does not require kernel modifications is desirable.

Fortunately, the UDT library provides a user space implementation as well as a framework for user-space implementations of congestion-control algorithms [GG05]. UDT has provided[2] reference implementations for the congestion control algorithms of some of the protocols described above including the following TCP variants: HighSpeed, Scalable, BiC, Westwood, Vegas, and Fast. The UDT library also implements a "native" UDT ("*U*DP-based *D*ata *T*ransfer Protocol") protocol, as well as rate-based reliable transmission without congestion control (named RATE in the following).

In addition to the congestion control algorithm, the performance of the transport protocols is influenced by other parameters such as: buffer size, packet size, data rate (for the rate-based protocol) and total data size.

The buffer size directly influences protocol performance because it may limit the quantity of data that can be "in flight", or in transmission between the sender and the receiver. For best results, the buffer size should be set such that the network link can be filled with data. The capacity of a network

[2] in older, not the latest release

link is computed by multiplying its one-way latency by its bandwidth. For example, for a 75 ms, 10 Gbps network link the link capacity is 0.75 Gigabits = 93 Megabytes.

Another factor influencing performance is the maximum packet size supported by the network (or MTU). The smaller the size of the data packets, the more packets are needed to transfer a given amount of data. More packets means more processing on the receiver and sender ends, therefore higher load and lower overall performance.

An important issue when using user defined rate-based transmission is determining the data transmission rate that produces maximum overall throughput. In the benchmarks several measurements were needed to determine the best transmission rate for each particular network and execution scenario. A too low transmission rate under-utilizes the network while a too high rate overflows either the network, or more commonly will overwhelm the receiver resulting in dropped packets and lower overall throughput.

In this work the UDT library is used to compare the performance of the various protocols because of its flexibility in choosing the congestion control algorithm. The next section describes two benchmarks performed using the UDT library, comparing it with TCP and analyzing the protocol parameters that are best suitable for the *eaviv* application. The goal is to gain an understanding of how network-related parameters can be controlled by the application or the user to minimize data transfer time.

4.3 Benchmarks

The benchmarks were performed directly using the *eavivdata* library (Chapter 3).

The first experiment was run using *eavivdata* inside the *eaviv* application, for data transfer from a single LONI cluster (Painter, housed at Louisiana Tech University, Ruston, Louisiana; 16 nodes used as data servers). to an 8-node visualization cluster running the parallel renderer at LSU (aerial distance between Ruston and Baton Rouge is approximately 170 miles). This was part of the *eaviv* distributed visualization benchmarks and the goal of these measurements was to determine what is the best transport protocol option for data transfer within the visualization application.

The protocols used for the benchmark are those supported by the UDT library (native UDT, and the TCP, BIC TCP, HighSpeed TCP, TCP Westwood, Scalable TCP and RATE congestion control plug-ins for the UDT library) plus the native Linux implementation of TCP.

The theoretical capacity of the link is 8 Gbps (network cards on LSU nodes have a capacity of 1 Gbps each), round-trip-time was measured as 7.6 ms. The LONI network is shared by all its users and has a capacity of 10 Gbps. A total of 30 GB of data was used for the benchmark, equally distributed between the 16 data nodes.

The results (Table 4.1) show that RATE achieves the best performance, followed by native UDT and the various TCP versions implemented using the UDT library. The lowest performance was obtained using native TCP which was 8 times slower than RATE.

TABLE 4.1: Transfer rate (in Gbps) achieved over a shared 10 Gbps link (7.6 ms round-trip-time) using various transport protocol algorithms. 30 Gigabyte data total transfer, 16 data clients (running on 8 machines) and servers (running on 16 machines). BIC, HS=HighSpeed TCP, Ww = TCP Westwood, Scal = Scalable TCP are all TCP variants, RATE is user defined rate-based transport. TCP (nat) is the kernel implementation of TCP, TCP (UDT) is TCP congestion control implemented in UDT

TCP (nat)	TCP (UDT)	BIC	HS	RATE	Scal	UDT	Ww
0.64	3.52	3.75	3.11	5.3	2.35	4.4	3.63

The second experiment used the stand-alone *eavivdata* library under a simple scenario of several data senders performing a short-lived data transfer to a single data consumer over a dedicated long distance high-capacity network link.

The benchmark was executed on a network link with an RTT of 149 ms between Baton Rouge, Louisiana and Brno, Czech Republic, with a capacity of 10 Gbps. A 2 GB data size was used with a single data receiver in Baton Rouge and one to six network flows. The six network flows correspond to six remote data servers which were run on one or two machines in Brno. The Brno machines were running Linux and were equipped with two dual-core AMD Opteron processors (2.6 GHz) and 4 GB RAM. The receiver at LSU was a dual quad-core Intel(R) Xeon(R) machine, CPU frequency 2.66 GHz, 16 GB RAM.

The goal of this benchmark was to investigate the effect of a truly long-distance network on transport protocol performance, the effect of data streaming parallelism and also to see the difference between parallelism on a single server and parallelism on multiple servers.

The same set of protocols as in the first benchmark was used, however, for clarity, the results in Table 4.2 show only the three best and most stable protocols.

TABLE 4.2: Transfer rate (in Mbps) achieved over a dedicated 10 Gbps link (149 ms round-trip-time) using various congestion control algorithms in the UDT library. 2 GB data messages were used. The number of streams is equal to the number of data servers and the servers were run on one and two machines (shown in parentheses next to protocol). Scal = Scalable TCP plug-in inside the UDT library, UDT is the native UDT protocol and RATE is the user-defined rate-based protocol plug-in inside the UDT library. Best results are shown in italics. $\sim$ indicates great variance in measurements

# streams	RATE (1)	RATE (2)	UDT (1)	UDT (2)	Scal (1)	Scal (2)
1	*1858*	*1858*	808	808	1619	1619
2	*3361*	1501	954	991	2361	$\sim$ 1633
3	*4093*	*1932*	1096	1111	183	$\sim$ 1321
4	*4307*	*2174*	1222	1274	202	$\sim$ 1038
5	*4683*	*2504*	1296	1343	219	$\sim$ 219
6	*4880*	*2629*	1383	1443	236	$\sim$ 805

Three to five measurements were taken for each experiment, the results presented here being the average over all measurements. Some of the protocols show great variance in performance (standard deviation), up to a factor of 10 in performance difference between measurements for Scalable TCP and TCP Westwood, something that should not happen on a dedicated link like that used for these measurements. This indicates that these protocols are not suitable for applications such as *eaviv*. Of note, the measured performance of the kernel TCP implementation was 16 Mbps (with 6 servers running on two machines) and less.

The results show that the rate-based approach is the best one, followed by UDT for parallel streams and Scalable TCP for single streams respectively. The results for Scalable TCP when data servers were executed on two machines show great variance, showing the unreliable performance of Scalable TCP when parallel streams are used.

The theoretical capacity of the link is near 10 Gbps and the performance of the best protocol (RATE) is limited by the CPU speed. There is room for further improvement and *valgrind* profiling

shows that up to 60% of data transfer time is spent doing possibly unnecessary memory copy inside the UDT library.

When the servers are executed on two machines, although it still shows the best performance, the rate-based approach has a reduced performance compared to the case when the servers are run on a single machine. The most likely reason for this is that when the servers are executed on multiple machines the different machines may send their data at the same time, creating artificial bottlenecks, and bursts of traffic in the network (a possible solution to this problem is discussed in the next section).

4.4 Conclusions

The benchmarks show that the user defined rate-based protocol has the highest data transfer performance, which was expected, making this the current protocol of choice for *eaviv*. The rate-based protocol aggressively consumes as much bandwidth as it is instructed by the user. On shared networks, such as LONI this type of aggressive network use may compete unfairly with TCP-based applications, and depending on the network provider policy, the native UDT or other TCP-friendly protocols may need to be used.

TCP variants, as well as native UDT have problems in taking advantage of the network capacity making them unsuitable for *eaviv*. Their problems arise from the premises of uncertainty that they are designed to work under: unknown available network capacity and possible concurrent traffic that needs to be accommodated. The rate-based approach works under the assumption that concurrent traffic either does not exist or is precisely known, and can achieve much better performance than the other protocols.

Determining the best transmission rate for the rate-based protocol is an important issue, and setting the rate automatically is an useful next step. Chapter 6 provides some ideas about how this issue can be addressed in the future.

Future work will include continuous evaluation of transport protocols for various network situations as well as for other application scenarios and searching for new transport protocol implementations and integrating them in *eavivdata*.

The performance issues seen when multiple data servers transmit data to a single client machine are probably related to burstiness of data transfer (see PSPacer [TMK+07]). This issue will need to be addressed in the future, so that as the number of data servers is increased the performance will not drop. PSPacer may be able to provide a solution to this issue.

The transport protocol options that are available for network-aware application today are very limited. There is a strong need for transport protocol implementations that will enable applications to take advantage of high-speed networks. The UDT library provides a great tool today but more such implementations are needed.

Chapter 5
HD Classroom

Most of the work described in the previous chapters (visualization, remote data and transport protocols) has been motivated by the *eaviv* visualization application. This chapter illustrates the use of high-speed networks in an area unrelated to *eaviv* showing network-aware application design in a completely different field by describing a different application: the "HD Classroom".

Motivated by the previous development and successful demonstrations of video conferencing using uncompressed high-definition video, an experiment assessing the value and applicability of the new technology for distributed and collaborative teaching has been conducted. This experiment has been designed to be conducted in a production environment (classroom teaching) with very strong application quality requirements.

The opportunity was given by the introduction of a new course, "Introduction to High Performance Computing", taught at Louisiana State University by Professor Thomas Sterling.

With the participation of Masaryk University in Brno (Czech Republic), University of Arkansas in Fayetteville, Louisiana Tech University in Ruston and later joined by North Carolina State University in Raleigh (through MCNC, North Carolina) LSU has initiated the "HD Classroom" experiment attempting to create a highly interactive environment to allow students and teachers from all these universities to actively participate in the HPC course. The goal was to both provide the tools needed for remote teaching of the HPC course and to analyze the requirements for utilizing the supporting research technology in a stable production environment.

The class took place in the spring semester (January – May) of 2007. This chapter briefly presents the design and technology utilized for the HD classroom experiment, starting with an application overview (Section 5.1), continuing with details of the supporting network infrastructure (Section 5.2) and finishing with conclusions (Section 5.3). A complete description of the experiment can be found in: Hutanu et al. [HPE+07].

5.1 Overview

At the core of the experiment was the decision to use uncompressed high-definition video. HD video offers high detail, allowing students to see detailed facial expressions of the lecturers and providing a realistic remote presence experience.

This comes at the cost of generating large amounts of data. This data could be compressed however even on dedicated hardware this takes a long time. Additionally, the lag created by compressing and decompressing video without dedicated hardware (unavailable at the time of the experiment) is unacceptable for a real-time collaborative environment. Compression overhead can be avoided by using uncompressed video. This however comes at a cost: uncompressed video uses much more network bandwidth.

The videoconferencing system used for the experiment was based on the Ultragrid [HML+06] software and enabled multi-party video-conferencing using uncompressed 1080i HD (1920x1080 pixels) video.

The video communication using UltraGrid is point-to-point only. However the five partners participating in the virtual classroom needed a multi-point distribution of the video stream. To create a multipoint conference and deliver the class contents to all participants an overlay network was built where specific nodes took care of data distribution. These distributing nodes receive a copy of the video stream and distribute it to up to 4 other nodes, and they can also be chained.

Previous measurements of end-to-end latency caused by video processing alone showed that capturing, sending, receiving and displaying of the 1080i video takes 175±5 ms. More details on this can be found in [HML+06]. Adding network latency and distribution latency, the end-to-end latency on the longest link (from LSU to Brno) was about 250–300 ms, still barely noticeable for human perception and generally not disturbing interactive communication.

5.2 Network

The HD video transmission has substantial network requirements. In addition to the raw bandwidth needed (1.5 Gbps for each stream including the overhead generated by packetization) a real-time HD video transmission requires minimal transmission latency. The transmission latency is influenced

by the wire latency (fixed) but also by the presence of data rate fluctuations (jitter). Jitter can be compensated by using buffering but this comes at the cost of additional latency and is undesirable in an interactive environment. Additionally, large data packets are necessary for the software on the end hosts to be able to sustain the high transmission rates, so "jumbo" packets need to be enabled in the entire network.

Commodity Internet cannot normally meet these requirements and even specialized shared network services are generally not suitable for this application, dedicated circuits being the only network service that can with certainty meet all network requirements. Since any unnecessary packet processing comes at a cost (adds jitter) routing should also be avoided whenever possible.

The network for the experiment was built from a combination of dedicated point-to-point (layer 1) and switched (layer 2) network links offered by various providers combined into a switched network using minimal bridging over routed (layer 3) networks where necessary.

Dedicated circuits are able to provide the necessary service but the cost of having all these resources dedicated for a single application that runs occasionally such as the HD classroom is not cost effective.

The Enlightened project, which provided part of its testbed for the experiment has developed HARC [Mac07], a software package used for co-allocating multiple types of resources at the time the application needs them. Using HARC, resource utilization can be maximized as the resources are freed when the application execution time is over and can be allocated and used by other applications.

HARC can be utilized to allocate both network and compute resources, however in this experiment the compute end nodes were dedicated for the application so HARC was only utilized to allocate network links.

The experiment showed that although not widely deployed, all the network-related components needed to execute such a demanding application in a production environment are available and the experiment proved that they can be effectively utilized to support a real application.

5.3 Conclusions

"HD Classroom" was the first ever teaching experiment using uncompressed HD video with application driven network provisioning.

FIGURE 5.1: HD classroom: Professor Thomas Sterling teaching remotely to students in Brno (left) and Arkansas (right).

The experiment showed that despite the high development, deployment and maintenance costs plus a wide range of technical difficulties (for details see [HPE+07]) we could provide the necessary service for students around the world to effectively participate in the "Introduction to HPC" class.

This experiment illustrates how developing network-aware applications that may drive the deployment of high-speed networks can lead to applications and environments that have a strong impact on the society. Education is an important area in all parts of the world, and this experiment showed how network-aware tools can help change the classic classroom and education model. High-quality remote teaching tools enable students from potentially anywhere in the world to get access to the best specialists in any particular field and participate in their classes, even if they are not physically located at the local university.

Following the experiment, a new classroom space, specifically designed for remote teaching was prepared at LSU. The teaching of "Introduction to HPC" to remote students continued in 2008 and 2009, and a new distributed class for video game design has been initiated (using a different videoconferencing technology that has lower bandwidth requirements). The HD classroom experiment has been a motivating factor for new developments such as the "High Performance Digital Media Network" testbed [HPD09] and the CoUniverse [LH08] application manager, all of these showing the enabling potential of this application.

Chapter 6
Conclusions and Future Work

This thesis has shown how networks can be used effectively to build next-generation distributed applications. Using an integrated approach and taking into consideration all the building blocks of an application from the design stages enables the creation of applications that have higher performance, improved scalability and more features than comparable applications existing today.

A main focus of this work is on interactive visualization of large data where I have shown how high-speed networks can be used to increase the amount of data that can be interactively visualized, improve I/O performance and enable collaboration while maintaining high quality for the interactive visualization environment. The results show that remote I/O using high-speed networks can be up to 30 times faster than local I/O and that, by using remote rendering clusters, up to 60 times more data than on a typical workstation can be interactively visualized.

For the visualization application to be able to effectively access remote distributed data servers without degrading the user experience we need to take into account network latency and design the remote data access system to reduce its damaging effect on application performance. This work evaluated five alternative mechanisms for remote data access and concluded that an architecture that combines bulk and pipeline processing is the best solution for high-throughput remote data access. The resulting system, combining high-throughput remote access with high-speed data transport protocols and configurable remote data access operations is, depending on the transport protocol configuration and number of operations, from 3 to over 400 times faster than a comparable existing remote data access system.

The evaluation of transport protocols on dedicated wide-area networks showed that rate-based protocols designed for high-speed networks achieve a transfer rate of over 8 times faster than that of the standard Internet transport protocol TCP.

The experiment with HD video conference supported remote teaching illustrated the enabling potential of high-speed network applications for areas other than visualization.

Each system component presented in this work has a future development path. The visualization application can be enhanced to use multiple distributed rendering clusters. Direct visualization of streamed simulation data could be supported by integrating the remote data access server into a live running simulation.

More transport protocols need to be evaluated, and methods for eliminating the burstiness of data transfer need to be investigated to further improve the performance of the data transport system.

Other application areas should be investigated, and more applications should be modified or developed to take advantage of emerging network infrastructure, to further research the generality of this approach. Education is a particularly promising area for future development, and considering the emergence of specialized hardware architectures so are distributed simulations where each component of the simulation is executed on the most suitable hardware in the network. Other possible applications that may benefit from the new high-speed network infrastructure are emergency response applications that require immediate access to a vast pool of interconnected resources.

For applications such as the presented *eaviv* distributed visualization it is important that all resources required are allocated together so that a single application can use them at the same time. In the future we will need to work with both compute and network resource providers to enable coordinated service provisioning and co-allocation of resources.

An important part of (distributed) application execution is optimizing the application performance, which involves both choosing the resources where the applications runs, and configuring the application to achieve optimal performance. Since initial investigations have already been undertaken in this area and because of its crucial importance for application performance and scalability, the following text describes the future work in optimization in more detail.

Assuming that the network and compute resources are scheduled (a reasonable assumption for dedicated networks and grids) so for any given time period a user can discover what resources are available for execution, the proposed methodology is:

1. user requests the resource (network, compute, etc) schedule for the time period of interest;

2. based on the application requirements, an optimization algorithm leads to the selection of an appropriate subset of the available resources including configuration details;

3. using co-allocation middleware the selected resources are allocated and the application executed on these resources at the selected start time;

To address the issue of how to optimize a general distributed application we first need to answer two questions: (1) what are the elements to be optimized and (2) what are the inputs for the optimization problem.

Regarding the first question, the performance of an application such as the *eaviv* visualization system, as described in Chapter 2, is defined by multiple parameters, including frame rate, resolution, data size and data transfer rate. They can each be optimized individually, but that can have severe limitations. For example, there would be no purpose in having a system that to improve video quality allocates so much bandwidth for video transmission that it leaves no bandwidth available for data transfer. For an application such as *eaviv* multiple system parameters should be considered for optimization.

Regarding the second question, the input for the optimization problem is defined by the set of resources available for execution and the parameters describing them, as well as performance models that give reasonable predictions on how application components perform on various resources. In the case of a distributed application the performance model should accurately model both application component performance and the performance of the network.

Resource selection and optimization algorithms cannot use information about network resources in making decisions if that information is hidden, as is the case in the standard layered Internet architecture (see Chapter 1). Since network performance is crucial for distributed applications, some optimization algorithms circumvent the issue by relying on estimates or guesses of available network bandwidth using tools such as Pathchirp [RRB+03] and Pathload [JD02]. However, assuming a dedicated, deterministic network infrastructure is available, the approach we will follow is to design and utilize optimizations algorithms that have exact knowledge of network resources [TC07, CT09].

An example optimization problem that uses information about the network is minimizing data transfer time. For the *eaviv* application it has been shown (Chapter 2) that the I/O bandwidth of the application can be increased when using network data servers. In the visualization scenario, distributed data servers are used to serve data to the rendering component. The problem is, assuming the rendering component is executed at a particular fixed location, to select and configure the resources used for data server deployment to get maximum data throughput.

If network topology and capacity are known, which is the assumption that we work under, network flows can be used to model network and application performance, and the data transfer optimization problems are translated to maximum network flow problems, as shown in Toole and Hutanu [TH08].

Applications needing optimizations of a single criteria such as the data transfer rate already present some very challenging and interesting problems but for applications such as *eaviv* multiple criteria optimization problems need to be formulated and solved. In the previous data transfer example the location of the rendering component is fixed, however by changing this and including the selection of the renderer location in the optimization, the problem immediately transforms into a multiple criteria optimization problem.

In *eaviv* the user should be able to select a set of preferences or importance factors for the various optimization criteria (frame rate, data size, etc.) and the system should take some or all these criteria into consideration to provide one or more configuration options to the user. Using a multi-criteria optimization approach [KNOW08] may be a promising method, however the *eaviv* optimization problem remains an open issue for future work.

Concluding Remarks

The National Science Foundation has recently funded our "Strategies for Remote Visualization on a Dynamically Configurable Testbed" project [NSF09] that, with a focus on distributed visualization, will develop and research strategies for designing and optimizing distributed applications on configurable high speed networks where the networks are considered of the same importance as compute and storage resources. The project will build a state-of-the-art real world testbed for distributed applications development between LSU (Baton Rouge, LA), NCSA (Urbana-Champaign, IL), ORNL

(Oak Ridge, TN), TACC (Austin, TX) and MU (Brno, Czech Republic) connected by Internet2 ION dynamic network services.

Deterministic, dedicated network services should become available outside the research network community and accessible for a wide range of users. Following this approach, considering for example HD video streaming applications, we should believe that instant streaming of at least blu-ray quality movies, HD videoconferencing to any place in the world, remote medical consultations or live streaming of any event will be possible in the near future, for any broadband network subscriber.

For the past 20 years, networks have continuously changed our lives and by taking the right steps forward and enabling collaboration between application developers and network providers we can ensure that future generations will be even closer connected than today, in a truly global community.

Bibliography

[AAB05] Eitan Altman, Konstantin Avrachenkov, and Chadi Barakat. A stochastic model of TCP/IP with stationary random losses. *IEEE/ACM Trans. Netw.*, 13(2):356–369, 2005.

[ABKL05a] William Allcock, John Bresnahan, Rajkumar Kettimuthu, and Joseph Link. The Globus eXtensible Input/Output System (XIO): A Protocol Independent IO System for the Grid. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 4*, page 179.1, Washington, DC, USA, 2005. IEEE Computer Society.

[ABKL05b] William Allcock, John Bresnahan, Rajkumar Kettimuthu, and Michael Link. The Globus Striped GridFTP Framework and Server. In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 54, Washington, DC, USA, 2005. IEEE Computer Society.

[ADG+05] G. Allen, K. Davis, T. Goodale, A. Hutanu, H. Kaiser, T. Kielmann, A. Merzky, R. van Nieuwpoort, A. Reinefeld, F. Schintke, T. Schütt, E. Seidel, and B. Ullmer. The Grid Application Toolkit: Toward Generic and Easy Application Programming Interfaces for the Grid. *Proceedings of the IEEE*, 93(3):534–550, March 2005.

[AOS+00] Alexander B. Arulanthu, Carlos O'Ryan, Douglas C. Schmidt, Michael Kircher, and Jeff Parsons. The design and performance of a scable ORB architecture for CORBA asynchronous messaging. In *Middleware '00: IFIP/ACM International Conference on Distributed systems platforms*, pages 208–230, Secaucus, NJ, USA, 2000. Springer-Verlag New York, Inc.

[AW06] Robert D. Arkinson and Philip J. Weiser. A "Third Way" on Network Neutrality. The Information Technology and Innovation Foundation, May 2006. `http://www.itif.org/files/netneutrality.pdf`.

[BAK09] Mehmet Balman, Ismail Akturk, and Tevfik Kosar. Intermediate Gateway Service to Aggregate and Cache the I/O Operations into Distributed Storage Repositories. In *FAST09 Poster Session*, 2009.

[BALL89] B. Bershad, T. Anderson, E. Lazowska, and H. Levy. Lightweight remote procedure call. In *SOSP '89: Proceedings of the twelfth ACM symposium on Operating systems principles*, pages 102–113, New York, NY, USA, 1989. ACM Press.

[BBC+07] K. W. Brodlie, J. Brooke, M. Chen, D. Chisnall, C. J. Hughes, Nigel W. John, M. W. Jones, M. Riding, N. Roard, M. Turner, and J. D. Wood. Adaptive Infrastructure for Visual Computing. In *Theory and Practice of Computer Graphics*, pages 147–156, 2007.

[BBM+03] Alessandro Bassi, Micah Beck, Terry Moore, James S. Plank, Martin Swany, Rich Wolski, and Graham Fagg. The internet backplane protocol: a study in resource sharing. *Future Gener. Comput. Syst.*, 19(4):551–562, 2003.

[BCD+06] A. Bobyshev, M. Crawford, P. DeMar, V. Grigaliunas, M. Grigoriev, A. Moibenko, D. Petravick, and R. Rechenmacher. Lambda Station: On-Demand Flow Based Routing for Data Intensive Grid Applications Over Multitopology Networks. In *Broadband*

Communications, Networks and Systems, 2006. BROADNETS 2006. 3rd International Conference on, pages 1–9, Oct. 2006.

[BCHJ03] Hadrien Bullot, R. Les Cottrell, and Richard Hughes-Jones. Evaluation of Advanced TCP Stacks on Fast Long-Distance Production Networks. *Journal of Grid Computing*, 1(4):345–359, December 2003.

[BD05] Sean Baker and Simon Dobson. Comparing Service-Oriented and Distributed Object Architectures. In *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, volume 3760 of *Lecture Notes in Computer Science*, pages 631–645. Springer Berlin / Heidelberg, October 2005.

[BDG+04] K. Brodlie, D. Duce, J. Gallop, M. Sagar, J. Walton, and J. Wood. Visualization in grid computing environments. *IEEE Visualization*, pages 155–162, 2004.

[BEK+00] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. Simple Object Access Protocol (SOAP) 1.1. W3C Note, May 2000. `http://www.w3.org/TR/2000/NOTE-SOAP-20000508`.

[BGI+03] Raouf Boutaba, Wojciech Golab, Youssef Iraqi, Tianshu Li, and Bill St. Arnaud. Grid-Controlled Lightpaths for High Performance Grid Applications . *Journal of Grid Computing*, 1(4):387–394, December 2003.

[BHKE+07] L. Battestilli, A. Hutanu, G. Karmous-Edwards, D.S. Katz, J. MacLaren, J. Mambretti, J.H. Moore, Seung-Jong Park, H.G. Perros, S. Sundar, S. Tanwir, S.R. Thorpe, and Yufeng Xin. EnLIGHTened Computing: An architecture for co-allocating network, compute, and other grid resources for high-end applications. *High Capacity Optical Networks and Enabling Technologies, 2007. HONET 2007. International Symposium on*, pages 1–8, Nov. 2007.

[BHM+00] Werner Benger, Hans-Christian Hege, André Merzky, Thomas Radke, and Edward Seidel. Efficient Distributed File I/O for Visualization in Grid Environments. In B. Engquist, L. Johnsson, M. Hammill, and F. Short, editors, *Simulation and Visualization on the Grid*, volume 13 of *Lecture Notes in Computational Science and Engineering*, pages 1–6. Springer Verlag, 2000.

[BJA+09] E.W. Bethel, C.R. Johnson, S. Ahern, J. Bell, P.-T. Bremer, H. Childs, E. Cormier-Michel, M. Day, E. Deines, P.T. Fogal, C. Garth, C.G.R. Geddes, H. Hagen, B. Hamann, C.D. Hansen, J. Jacobsen, K.I. Joy, J. Krüger, J. Meredith, P. Messmer, G. Ostrouchov, V. Pascucci, K. Potter, Prabhat, D. Pugmire, O. Rübel, A.R. Sanderson, C.T. Silva, D. Ushizima, G.H. Weber, B. Whitlock, and K. Wu. Occam's Razor and Petascale Visual Data Analysis. In *Proceedings of SciDAC 2009*, volume 180 of *Journal of Physics: Conference Series*, page (published online), 2009.

[BKC+01] Michael D. Beynon, Tahsin Kurc, Umit Catalyurek, Chialin Chang, Alan Sussman, and Joel Saltz. Distributed processing of very large datasets with DataCutter. *Parallel Comput.*, 27(11):1457–1478, 2001.

[BKS05] Rüdiger Berlich, Marcel Kunze, and Kilian Schwarz. Grid computing in Europe: from research to deployment. In *CRPIT '44: Proceedings of the 2005 Australasian workshop on Grid computing and e-research*, pages 21–27, Darlinghurst, Australia, 2005. Australian Computer Society, Inc.

[BL94] Phillip Bogle and Barbara Liskov. Reducing cross domain call overhead using batched futures. *SIGPLAN Not.*, 29(10):341–354, 1994.

[BLK+07] John Bresnahan, Michael Link, Rajkumar Kettimuthu, Dan Fraser, and Ian Foster. GridFTP Pipelining. In *Proceedings of the 2007 TeraGrid Conference*, June 2007.

[BLKF09] John Bresnahan, Michael Link, Rajkumar Kettimuthu, and Ian Foster. UDT as an Alternative Transport Protocol for GridFTP. In *Proceedings of the International Workshop on Protocols for Future, Large-Scale and Diverse Network Transports (PFLDNeT)*, 2009.

[BMRW98] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC Storage Resource Broker. In *Conference of the Center for Advanced Studies on Collaborative Research*, 1998.

[BN84] Andrew D. Birrell and Bruce Jay Nelson. Implementing remote procedure calls. *ACM Trans. Comput. Syst.*, 2(1):39–59, 1984.

[BOP94] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. TCP Vegas: new techniques for congestion detection and avoidance. In *SIGCOMM '94: Proceedings of the conference on Communications architectures, protocols and applications*, pages 24–35, New York, NY, USA, 1994. ACM.

[BPS+03] Arthurine Breckenridge, Lyndon Pierson, Sergiu Sanielevici, Joel Welling, Rainer Keller, Uwe Woessner, and Juergen Schulze. Distributed, on-demand, data-intensive and collaborative simulation analysis. *Future Generation Computer Systems*, 19(6):849–859, 2003.

[BR92] Bob Boothe and Abhiram Ranade. Improved multithreading techniques for hiding communication latency in multiprocessors. *SIGARCH Comput. Archit. News*, 20(2):214–223, 1992.

[Bri07] Bob Briscoe. Flow rate fairness: Dismantling a religion. *ACM SIGCOMM Computer Communication Review*, 37(2):63–74, April 2007.

[BS03] E. Wes Bethel and John Shalf. Grid-Distributed Visualizations Using Connectionless Protocols. *IEEE Comput. Graph. Appl.*, 23(2):51–59, 2003.

[BW04] T. Baer and P. Wyckoff. A parallel I/O mechanism for distributed systems. *Cluster Computing, IEEE International Conference on*, 0:63–69, 2004.

[CBB+05] H. Childs, E. Brugger, K. Bonnell, J. Meredith, M. Miller, B. Whitlock, and N. Max. A Contract Based System For Large Data Visualization. *Visualization, IEEE 2005*, pages 25–25, 2005.

[CBB+07] Joan M Centrella, John G Baker, William D Boggs, Bernard J Kelly, Sean T McWilliams, and James R van Meter. Binary black holes, gravitational waves, and numerical relativity. *Journal of Physics: Conference Series*, 78:012010 (10pp), 2007.

[Cer07] Vinton G. Cerf. Prepared statement. U.S. Senate Committee on Commerce, Science, and Transportation Hearing on "Network Neutrality", February 2007. http://commerce.senate.gov/pdf/cerf-020706.pdf.

[CES09] CESNET, Czech NREN operator. Web page, 2009. http://www.ces.net/.

[CG90] Douglas Comer and James Griffioen. A New Design for Distributed Systems: The Remote Memory Model. In USENIX Association, editor, *Proceedings of the USENIX Summer 1990 Technical Conference*, pages 127–136, June 1990.

[CGM+06] A. Cedilnik, B. Geveci, K. Moreland, J. Ahrens, and J. Favre. Remote Large Data Visualization in the ParaView Framework. *Proceedings of the Eurographics Parallel Graphics and Visualization*, pages 162–170, 2006.

[Chi03] Radu Chisleag. Providing Central and East European Engineering Students with International Experience. In *Enhancement of the Global Perspective for Engineering Students by Providing an International Experience*, 2003. http://services.bepress.com/eci/enhancement/19/.

[Cla88] D. Clark. The design philosophy of the DARPA internet protocols. In *SIGCOMM '88: Symposium proceedings on Communications architectures and protocols*, pages 106–114, New York, NY, USA, 1988. ACM.

[Coh06] David L. Cohen. Testimony before the committee on the judiciary United States Senate, June 2006. http://netcompetition.org/docs/pronetcomp/Cohen-Testimony-6-14-06.pdf.

[Com07a] Federal Trade Commission. Broadband Connectivity Competition Policy. Staff Report, June 2007. http://www.ftc.gov/reports/broadband/v070000report.pdf.

[Com07b] Cox Communications. Limitations of Service. http://www.cox.com/policy/limitations.asp, November 2007.

[Com08a] Federal Communications Commision. Commision orders Comcast to end discriminatory network management practices. News release, August 2008. http://hraunfoss.fcc.gov/edocs_public/attachmatch/DOC-284286A1.pdf.

[Com08b] Federal Communications Commision. In the Matters of Formal Complaint of Free Press and Public Knowledge Against Comcast Corporation for Secretly Degrading Peer-to-Peer Applications; Broadband Industry Practices Petition of Free Press et al. for Declaratory Ruling that Degrading an Internet Application Violates the FCC's Internet Policy Statement and Does Not Meet an Exception for Reasonable Network Management. Memorandum opinion and order, August 2008. http://hraunfoss.fcc.gov/edocs_public/attachmatch/FCC-08-183A1.pdf.

[Com09a] Comcast. Acceptable Use Policy. http://www.comcast.net/terms/use/, 2009.

[Com09b] Comcast. Network Management Policy. http://www.comcast.net/terms/network/, 2009.

[Com09c] Federal Communications Commission. In the Matters of Formal Complaint of Free Press and Public Knowledge Against Comcast Corporation for Secretly Degrading Peer-to-Peer Applications; Broadband Industry Practices Petition of Free Press et al. for Declaratory Ruling that Degrading an Internet Application Violates the FCC's Internet Policy Statement and Does Not Meet an Exception for Reasonable Network Management. Communication, January 2009. http://hraunfoss.fcc.gov/edocs_public/attachmatch/DOC-288047A1.pdf.

[Com09d] Cox Communications. Congestion Management FAQ. http://www.cox.com/policy/congestionmanagement/, January 2009.

[CT09] Andrew A. Chien and Nut Taesombut. Integrated resource management for lambda-grids: The Distributed Virtual Computer (DVC). *Future Gener. Comput. Syst.*, 25(2):147–152, 2009.

[Dav08] Sashka Davis. Progress Towards Efficient Data Ingestion into iRODS. Technical report, UCSD, 2008. https://www.irods.org/pubs/SCEC_iRODS_Transfer_Report-0902.pdf.

[DFN09] DFN – Deutsches Forschungsnetz. Web page, 2009. http://www.dfn.de/index.php?id=74989&L=2.

[DIK+06] Suchuan Dong, Joseph Insley, Nicholas T. Karonis, Michael E. Papka, Justin Binns, and George Karniadakis. Simulating and visualizing the human arterial system on the teragrid. *Future Gener. Comput. Syst.*, 22(8):1011–1017, 2006.

[dLH09] Cees T. de Laat and Laurin Herr. Ultra high definition media over optical networks (cinegrid). In *Optical Fiber Communication Conference*, page OWK1. Optical Society of America, 2009.

[DOE04] The Office of Science Data-Management Challenge. Report from the DOE Office of Science Data-Management Workshops, March–May 2004.

[DSOB02] Mayur Deshpande, Douglas C. Schmidt, Carlos O'Ryan, and Darrell Brunsch. Design and Performance of Asynchronous Method Handling for CORBA. In *On the Move to Meaningful Internet Systems, 2002 - DOA/CoopIS/ODBASE 2002 Confederated International Conferences DOA, CoopIS and ODBASE 2002*, pages 568–586, London, UK, 2002. Springer-Verlag.

[ea08] Edward Markey et. al. Internet Freedom Preservation Act of 2008. Congress Bill, February 2008. http://www.opencongress.org/bill/110-h5353/show.

[EMP08] Stefan Eilemann, Maxim Makhinya, and Renato Pajarola. Equalizer: A Scalable Parallel Rendering Framework. In *IEEE Transactions on Visualization and Computer Graphics*, 2008.

[ENS05] R. Eccles, B. Nonneck, and D.A. Stacey. Exploring parallel programming knowledge in the novice. *High Performance Computing Systems and Applications, 2005. HPCS 2005. 19th International Symposium on*, pages 97–102, May 2005.

[EWW+07] Thomas Eickermann, Lidia Westphal, Oliver Wäldrich, Wolfgang Ziegler, Christoph Barz, and Markus Pilz. Co-allocating compute and network resources - bandwidth on demand in the Viola testbed. In *Towards Next Generation Grids*, volume V, pages 193–202. Springer US, 2007.

[Far08] David J. Farber. Predicting the Unpredictable - Future Directions in Internetworking and their Implications. Testimony before the Federal Communications Commission, July 2008. `http://www.fcc.gov/broadband_digital_future/072108/farber.pdf`.

[FCL+07] S. Figuerola, N. Ciulli, M. De Leenheer, Y. Demchenko, W. Ziegler, and A. Binczewski on behalf of the PHOSPHORUS consortium. PHOSPHORUS: Single-step on-demand services across multi-domain networks for e-science. In *Proceedings of the European Conference and Exhibition on Optical Communication '07*, 2007.

[FDKKM97] Ian Foster, Jr. David Kohr, Rakesh Krishnaiyer, and Jace Mogill. Remote I/O: fast access to distant storage. In *IOPADS '97: Proceedings of the fifth workshop on I/O in parallel and distributed systems*, pages 14–25, New York, NY, USA, 1997. ACM Press.

[FKT01] Ian Foster, Carl Kesselman, and Steven Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15, 2001.

[Flo03] Sally Floyd. HighSpeed TCP for Large Congestion Windows. Internet draft, work in progress ftp://ftp.rfc-editor.org/in-notes/rfc3649.txt, December 2003.

[FM92] Edward W. Felten and Dylan McNamee. Improving the Performance of Message-Passing Applications by Multithreading. In *Proceedings of the Scalable High Performance Computing Conference SHPCC-92*, pages 84–89, 1992.

[For08] George S. Ford. Testimony before the Federal Communications Commission. Written Testimony, April 2008. `http://www.fcc.gov/broadband_network_management/041708/ford.pdf`.

[FUS09] FUSE: Filesystem in userspace. `http://fuse.sourceforge.net`, 2009.

[GACHJ05] Ruchi Gupta, Saad Ansari, R. Les Cottrell, and Richard Hughes-Jones. Characterization and Evaluation of TCP and UDP-based Transport on Real Networks. In *Third International Workshop on Protocols for Fast Long-Distance Networks*, February 2005.

[GAL+03] Tom Goodale, Gabrielle Allen, Gerd Lanfermann, Joan Masso, Thomas Radke, Ed Seidel, and John Shalf. The Cactus Framework and Toolkit: Design and Applications. In *High Performance Computing for Computational Science – VECPAR 2002*, volume 2565 of *Lecture Notes in Computer Science*, pages 15–36. Springer Berlin / Heidelberg, 2003.

[GAW09] I. J. Grimstead, N. J. Avis, and D. W. Walker. RAVE: the resource-aware visualization environment. *Concurrency and Computation: Practice and Experience*, 21(4):415–448, 2009.

[GEA09] GÉANT2, the high-bandwidth, academic Internet serving Europe's research and education community. Web page, 2009. `http://www.geant2.net/`.

[GG05] Yunhong Gu and Robert Grossman. Supporting Configurable Congestion Control in Data Transport Services. In *Supercomputing 2005*, November 2005.

[GG07] Yunhong Gu and Robert L. Grossman. UDT: UDP-based data transfer for high-speed wide area networks. *Comput. Networks*, 51(7):1777–1799, 2007.

[GG08] Robert Grossman and Yunhong Gu. Data mining using high performance data clouds: experimental studies using sector and sphere. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 920–927, New York, NY, USA, 2008. ACM.

[GHW+04] Andreas Gerndt, Bernd Hentschel, Marc Wolter, Torsten Kuhlen, and Christian Bischof. VIRACOCHA: An Efficient Parallelization Framework for Large-Scale CFD Post-Processing in Virtual Environments. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 50, Washington, DC, USA, 2004. IEEE Computer Society.

[GMM+09] Paola Grosso, Damien Marchal, Jason Maassen, Eric Bernier, Li Xu, and Cees de Laat. Dynamic photonic lightpaths in the StarPlane network. *Future Gener. Comput. Syst.*, 25(2):132–136, 2009.

[GPF09] IBM General Parallel File System, 2009. `http://www.ibm.com/systems/clusters/software/gpfs.html`.

[Gro03] W3C Working Group. SOAP Version 1.2 Usage Scenarios . W3C Note, July 2003. `http://www.w3.org/TR/xmlp-scenarios/`.

[Gro08] TeraGrid Scheduling Working Group. Advance Reservation and Co-Scheduling Report, 2008. `http://www.teragridforum.org/mediawiki/images/c/cd/Schedwg_RsrvCoschedReport.pdf`.

[GRT+06] Chin Guok, David Robertson, Mary Thompson, Jason Lee, Brian Tierney, and William Johnston. Intra and Interdomain Circuit Provisioning Using the OSCARS Reservation System. In *Proceedings of the 3rd International Conference on Broadband Communications, Networks and Systems*, pages 1–8, 2006.

[HAB+06] Andrei Hutanu, Gabrielle Allen, Stephen D. Beck, Petr Holub, Hartmut Kaiser, Archit Kulshrestha, Miloš Liška, Jon MacLaren, Luděk Matyska, Ravi Paruchuri, Steffen Prohaska, Ed Seidel, Brygg Ullmer, and Shalini Venkataraman. Distributed and collaborative visualization of large data sets using high-speed networks. *Future Generation Computer Systems. The International Journal of Grid Computing: Theory, Methods and Applications*, 22(8):1004–1010, 2006.

[HBH+04] Hans-Christian Hege, Daniel Baum, Andrei Hutanu, Andre Merzky, Brygg Ullmer, and Stefan Zachow. CoDiSP - Project web page. `http://www.zib.de/visual/projects/codisp/codisplong.en.html`, 2004.

[HHAM06] Andrei Hutanu, Stephan Hirmer, Gabrielle Allen, and Andre Merzky. Analysis of remote execution models for grid middleware. In *MCG '06: Proceedings of the 4th international workshop on Middleware for grid computing*, page 11, New York, NY, USA, 2006. ACM.

[HHB08] Kyungmin Ham, H.A. Harriett, and L.C Butler. Burning Issues in Tomography Analysis. In *Computing in Science & Engineering*, pages 78–81, 2008.

[HHD04] Eva Hladká, Petr Holub, and Jiří Denemark. An Active Network Architecture: Distributed Computer or Transport Medium. In *3rd International Conference on Networking (ICN'04)*, pages 338–343, Gosier, Guadeloupe, March 2004.

[HHK+05] Hans-Christian Hege, Andrei Hutanu, Ralf Kähler, André Merzky, Thomas Radke, Edward Seidel, and Brygg Ullmer. Progressive Retrieval and Hierarchical Visualization of Large Remote Data. *Scalable Computing: Practice and Experience*, 6(3):57–66, September 2005. `http://www.scpe.org/?a=volume&v=23`.

[HHN+02] Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter D. Kirchner, and James T. Klosowski. Chromium: a stream-processing framework for interactive rendering on clusters. *ACM Trans. Graph.*, 21(3):693–702, 2002.

[HHS+04] Felix Hupfeld, Andrei Hutanu, Thorsten Schütt, Brygg Ullmer, and Stefan Zwierlein. Deliverable D8.7 - Evaluation of Data Management & Visualization Services. GridLab Deliverable, 2004. `http://www.gridlab.org/Resources/Deliverables/D8.7.pdf`.

[HLYD02] Eric He, Jason Leigh, Oliver Yu, and Thomas A. DeFanti. Reliable Blast UDP: Predictable High Performance Bulk Data Transfer. In *CLUSTER '02: Proceedings of the IEEE International Conference on Cluster Computing*, pages 317–324, Washington, DC, USA, 2002. IEEE Computer Society.

[HML+06] Petr Holub, Ludek Matyska, Miloš Liška, Lukás Hejtmánek, Jirí Denemark, Tomás Rebok, Andrei Hutanu, Ravi Paruchuri, Jan Radil, and Eva Hladká. High-definition multimedia for multiparty low-latency interactive communication. *Future Generation Computer Systems*, 22(8):856–861, 2006.

[HP07] John L. Hennessy and David A. Patterson. *Computer Architecture - A Quantitative Approach*. Morgan Kaufmann, fourth edition, 2007.

[HPD09] HPDMnet. Project web page, 2009. `http://www.hpdmnet.net`.

[HPE+07] Andrei Hutanu, Ravi Paruchuri, Daniel Eiland, Miloš Liška, Peter Holub, Steven R. Thorpe, and Yufeng Xin. Uncompressed HD video for collaborative teaching – an experiment. *Collaborative Computing: Networking, Applications and Worksharing, 2007. CollaborateCom 2007. International Conference on*, pages 253–261, Nov. 2007.

[HT03] Tony Hey and Anne Trefethen. *The data deluge: an e-Science perspective*, chapter 36, pages 809–824. John Wiley & Sons Ltd., 2003.

[iGr05] iGrid 2005, the 4th community-driven biennial International Grid. Web page, 2005. `http://www.igrid2005.org/`.

[Ima09] 2009. ImageVis3D: A Real-time Volume Rendering Tool for Large Data. Scientific Computing and Imaging Institute (SCI).

[Int09] Internet2. Web page, 2009. `http://www.internet2.edu/`.

[ION09] Internet2 ION. Web page, 2009. `http://www.internet2.edu/ion/`.

[JD02] Manish Jain and Constantinos Dovrolis. Pathload: A measurement tool for end-to-end available bandwidth. In *In Proceedings of Passive and Active Measurements (PAM) Workshop*, pages 14–25, 2002.

[JGN09] JGN2plus Official Web Site: Advanced Testbed Network for R&D. Web page, 2009. `http://www.jgn.nict.go.jp/english/index.html`.

[Kel03] Tom Kelly. Scalable TCP: improving performance in highspeed wide area networks. *SIGCOMM Comput. Commun. Rev.*, 33(2):83–91, 2003.

[KHJ08] Stephen Kershaw and Richard Hughes-Jones. Transfer of real-time constant bit-rate data over TCP and the effect of alternative congestion control algorithms. In *Sixth International Workshop on Protocols for Fast Long-Distance Networks*, March 2008.

[KHR02] Dina Katabi, Mark Handley, and Charlie Rohrs. Congestion control for high bandwidth-delay product networks. In *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 89–102, New York, NY, USA, 2002. ACM.

[KLE07] Gary Kumfert, James Leek, and Thomas Epperly. Babel Remote Method Invocation. In *Parallel and Distributed Processing Symposium*, pages 1–10, 2007.

[KNOW08] Krzysztof Kurowski, Jarek Nabrzyski, Ariel Oleksiak, and Jan Weglarz. A multicriteria approach to two-level hierarchy scheduling in grids. *J. of Scheduling*, 11(5):371–379, 2008.

[KPB+03] Nicholas T. Karonis, Michael E. Papka, Justin Binns, John Bresnahan, Joseph A. Insley, David Jones, and Joseph M. Link. High-resolution remote rendering of large datasets in a collaborative environment. *Future Generation Computer Systems*, 19(6):909–917, 2003.

[KPHH05] R. Kaehler, S. Prohaska, A. Hutanu, and H.-C. Hege. Visualization of time-dependent remote adaptive mesh refinement data. *Visualization, 2005. VIS 05. IEEE*, pages 175–182, Oct. 2005.

[KTO07] Kazumi Kumazoe, Masato Tsuru, and Yuji Oie. Performance of high-speed transport protocols coexisting on a long distance 10-Gbps testbed network. In *Proceedings of the First International Conference on Networks for Grid Applications*, 2007.

[KTS06] Qian Zhang Kun Tan, Jingmin Song and Murari Sridharan. Compound TCP: A Scalable and TCP-Friendly Congestion Control for High-speed Networks. In *Fourth International Workshop on Protocols for Fast Long-Distance Networks*, 2006.

[KYGM07] D. Katramatos, Dantong Yu, B. Gibbard, and S. McKee. The TeraPaths Testbed: Exploring End-to-End Network QoS. In *Testbeds and Research Infrastructure for the Development of Networks and Communities, 2007. TridentCom 2007. 3rd International Conference on*, pages 1–7, May 2007.

[LAQ+08] Douglas J. Leith, Lachlan L. H. Andrew, Tom Quetchenbach, Robert N. Shorten, Injong Rhee, and Lisong Xu. Experimental Evaluation of Delay/Loss-based TCP Congestion Control Algorithms. In *Sixth International Workshop on Protocols for Fast Long-Distance Networks*, March 2008.

[Lei09] Tom Leighton. Improving performance on the Internet. *Commun. ACM*, 52(2):44–51, 2009.

[LGT+01] J. Lee, D. Gunter, B. Tierney, B. Allcock, J. Bester, J. Bresnahan, and S. Tuecke. Applied Techniques for High Bandwidth Data Transfers across Wide Area Networks. In *Proc. International Conference on Computing in High Energy and Nuclear Physics (CHEP 01)*, Beijing, China, September 2001.

[LH02] Eric J. Luke and Charles D. Hansen. Semotus Visum: a flexible remote visualization framework. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 61–68, Washington, DC, USA, 2002. IEEE Computer Society.

[LH08] Milos Liska and Petr Holub. CoUniverse: Framework for Building Self-organizing Collaborative Environments Using Extreme-Bandwidth Media Applications. In Eduardo César, Michael Alexander, Achim Streit, Jesper Larsson Träff, Christophe Cérin, Andreas Knüpfer, Dieter Kranzlmüller, and Shantenu Jha, editors, *Euro-Par Workshops*, volume 5415 of *Lecture Notes in Computer Science*, pages 339–351. Springer, 2008.

[LON09] LONI: Louisiana Optical Network Initiative. Web page, 2009. `http://www.loni.org/`.

[LQD05] D. Lu, Y. Qiao, and P. A. Dinda. Characterizing and Predicting TCP Throughput on the Wide Area Network. In *Proc. IEEE International Conference on Distributed Computing Systems (ICDCS 05)*, pages 414–424, June 2005.

[LRA+06] Jonghyun Lee, R. Ross, S. Atchley, M. Beck, and R. Thakur. MPI-IO/L: efficient remote I/O for MPI-IO via logistical networking. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, page 10, April 2006.

[LRT+04] Jonghyun Lee, R. Ross, R. Thakur, Xiaosong Ma, and M. Winslett. RFS: efficient and flexible remote file access for MPI-IO. In *CLUSTER '04: Proceedings of the 2004 IEEE International Conference on Cluster Computing*, pages 71–81, Washington, DC, USA, 2004. IEEE Computer Society.

[LS88] B. Liskov and L. Shrira. Promises: linguistic support for efficient asynchronous procedure calls in distributed systems. *SIGPLAN Not.*, 23(7):260–267, 1988.

[Lus09] Lustre File System, 2009. `www.lustre.org`.

[Mac07] Jon MacLaren. HARC: The Highly-Available Resource Co-allocator. In *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, volume 4804/2009 of *Lecture Notes in Computer Science*, pages 1385–1402. Springer Berlin / Heidelberg, 2007.

[MCN09] MCNC. Web page, 2009. `https://www.mcnc.org/`.

[MGF+04] S. Mascolo, L. A. Grieco, R. Ferorelli, P. Camarda, and G. Piscitelli. Performance evaluation of Westwood+ TCP congestion control. *Perform. Eval.*, 55(1-2):93–111, 2004.

[MGYC06] Joe Mambretti, Rachel Gold, Fei Yeh, and Jim Chen. Amroeba: computational astrophysics modeling enabled by dynamic lambda switching. *Future Gener. Comput. Syst.*, 22(8):949–954, 2006.

[MRH+07] Kwan-Liu Ma, Robert Ross, Jian Huang, Greg Humphreys, Nelson Max, Kenneth Moreland, John D. Owens, and Han-Wei Shen. Ultra-Scale Visualization: Research and Education. *Journal of Physics*, 78, June 2007. (Proceedings of SciDAC 2007 Conference).

[MSHC99] K. Mueller, N. Shareef, J. Huang, and R. Crawfis. IBR-Assisted Volume Rendering. In *Proc. IEEE Visualization*, pages 5–8, 1999.

[MWY+09] Kwan-Liu Ma, Chaoli Wang, Hongfeng Yu, Kenneth Moreland, Jian Huang, and Rob Ross. Next-Generation Visualization Technologies: Enabling Discoveries at Extreme Scale. *SciDAC Review*, 12:12–21, February 2009.

[N. 08] N. S. V. Rao, W. R. Wing, S. Hicks, S. Poole, F. Denap, S. M. Carter, and Q. Wu. Ultrascience net: research testbed for high-performance networking. Proceedings of International Symposium on Computer and Sensor Network Systems, April 2008.

[NGBS+97] Henrik Frystyk Nielsen, James Gettys, Anselm Baird-Smith, Eric Prud'hommeaux, Håkon Wium Lie, and Chris Lilley. Network performance effects of HTTP/1.1, CSS1, and PNG. In *SIGCOMM '97: Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 155–166, New York, NY, USA, 1997. ACM Press.

[NLR09] National LambdaRail. Web page, 2009. `http://www.nlr.net/`.

[NOL02] E. Nallipogu, F. Ozguner, and M. Lauria. Improving the Throughput of Remote Storage Access through Pipelining. In *Third International Workshop on Grid Computing*, 2002.

[NSF09] Strategies for Remote Visualization on a Dynamically Configurable Testbed. Award Web Page, September 2009. `http://www.nsf.gov/awardsearch/showAward.do?AwardNumber=0947825`.

[OMG91] OMG. The Common Object Request Broker: Architecture and Specification. Technical report, Object Management Group and X/Open. OMG Document Number 91.12.1, 1991.

[OSA+08] C. D. Ott, E. Schnetter, G. Allen, E. Seidel, J. Tao, and B. Zink. A case study for petascale applications in astrophysics: simulating gamma-ray bursts. In *MG '08: Proceedings of the 15th ACM Mardi Gras conference*, pages 1–9, New York, NY, USA, 2008. ACM.

[Ou08] George Ou. Why Network Management is Essential to the Internet. Testimony before the Federal Communications Commission, April 2008. `http://www.fcc.gov/broadband_network_management/041708/ou-stmt.pdf`.

[PH05] Steffen Prohaska and Andrei Hutanu. Remote Data Access for Interactive Visualization. In *Proceedings of 13th Annual Mardi Gras Conference: Frontiers of Grid Applications and Technologies*, pages 17–22, 2005.

[PHKH04] S. Prohaska, A. Hutanu, R. Kahler, and H.-C. Hege. Interactive exploration of large remote micro-CT scans. *Visualization, 2004. IEEE*, pages 345–352, Oct. 2004.

[PIO09] PIONIER – Polish Optical Internet. Web page, 2009. `http://www.pionier.gov.pl/eindex.html`.

[PSH04] Polyvios Pratikakis, Jaime Spacco, and Michael Hicks. Transparent proxies for java futures. In *OOPSLA '04: Proceedings of the 19th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 206–223, New York, NY, USA, 2004. ACM Press.

[RHH85] Jr. Robert H. Halstead. MULTILISP: a language for concurrent symbolic computation. *ACM Trans. Program. Lang. Syst.*, 7(4):501–538, 1985.

[RJH+09] Luc Renambot, Byungil Jeong, Hyejung Hur, Andrew Johnson, and Jason Leigh. Enabling high resolution collaborative visualization in display rich virtual organizations. *Future Gener. Comput. Syst.*, 25(2):161–168, 2009.

[RRB+03] Vinay J. Ribeiro, Rudolf H. Riedi, Richard G. Baraniuk, Jiri Navratil, and Les Cottrell. pathchirp: Efficient available bandwidth estimation for network paths. In *In Passive and Active Measurement Workshop*, 2003.

[RSFWH98] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. Virtual Network Computing. *IEEE Internet Computing*, 2(1):33–38, 1998.

[RWCW05] N.S.V. Rao, W.R. Wing, S.M. Carter, and Q. Wu. Ultrascience net: network testbed for large-scale science applications. *Communications Magazine, IEEE*, 43(11):S12–S17, Nov. 2005.

[RX05] Injong Rhee and Lisong Xu. CUBIC: A New TCP-Friendly High-Speed TCP variant. In *Third International Workshop on Protocols for Fast Long-Distance Networks*, February 2005.

[RYW+08] Nageswara S. V. Rao, Weikuan Yu, William R. Wing, Stephen W. Poole, and Jeffrey S. Vetter. Wide-area performance profiling of 10GigE and InfiniBand technologies. In *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press.

[SB03] J. Shalf and EW Bethel. The grid and future visualization system architectures. *Computer Graphics and Applications, IEEE*, 23(2):6–9, 2003.

[SBG00] H. Sivakumar, S. Bailey, and R. L. Grossman. PSockets: The case for Application-level Network Striping for Data intensive Applications using High Speed Wide Area Networks. In *Proc. IEEE Super Computing Conference (SC 00)*, pages 63–63, Texas, USA, November 2000.

[SBSdL06] M. Scarpa, R. G. Belleman, P. M. A. Sloot, and C. T. A. M. de Laat. Highly interactive distributed visualization. *Future Gener. Comput. Syst.*, 22(8):896–900, 2006.

[SCD+03] Larry L. Smarr, Andrew A. Chien, Tom DeFanti, Jason Leigh, and Philip M. Papadopoulos. The optiputer. *Commun. ACM*, 46(11):58–67, 2003.

[Sch97] Douglas C. Schmidt. Applying Patterns and Frameworks to Develop Object-Oriented Communication Software. In *Handbook of Programming Languages*, volume I. MacMillan Computer Publishing, 1997.

[SG86] Robert W. Scheifler and Jim Gettys. The X window system. *ACM Trans. Graph.*, 5(2):79–109, 1986.

[SGP+09] Larry Smarr, Paul Gilna, Phil Papadopoulos, Thomas A. DeFanti, Greg Hidley, John Wooley, E. Virginia Armbrust, Forest Rohwer, and Eric Frost. Building an optiplanet collaboratory to support microbial metagenomics. *Future Gener. Comput. Syst.*, 25(2):124–131, 2009.

[SHTS01] Mitsuhisa Sato, Motonari Hirano, Yoshio Tanaka, and Satoshi Sekiguchi. OmniRPC: A Grid RPC Facility for Cluster and Global Computing in OpenMP. In *WOMPAT '01: Proceedings of the International Workshop on OpenMP Applications and Tools*, pages 130–136, London, UK, 2001. Springer-Verlag.

[Sil05] SGI OpenGL Vizserver 3.5, Visualization and Collaboration. White Paper, 2005. `http://www.sgi.com/pdfs/3263.pdf`.

[SKF+09] Daisuke Shirai, Tetsuo Kawano, Tatsuya Fujii, Kunitake Kaneko, Naohisa Ohta, Sadayasu Ono, Sachine Arai, and Terukazu Ogoshi. Real time switching and streaming transmission of uncompressed 4K motion pictures. *Future Gener. Comput. Syst.*, 25(2):192–197, 2009.

[SMHS04] T. Schütt, A. Merzky, A. Hutanu, and F. Schintke. Remote partial file access using compact pattern descriptions. In *CCGRID '04: Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid*, pages 482–489, Washington, DC, USA, 2004. IEEE Computer Society.

[Smi09] Jonathan M. Smith. Fighting physics: a tough battle. *Communications of the ACM*, 52(7):60–65, 2009.

[SNM+02] Keith Seymour, Hidemoto Nakada, Satoshi Matsuoka, Jack Dongarra, Craig Lee, and Henri Casanova. Overview of GridRPC: A Remote Procedure Call API for Grid Computing. In *GRID '02: Proceedings of the Third International Workshop on Grid Computing*, pages 274–278, London, UK, 2002. Springer-Verlag.

[SNS02] Satoshi Shirasuna, Hidemoto Nakada, and Satoshi Sekiguchi. Evaluating Web Services Based Implementations of GridRPC. In *HPDC '02: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing*, page 237, Washington, DC, USA, 2002. IEEE Computer Society.

[SPM06] Jonathan Strasser, Valerio Pascucci, and Kwan-Liu Ma. Multi-Layered Image Caching for Distributed Rendering of Large Multiresolution Data. In Alan Heirich, Bruno Raffin, and Luis Paulo dos Santos, editors, *Proceedings of Eurographics Symposium on Parallel Graphics and Visualization*, pages 171–177, May 2006.

[STBK08] I. Suslu, F. Turkmen, M. Balman, and T. Kosar. Choosing between remote I/O versus staging in large scale distributed applications. In *Proceedings of ISCA 21st Int. Conference on Parallel and Distributed Computing and Applications, PDCCS*, 2008.

[SWH05] Detlev Stalling, Malte Westerhoff, and Hans-Christian Hege. Amira: A highly interactive system for visual data analysis. In Charles D. Hansen and Christopher R. Johnson, editors, *The Visualization Handbook*, chapter 38, pages 749–767. Elsevier, 2005.

[TBSL01] Douglas Thain, Jim Basney, Se-Chang Son, and Miron Livny. The Kangaroo Approach to Data Movement on the Grid, 2001.

[TC07] Nut Taesombut and Andrew A. Chien. Evaluating network information models on resource efficiency and application performance in lambda-grids. In *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, pages 1–12, New York, NY, USA, 2007. ACM.

[Ter09] Teragrid, web page, documentation, publications: `http://www.teragrid.org/userinfo/data/vis/vis_portal.php`, 2009.

[TH08] Cornelius Toole, Jr. and Andrei Hutanu. Network flow based resource brokering and optimization techniques for distributed data streaming over optical networks. In *MG '08: Proceedings of the 15th ACM Mardi Gras conference*, pages 1–8, New York, NY, USA, 2008. ACM.

[THN+06] Atsuko Takefusa, Michiaki Hayashi, Naohide Nagatsu, Hidemoto Nakada, Tomohiro Kudoh, Takahiro Miyamoto, Tomohiro Otani, Hideaki Tanaka, Masatoshi Suzuki, Yasunori Sameshima, Wataru Imajuku, Masahiko Jinno, Yoshihiro Takigawa, Shuichi Okamoto, Yoshio Tanaka, and Satoshi Sekiguchi. G-lambda: coordination of a grid scheduler and lambda path service over GMPLS. *Future Gener. Comput. Syst.*, 22(8):868–875, 2006.

[TL05] Douglas Thain and Miron Livny. Parrot: An Application Environment for Data-Intensive Computing. *Scalable Computing: Practice and Experience*, 6(3):9–18, 2005.

[TLC+99] Brian L. Tierney, Jason Lee, Brian Crowley, Mason Holding, Jeremy Hylton, and Fred L. Drake Jr. A Network-Aware Distributed Storage Cache for Data Intensive Environments. In *HPDC '99: Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing*, page 33, Washington, DC, USA, 1999. IEEE Computer Society.

[TMK+07] Ryousei Takano, Motohiko Matsuda, Tomohiro Kudoh, Yuetsu Kodama, Fumihiro Okazaki, and Yutaka Ishikawa. Effects of packet pacing for MPI programs in a Grid environment. In *CLUSTER '07: Proceedings of the 2007 IEEE International Conference on Cluster Computing*, pages 382–391, Washington, DC, USA, 2007. IEEE Computer Society.

[TMKE06] Franco Travostino, Joe Mambretti, and Gigi Karmous-Edwards, editors. *Grid Networks: Enabling Grids with Advanced Communication Technology*. Wiley, 2006.

[TTL05] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the Condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[TTNS04] Yoshio Tanaka, Hiroshi Takemiya, Hidemoto Nakada, and Satoshi Sekiguchi. Design, Implementation and Performance Evaluation of GridRPC Programming Middleware for a Large-Scale Computational Grid. In *GRID '04: Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*, pages 298–305, Washington, DC, USA, 2004. IEEE Computer Society.

[USJ+08] Brygg Ullmer, Rajesh Sankaran, Srikanth Jandhyala, Blake Tregre, Cornelius Toole, Karun Kallakuri, Christopher Laan, Matthew Hess, Farid Harhad, Urban Wiggins, and

Shining Sun. Tangible menus and interaction trays: core tangibles for common physical/digital activities. In Albrecht Schmidt, Hans Gellersen, Elise van den Hoven, Ali Mazalek, Paul Holleis, and Nicolas Villar, editors, *Tangible and Embedded Interaction*, pages 209–212. ACM, 2008.

[VBLS09] Venkatram Vishwanath, Robert Burns, Jason Leigh, and Michael Seablom. Accelerating tropical cyclone analysis using lambdaram, a distributed data cache over wide-area ultra-fast networks. *Future Gener. Comput. Syst.*, 25(2):184–191, 2009.

[vECGS92] Thorsten von Eicken, David E. Culler, Seth Copen Goldstein, and Klaus Erik Schauser. Active messages: a mechanism for integrated communication and computation. In *ISCA '92: Proceedings of the 19th annual international symposium on Computer architecture*, pages 256–266, New York, NY, USA, 1992. ACM Press.

[vS08] Barbara van Schewick. Official testimony. Federal Communications Commission Second Public En Banc Hearing on Broadband Network Management Practices, April 2008. `http://www.fcc.gov/broadband_network_management/041708/vanschewick-written.pdf`.

[vSF09] Barbara van Schewick and David Farber. Point/Counterpoint: Network neutrality nuances. *Commun. ACM*, 52(2):31–37, 2009.

[VZL08] V. Vishwanath, L.D. Zuck, and J. Leigh. Specification and verification of lambdaram-a wide-area distributed cache for high performance computing. In *Formal Methods and Models for Co-Design, 2008. MEMOCODE 2008. 6th ACM/IEEE International Conference on*, pages 187–198, June 2008.

[Wal08] Scott Wallsten. Official testimony. Federal Communications Commission Third Public En Banc Hearing on Broadband Network Management Practices, July 2008. `http://www.fcc.gov/broadband_digital_future/072108/wallsten.pdf`.

[WC95] R.W. Watson and R.A. Coyne. The parallel I/O architecture of the high-performance storage system (HPSS). *Mass Storage Systems, IEEE Symposium on*, 0:27, 1995.

[WC04] Ryan X. Wu and Andrew A. Chien. GTP: Group Transport Protocol for Lambda-Grids. In *Cluster Computing and the Grid*, pages 228–238, April 2004.

[WFN90] E.F. Walker, R. Floyd, and P. Neves. Asynchronous remote operation execution in distributed systems. *Proceedings of the 10th International Conference on Distributed Computing Systems Distributed Computing Systems*, pages 253–259, 1990.

[WJLH06] David X. Wei, Cheng Jin, Steven H. Low, and Sanjay Hegde. FAST TCP: motivation, architecture, algorithms, performance. *IEEE/ACM Trans. Netw.*, 14(6):1246–1259, 2006.

[Wor09] Jenna Wortham. Customers Angered as iPhones Overload AT&T. New York Times, September 2009. `http://www.nytimes.com/2009/09/03/technology/companies/03att.html`.

[XHR04] Lisong Xu, Khaled Harfoush, and Injong Rhee. Binary Increase Congestion Control for Fast Long Distance Networks. In *INFOCOM 2004*, volume 4, pages 2514–2524, March 2004.

[XLH+05] Chaoyue Xiong, Jason Leigh, Eric He, Venkatram Vishwanath, Tadao Murata, Luc Renambot, and Thomas A. DeFanti. LambdaStream – a Data Transport Protocol for Streaming Network-intensive Applications over Photonic Networks. In *Third International Workshop on Protocols for Fast Long-Distance Networks*, February 2005.

[YK03] K.C. Yeung and P.H.J. Kelly. Optimising Java RMI programs by communication restructuring. In *Proceedings of the 4th Middleware Conference*, pages 324–343. Springer Berlin / Heidelberg, 2003.

[Yoo08] Christopher S. Yoo. Hearing on Broadband Network Management Practices. Written Testimony, February 2008. `http://www.fcc.gov/broadband_network_management/022508/yoo.pdf`.

[YSK08] E. Yildirim, I. H. Suslu, and T. Kosar. Which Network Measurement Tool is Right for You? A Multidimensional Comparison Study. In *Proc. IEEE International Conference on Grid Computing (GRID 08)*, sep 2008.

[Zha08] Charles Zhang. *OptiStore: An On-Demand Data Processing Middleware for Very Large Scale Interactive Visualization*. PhD thesis, Computer Science, University of Illinois at Chicago, 2008.

[ZLD+03] Chong Zhang, Jason Leigh, Thomas A. DeFanti, Marco Mazzucco, and Robert Grossman. TeraScope: distributed visual data mining of terascale data sets over photonic networks. *Future Generation Computer Systems*, 19(6):935–943, 2003.

[ZSH05] Hans-Florian Zeilhofer, Robert Sader, and Hans-Christian Hege. CoDiSP - Collaborative Distributed Surgery Planning. Project report(Schlussbericht), March 2005. `http://webdoc.sub.gwdg.de/ebook/ah/dfn/codisp.pdf`.

[ZVR+05] Xuan Zheng, M. Veeraraghavan, N.S.V. Rao, Qishi Wu, and Mengxia Zhu. CHEETAH: circuit-switched high-speed end-to-end transport architecture testbed. *Communications Magazine, IEEE*, 43(8):s11–s17, Aug. 2005.

[ZWRI07] Mengxia Zhu, Qishi Wu, Nageswara S. V. Rao, and Sitharama Iyengar. Optimal pipeline decomposition and adaptive network mapping to support distributed remote visualization. *J. Parallel Distrib. Comput.*, 67(8):947–956, 2007.

Vita

Andrei Huţanu was born in Bucharest, Romania, in 1979. He graduated in 2002 from "Politehnica" University of Bucharest with an Engineer Diploma in Computer Science, with the final two years of his studies done as an exchange student at "Freie" University of Berlin, Germany. Between 2000 and 2002 he also worked as a student employee in the Scientific Optimization Department of Zuse Institute Berlin.

After graduating, he joined the Scientific Visualization Department of Zuse Institute Berlin, where he worked two years as a Research Employee. In 2004 he joined Louisiana State University and has since been working as an IT Analyst at the Center for Computation & Technology. In 2006 he enrolled as a part-time student in computer science at Louisiana State University to pursue the doctoral degree.

He co-authored 20 peer-reviewed articles and participated in over 15 major technical demonstrations. He contributed to 7 research projects, participated to 2 panels and reviewed articles for 8 conferences.

He led the visualization team part of the team that won SCALE 2009 (The second IEEE International Scalable Computing Challenge at CCGrid 2009) by demonstrating a prototype of an end-to-end problem solving system combining numerical simulation and distributed visualization.

He is the main author, co-PI and leader of the technical team of the "Strategies for Remote Visualization on a Dynamically Configurable Testbed" project that has been awarded with $299,447 by NSF and is senior investigator in 4 other funded research projects.

His research is in distributed computing and high-speed network applications with emphasis on distributed visualization, data management and collaborative applications. He is also interested in system design and software engineering, and design of large, complex applications as well as the exploration future needs of high end scientific applications.

His most significant publications are:

- A. Hutanu, J. Ge, C. Toole, R. Kooima, B. Ullmer, G. Allen: *"eaviv: Network-aware interactive visualization of large datasets"*, submitted to IEEE Computer Graphics and Applications

- T. Kosar, A. Hutanu, J. MacLaren, D. Thain: *"Coordination of Access to Large-scale Datasets in Distributed Environments"* to appear in Scientific Data Management: Challenges, Technology, and Deployment, Editors: A. Shoshani and D. Rotem, CRC Press/Taylor and Francis Books, 2009.

- A. Hutanu, E. Schnetter, W. Benger, E. Bentivegna, A. Clary, P. Diener, J. Ge, R. Kooima, O. Korobkin, K. Liu, F. Loffler, R. Paruchuri, J. Tao, C. Toole, A. Yates, G. Allen: *"Large-scale Problem Solving Using Automatic Code Generation and Distributed Visualization"*, CCT TR Series, 2009

- A. Hutanu, G. Allen: *"A case for application-level control of network resources"*, in Proceedings of CESNET Conference 2008, pp 69–76

- A. Hutanu, M. Liška, P. Holub, R. Paruchuri, D. Eiland, S. Thorpe, Y. Xin: *"Uncompressed HD video for collaborative teaching - an experiment"*, in Proceedings of CollaborateCom 2007, pp. 253–261

- A. Hutanu, G. Allen, S. D. Beck, P. Holub, H. Kaiser, A. Kulshrestha, M. Liška, J. MacLaren, L. Matyska, R. Paruchuri, S. Prohaska, E. Seidel, B. Ullmer, S. Venkataraman: *"Distributed and collaborative visualization of large data sets using high-speed networks"* Future Generation Computer Systems: The International Journal of Grid Computing: Theory, Methods and Applications, Volume 22, Issue 8, pp. 1004–1010 (2006)

- *EnLIGHTened:* S. Thorpe, L. Battestilli, G. Karmous-Edwards, A. Hutanu, J. MacLaren, J. Mambretti, J. Moore, K. S. Sundar, Y. Xin; *G-lambda:* A. Takefusa, M. Hayashi, A. Hirano, S. Okamoto, T. Kudoh, T. Miyamoto, Y. Tsukishima, T. Otani, H. Nakada, H. Tanaka, A. Taniguchi, Y. Sameshima, M. Jinno: *"G-lambda and EnLIGHTened: Wrapped In Middleware Co-allocating Compute and Network Resources Across Japan and the US"*, in Proceedings of GridNets 2007, pp. 1–8

- A. Hutanu, S. Hirmer, G. Allen, A. Merzky: *"Analysis of Remote Execution Models for Grid Middleware"* in Proceedings of the 4th International Workshop on Middleware for Grid Computing (MGC 2006), pp. 62–67

- H.-C. Hege, A. Hutanu, R. Kähler, A. Merzky, T. Radke, E. Seidel, B. Ullmer: *"Progressive Retrieval and Hierarchical Visualization of Large Remote Data"* in Scalable Computing: Practice and Experience: Volume 6, No. 3, pp. 57–66

- S. Prohaska, A. Hutanu, R. Kähler, H.-C. Hege: *"Interactive Exploration of Large Remote Micro-CT Scans"* in Proc. IEEE Visualization 2004, H. Rushmeier, J. J. van Wijk, G. Turk (eds.), Austin, Texas, pp. 345–352, (2004)