Channel Set Adaptation: Scalable and Adaptive Streaming for Non-Linear Media

David Gotz

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2006

Approved by:

Ketan Mayer-Patel, Advisor

Leonard McMillan, Reader

F. Donelson Smith, Reader

Dinesh Manocha, Committee Member

Daniel Aliaga, Committee Member

ABSTRACT
DAVID GOTZ: Channel Set Adaptation: Scalable and Adaptive Streaming for Non-Linear Media.
(Under the direction of Ketan Mayer-Patel.)

Streaming of linear media objects, such as audio and video, has become ubiquitous on today's Internet. Large groups of users regularly tune in to a wide variety of online programming, including radio shows, sports events, and news coverage. However, non-linear media objects, such as large 3D computer graphics models and visualization databases, have proven more difficult to stream due to their interactive nature. In this dissertation, I present a complete framework for the efficient streaming of non-linear datasets to large user groups.

The framework has several components. First, I present the *Representation Graph*, an abstract data representation for expressing the semantic and syntactic relationships between elements of information in a non-linear multimedia database.

I then present a computational model for achieving multidimensional adaptation. The model is based on a spatially defined utility metric that allows applications to mathematically express trade-offs between different dimensions of adaptivity.

The representation graph and adaptation model serve as the foundation for the *Generic Adaptation Library* (GAL). GAL defines a layered design for non-linear media applications and provides an implementation for the middle adaptation layer.

The representation graph, adaptation model, and GAL can be combined to support adaptive non-linear streaming. I evaluate the performance of an experimental prototype based on these principles and show that they can effectively support the adaptive requirements of dynamic and interactive access to non-linear media.

I also define *Channel Set Adaptation* (CSA), an architecture for scalable delivery of non-linear media. CSA maps concepts from the representation graph to a scalable subscription-based network model. CSA provides, in the ideal case, an infinitely scalable streaming solution for non-linear media applications. I include a evaluation of CSA's performance on both multicast and broadcast networks.

In addition, I develop a performance model based on results from the experimental evaluation. The performance model highlights several key properties of the underlying communication model that are most important to CSA performance.

ACKNOWLEDGMENTS

This document represents the end of a long and windy road through my graduate studies, my formal education, and my sheltered life in the halls of academia. As I reach the end, I find myself upon a hill. Unfolding before me, I can see the open landscape of my future, waiting to be conquered.

However, before I progress into the next stage of my life, I would like to take a moment to turn around and look back at where I've been, what I've attempted to achieve, and who has helped me along the way.

When I first arrived at the University of North Carolina at Chapel Hill, I was unsure of my own desires. I did not know what I was looking for out of my time here. With the strong support of my loving parents Ben and Bernice, I was simply here to learn, to experience, and to grow. Thanks to there example, I have within me a respect for knowledge, a need to understand how things work, and that respect has led me to where I now stand. Mom and Dad: For all of your love, kindness, and support, *thank you.*

Shortly after my arrival, the most important event to take place during my graduate career occurred: I met my lovely wife Anne without whom none of my achievements would be possible. Her love and encouragement, in good times and in bad, have been an anchor in the ups and downs that come with any significant undertaking. Her family adopted me as one of their own, and for this I owe them a debt that can not be repaid. Anne: For your family's kindness, your unending dedication, and your unconditional love, *thank you.*

Another critical moment occurred near the end of my second year in Chapel Hill, as I was completing my Master of Science degree and beginning to interview for positions in industry. It was then that Dr. Dinesh Manocha approached me in the hallway to ask me about my plans and to encourage me to stay at the University. His simple act proved so timely and influential that I firmly believe I would not be here today if he had not done it. Dinesh: For your support and guidance over the years, *thank you.*

Shortly after choosing to enter the PhD program and remain at UNC, I began working with Dr. Ketan Mayer-Patel. At first, it was a loose association as I continued to work under the direction of Dinesh. However, as time progressed, my research became closer and closer to Ketan's interests until he officially became my adviser several months before my proposal. From that point forward, Ketan has been at my side for every major hurdle, supporting me when I needed his help and encouraging me to stand on my own when I needed to most. Ketan: For your mentorship and advice, *thank you*.

For the rest of my committee, Daniel, Don, and Leonard: Thank you all for your advice and discussions. To my family and friends, both at UNC and beyond, thank you for making me who I am. I could not have reached the top of this hill without you.

And now, as I take my next step forward to the hill beyond, let me say one last time to all those who have helped me travel this far: *Thank You*.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ACS	Active Channel Set
ALM	Application Layer Multicast
API	Application Programming Interface
CAD	Computer-Aided Design
CSA	Channel Set Adaptation
DCT	Discrete Cosine Transform
FFT	Fast Fourier Transform
GAL	The Generic Adaptation Library
GIS	Geographic Information Systems
HLODs	Hierarchical Levels of Detail
IBR	Image-Based Rendering
MPC	Multiple Predictor Coding
RG	Representation Graph
RLM	Receiver-driven Layered Multicast
SOI	Sea of Images
SUM	The Summed Utility Metric
SWIM	Streaming Walk-throughs of Image-based Models
UCR	Utility-Cost Ratio

Chapter 1

Introduction

Two pronounced trends have become evident over the the past decade of computing history. First, digital media techniques have become integral to an enormous number of applications. Ranging from feature films to scientific endeavors, digital media has changed the way we work, relax, and learn. In the ten years since Toy Story became the first feature length computer animated movie in 1995 (Rickitt, 2000), computer graphics have become a staple of Hollywood films for everything from special effects to synthetic characters (Kerlow, 2003).

The video game market has seen similar growth as console makers such as Nintendo, Sony and Microsoft introduce generation after generation of new gaming platforms (Black, 2001). Similarly, game developers have been creating games that set new standards for visual realism and interactive game play.

The digital media movement extends far beyond interactive computer graphics. Personal media devices, such as digital cameras and Apple's iPod digital music player, have been among the most popular products in consumer electronics sales (Salkever, 2003). The popularity of these devices has led to entirely new business models in both the photography and music industries (Stice, 2005). The products have also led to several new challenges, ranging from digital rights management to storing and accessing massive personal digital media libraries.

Digital media has also had a dramatic impact on business and science. From digital documents to complex scientific visualization techniques, digital technology has become an essential component in everything from oil exploration to real estate to medicine (Kharif, 2002).

At the same time, a second trend in computing has had a similarly large impact on our way of life. The proliferation of broadband data networking has had a profound

influence on how we value location, information, and communication. Since the dawn of the Internet, new communication and data dissemination tools have changed nearly every facet of life, from military communications to road maps (Wildstrom, 2005).

The past decade has seen a swift progression of technology from analog telephone modems to broadband network connectivity using DSL and cable modems. These changes have ushered in an era of widely available high-speed digital communication capabilities. Wireless network technology is now commonly available at coffee shops, hotels, and airports. Cellular phone networks now provide high-speed data access to their networks via third-generation mobile phone technology (Kharif, 2005).

The growing ubiquity of broadband technology has opened to door to a host of new applications. Massive libraries of information have been digitized and made available on-line, allowing users to access academic papers, rare artifacts, and historical archives from the comfort of their home. In depth news coverage is available on-line from nearly every media outlet, from minor newspapers to major television stations. Telecommuting is now commonplace among information technology professionals (Rendleman, 2002).

1.1 Digital Media Streaming

At the intersection of these two trends, where digital media and broadband networking meet, there is the broad application space that combines the power of both technologies. Included in this application space is digital media streaming: a technique for efficiently distributing large media datasets to interested receivers.

Unlike traditional downloading of data, which requires an entire file to be received before it can be utilized, streaming employs continuous transmission to allow immediate access to received information. Within a typical network-based application, a recipient first initiates a download and stores a large block of data in a local disk or memory space. Only then, after the entire data file has been locally stored, can the application make use of the data in its operations.

In contrast, recipients in a streaming application receive a continuous flow of data which has been carefully arranged to allow the application to make use of the data as it arrives, rather than waiting for the data to arrive in its entirety. The streaming transmission model allows applications to utilize data that arrives "just in time" to meet application-level requirements.

Streaming has been widely deployed to support access to multimedia objects because these objects are typically large in size and multimedia applications benefit greatly from

the reduced access time afforded by the technique. In addition, the critical role played by time in many media applications means that any errors or omissions of data that may occur during the streaming process are immediately obviated by newly arriving data.

Together, these properties make multimedia an ideal candidate to benefit from streaming technologies. Today, radio program streams are available on-line from a variety of sources ranging from major market music broadcasts to small college radio stations. Video streams from CSPAN to live Major League Baseball telecasts are received by legions of on-line consumers.

1.2 Linear and Non-Linear Media

Despite the technological and commercial successes of digital media streaming, the application space remains quite restricted. To date, media streaming has largely been limited to *linear media*. Linear media objects, such as audio and video, consist of data arranged in a fixed and linear ordering. For example, video consists of a linear sequence of frames arranged along the time dimension. Similarly, audio consists of a linear sequence of sound samples. Every user that accesses a linear media stream receives the same flow of information.

The dominance of linear media in the context of on-line streaming matches the long time dominance of linearity in more traditional media, including books, film, and television. Linear media is inherently well suited for large audiences. The linear nature of theater, for example, is what allows entire audiences to be satisfied by observing a common stage performance.

However, recent advances in computing and interactive technology have led to the growing importance of *non-linear media*. Non-linear media objects, such as video games, interactive visualizations, and virtual environments, provide individual data orderings to each user in response to their local requirements and interactions. Non-linear objects are therefore best suited for individual interaction. It is the non-linearity of video games, for example, which makes the idea of the same theater crowd gathering to play a video game seem so unreasonable.

Linear and non-linear media are fundamentally different in the experience they provide to consumers. Non-linear media experiences require unique presentations to each participating user. For example, every user of an interactive visualization is presented with a different flow of information in response to their individual interactions.

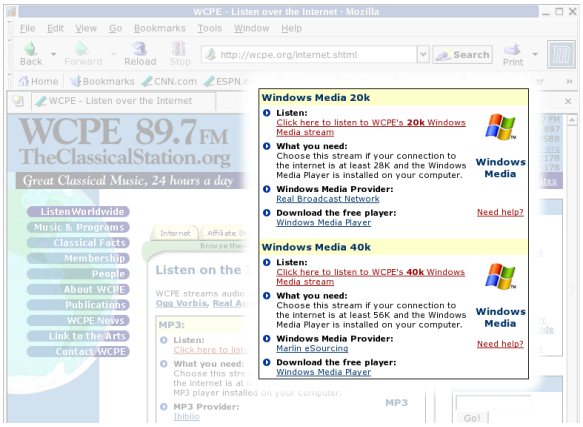

Figure 1.1: This figure shows the choice of streams available to listeners of the public radio station WCPE. Users choose the most appropriate stream when they start a new session and the bit-rate remains constant for the life of that session. Ideally, the transmission bit-rate for streamed data would adapt over time to match the available bandwidth for individual users. However, this adaptation is not easily achieved. As an alternative, content providers often provide a selection of streams, each of which is encoded at a different bit-rate.

The differences between linear and non-linear media pose new challenges to media streaming techniques. In particular, the need to deliver a custom data flow to each member of a large group of independent users can not be solved using traditional media streaming techniques which typically rely on the common data requirements inherent in linear media to provide efficient delivery.

Even the relatively simple adaptive application of controlling an audio stream's bit-rate to match a client's available bandwidth is not widely supported. Instead, streams are typically transmitted at a static, predefined rate to all users. If there is any choice at all in bit-rate, content providers will typically furnish a selection of static streams and require a human user to make the choice of which stream should be received, as shown in Figure 1.1.

For fully interactive non-linear media applications, the task of adapting data flows to match individual resource requirements and preferences is far more challenging then the audio bit-rate example. Adaptation for non-linear media requires both bit-rate control and content control: the ability to control both how fast data is arriving and what data is arriving.

There remain several fundamental problems that must be solved before a fully scalable and adaptive solution for streaming non-linear media can be developed. In the work presented in this dissertation, I begin to address some of these obstacles.

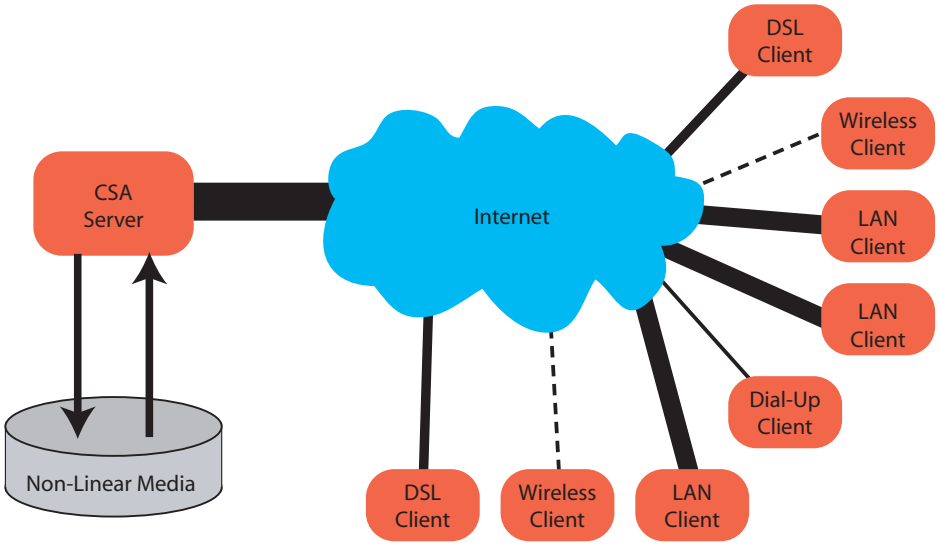

Figure 1.2: A non-linear media streaming system must be able to provide custom data flows to a large set of heterogeneously provisioned and independently operating clients. This dissertation presents a solution for this problem called Channel Set Adaptation, or CSA.

My work provides a complete framework that supports the streaming of non-linear media to a large group of independently operating and heterogeneously provisioned clients. Such a system is illustrated in Figure 1.2. As the results presented in this dissertation will show, the proposed framework scales well for large user groups and allows for each client to independently perform both content control (determining *what* data is received) and congestion control (determining the *rate* at which data is received).

This is achieved via three primary formalisms: (1) an abstract representation model for non-linear media called a *Representation Graph*, (2) a quantitative adaptation algorithm that frames adaptation as a maximization problem, and (3) the *Channel Set Adaptation* method that exploits scalable channel-based network models such as broadcast and multicast to provide highly scalable non-linear media streaming.

My research addresses a number of critical questions. How should large non-linear media data sets be organized to best facilitate scalable and adaptive streaming? How can servers support large independently operating user populations, particularly when data sizes are large and require significant amounts of bandwidth for transmission? How can clients adapt their data flows to changing application needs and resource allocations? How can this be done without negatively impacting the scalable properties of the server? Which communication models can support the competing needs for

scalable delivery and custom per-client data flows? How can applications efficiently express and evaluate adaptation policies?

Throughout this dissertation, I will present a unified framework to address all of these questions along with several other related issues. This framework is based upon a static and scalable channel-based transmission scheme which allows for dynamic and independent client adaptation through channel subscription operations.

1.3 Thesis Statement

The scalable streaming of non-linear media to a large group of independently adapting clients is enabled through Channel Set Adaptation: a framework that maps a partitioned media representation to a set of relatively thin multicast communication channels to provide scalable congestion and content control.

Scalable and adaptive non-linear media streaming can be achieved via three steps. The first step is mapping the non-linear data set to a simple generic data model capable of expressing semantically similar clusters of data.

Second, using the data model, an interactive utility-driven adaptation algorithm can be designed as an optimization problem, enabling an efficient, quantitative approach to multidimensional adaptation.

Finally, the data representation and adaptation algorithm support Channel Set Adaptation, a framework which can deliver custom data flows to individual users in a highly scalable fashion via standard multicast subscription operations. This approach moves all per-client operations away from the server, leaving only static transmission tasks which are independent of group size.

1.4 Major Contributions

This research has produced several research contributions. These include both practical software contributions, as well as more formal conceptual and algorithmic research contributions.

The research contributions include:

- **A Graph-based Model for Data Representation**: I have developed a graph-based abstraction that can be used to represent complex multidimensional and multi-resolutional non-linear media data sets. The model represents semantic

data relationships and syntactic dependencies through a structure called a Representation Graph. The abstraction can map to a very broad class of data representations and several examples are presented throughout this dissertation.

A representation graph abstraction uses nodes to represent individual elements of information and edges to represent the available means of resolving an individual node as well as any syntactic dependencies between nodes. Nodes are positioned within a multidimensional utility space to represent semantic relationships between elements of information. I represent access-level restrictions through the coloring of nodes into clusters.

I use this generic abstraction as the core representation in my work. This enables the incorporation of the remaining contributions of this thesis into any application which can map its dataset to the Representation Graph abstraction.

- **Utility-Driven Multidimensional Adaptation Algorithm**: Data adaptation is a critical system task in any non-linear media streaming system. The flow of data from source to receiver must be controlled to satisfy both the receiver's application-level requirements and the limited communication resources available to supply the receiver with the needed data.

  In this dissertation, I present a general framework for expressing multidimensional adaptation. The framework regards adaptation as a maximization problem in which the goal is to maximize the utility of the received data while simultaneously minimizing the access cost. The framework provides efficient algorithms for determining the appropriate behavior, as well as mechanisms for expressing application-specific data requirements.

- **Simple Server Design Philosophy**: I propose a driving philosophy for designing scalable distribution systems. This philosophy states that in order to achieve true scalable performance, all per-client tasks must be pushed away from any centralized resources and placed as close to the individual clients as possible.

  This design philosophy is embraced by the design of Channel Set Adaptation to provide truly scalable streaming of non-linear media. The server model employed by Channel Set Adaptation removes all per-client work from the central server resulting in a constant level of work independent of the size of the client pool. Thanks to the simple server design philosophy, this leads to a streaming architecture that is, theoretically, infinitely scalable.

- **Channel Set Adaptation**: I propose a channel-based communication framework for scalable and adaptive streaming of non-linear media. A combination of the multidimensional adaptation algorithm and the simple server design philosophy, Channel Set Adaptation uses a novel method of channel subscription management to exploit the scalable nature of multicast and broadcast networks for non-linear media distribution.

  Typically such scalable channel-based network models are used to distributed identical flows of information to large audiences. Channel Set Adaptation aggressively utilizes subscription operations to deliver unique data flows to each client in a highly scalable approach.

- **Performance Model**: I develop a performance model that expresses the impact on application performance of a number of important system parameters. The performance model is based on the results of a thorough evaluation of a non-linear media streaming application prototype built upon the concepts presented throughout the dissertation. The experiments, through network emulation, provide a realistic look at how underlying system properties can impact overall performance. The performance model highlights how current multicast technologies can be best improved in the future to support novel application-level technologies such as Channel Set Adaptation.

The practical contributions in my dissertation include:

- **A Library for Generic Adaptation**:

  I have implemented a C++ library, The Generic Adaptation Library (GAL), based on the representation abstraction and multidimensional adaptation algorithms presented in this dissertation. The library has direct support for multimedia and multi-dimensional adaptation.

- **Motivating Application Prototype**: As part of my evaluation, I have implemented a scalable non-linear media streaming infrastructure designed for an image-based rendering application that reconstructs entire spaces for virtual exploration. One possible application of this technology is in digital museums, where a centrally stored image-based model of a physical space can be streamed to a large number of virtual visitors.

1.5 Dissertation Organization

The remainder of this dissertation is organized as follows:

Chapter 2 reviews research most related to this dissertation and provides a context for my dissertation work.

Chapter 3 defines the representation graph generic data representation abstraction. It begins with an overview of common universal data properties. It then presents the formal structure of the representation graph and describes how the universal properties map to this structure. The chapter concludes with two use cases describing the application of the abstract model to concrete application scenarios.

Chapter 4 describes the multidimensional adaptation algorithm. After discussing the general problem of data adaptation, the chapter presents a framework for expressing adaptation as a maximization problem. The framework allows for the iterative evaluation of adaptation decisions via a quantitative formula. This makes adaptation more efficient and adaptation policies easier to express when compared to complex rule-based adaptation systems.

Chapter 5 details the Generic Adaptation Library, a library for multidimensional and multimedia adaptation. The library is a C++ implementation of the concepts presented in both Chapter 3 and Chapter 4. This chapter presents both a description of the overall design of the library and a high-level overview of the library's application program interface.

Chapter 6 begins with a description of the experimental prototype application used throughout the experimental evaluation. The prototype is an implementation of an image-based rendering streaming system designed for digital libraries. It then describes the experimental testbed and evaluation methodology. Finally, it covers a series of experiments evaluating the performance of the adaptation algorithm's performance under realistic application conditions. The results include a discussion of why the straightforward prototype design employed in this chapter falls short of the goal of scalable streaming.

Chapter 7 describes Channel Set Adaptation, a novel architecture for supporting truly scalable and adaptive streaming of non-linear media. This chapter presents the results from several experiments evaluating the improved performance of Channel Set Adaptation. A performance model is developed from both broadcast and multicast-based experiments that highlights which system parameters have the greatest impact on overall performance.

Chapter 8 concludes my dissertation with a summary of my research contributions and discussion of some areas for future work.

Chapter 2

Related Work

My dissertation research integrates concepts from a variety of research areas as it aims to support scalable and adaptive streaming of non-linear media. In this chapter, I present an overview of the most relevant research in each of these areas in the order in which they appear in this dissertation.

The chapter begins with an overview of related work in digital media representations and coding techniques. The vast array of application-specific data representations based on a common core of techniques motivates development of a generic data model in Chapter 3.

This is followed by a discussion of both ad hoc adaptation techniques as well as more generic attempts to model the adaptation problem. The adaptation solutions covered here serve as the foundation upon which the multidimensional adaptation algorithm of Chapter 4 is built.

Finally, I provide a brief sampling of related work in the area of media streaming, both linear and non-linear. This includes a review of both the underlying networking techniques as well as some application-level projects related to my thesis. These technologies describe the current tools available for media streaming and highlight the novel aspects of the Channel Set Adaptation technique presented in Chapter 7.

Taken together, these areas of related work provide a context for the remainder of my dissertation and frame my research contributions within the existing body of related work.

2.1 Digital Media Representations and Encoding Techniques

This section provides an overview of a wide range of encoding techniques and data representations employed by digital media applications. The individual encoding techniques form a toolbox from which complex data representations can built. This section first covers a series of encoding tools and representation techniques. It then covers a range of specific digital media representations.

The range of application-specific representations presented here are all built upon a common core of encoding tools. It is this common ground across data representations that both motivates and enables the creation of a generic data representation model in Chapter 3.

2.1.1 Representation Tools and Techniques

Across a range of media representations, from MPEG (ISO/IEC, 1993) to the Progressive Mesh (Hoppe, 1996), there exists a common set of tools that are combined in various ways to build efficient domain-specific data representations. These tools include transform coding, differential coding, and multi-resolution or hierarchical organization techniques.

Transform Coding

One of the most common techniques used for the encoding of digital media is transform coding. Transform coding is a mathematical operation that transforms a series of signal samples into a set of coefficients of the transform's basis functions. Because the goal is data compression, the basis functions are chosen such that the transformed coefficients can be stored more compactly than the original samples.

Several basis functions have been developed over the years for use with digital media. One of the earliest transforms is the Fast Fourier Transform (FFT) (Cooley and Tukey, 1965). The FFT is based on an efficient algorithmic implementation of the well known Fourier Transform. The FFT uses a set of sine functions as the basis functions.

More recently, alternative transforms have been widely employed including the Discrete Cosine Transform (DCT) (Rao and Yip, 1990) in a number of MPEG (ISO/IEC, 1993; ISO/IEC, 1995a) formats. Most recently, the Wavelet transform (Vetterli and

Kavacevic, 1995) has received a great deal of attention due to its inherently multi-resolutional properties.

While transform coding can be extremely effective at reducing the storage space required to represent a set of samples, it is typically applied only to relatively small region of digital media object. For example, the MPEG video formats apply a DCT to 8x8 blocks of pixels.

Regions encoded via a single transform are essentially atomic units of data. The block of coefficients must be transformed as a whole to recover the original samples.

Differential Coding

While transform coding allows data to be stored more compactly over a small neighborhood, differential coding aims to exploit similarities within the data that occur over larger distances. Differential coding techniques rely on coherence: the similarity between regions in a media dataset.

Perhaps the best example of differential coding is its use in video encoding. Because videos consist of a sequence of images taken closely in time, regions of successive frames f_t and f_{t+1} are often extremely similar. Because of the similarity, it is often more efficient to encode the difference $f_{t+1} - f_t$ rather than f_{t+1} itself.

This simplified example from video coding is indicative of the common technique of differential coding. This technique aims to create a more efficient data representation by encoding the difference between two data regions instead of the region itself.

Differential coding is often highly effective at increasing a representation's storage efficiency. However, the improvement comes at the expense of increased data dependence. These dependencies reduce the flexibility of data access.

Multi-resolution Or Hierarchical Organization Techniques

Media representations often employ multi-resolution or hierarchical data organizational techniques to provide more flexible data access to applications. Unlike monolithic data structures, these flexible techniques allow access to the same unit of data at various levels of detail.

Some of these techniques, such as wavelets, overlap with the methods presented earlier in this section. Wavelets (Vetterli and Kavacevic, 1995) are a transform coding technique which are also designed to automatically create multi-resolution representations. For example, they form the basis for the JPEG2000 (ISO/IEC, 2000) image

compression standard for multi-resolution image compression.

Other multi-resolution or hierarchical techniques include spatial subdivision methods (such as quad-trees and oct-trees (Samet, 1984)) and multiple description techniques which can encode data independently for different resolutions (such as levels-of-detail (Lindstrom et al., 1996) and multiple descriptor video (Chakareski et al., 2003) or audio as in Figure 1.1). Layered encodings are another multi-resolution technique where a dataset is divided into discrete quality layers. This can be used, for example, to provide access to video at a variety of fidelity levels (Rejaie et al., 2000).

Hierarchical or multi-resolution representations provide adaptive data access by allowing applications to resolve the encoded data at a variety of fidelity levels. That flexibility can be arranged along a single dimension or along multiple dimensions. A representation that provides multi-dimensional adaptive access allows an application a greater degree of flexibility in its data access patterns.

2.1.2 Application-Specific Media Representations

Specific media representations are designed with two primary goals in mind: efficiency and accessibility. Because digital media datasets are typically large in size, an efficient representation is often a critical design goal. At the same time, the representation must provide effective access to the encoded data that reflects the supported applications' expected behavior and requirements.

The two goals of efficiency and accessibility are often in conflict and, depending on a specific application's needs, a number of design decisions must be made to manage this trade-off (Gotz et al., 2002). As a result, a vast array of application-specific data representations have been developed that are tailored toward only a small subset of digital media applications. These representations often combine several of the representation tools in unique ways to exploit the application-specific properties of both the data set and the application.

In this section, I review a number of standard representations for digital media. This review begins with coverage of the highly standardized representations used for images, audio, and video. It then presents several application-specific data representations that have been developed for specific domains within computer graphics, and Image-Based Rendering (IBR).

Image Representations

Digital image representations have been standardized for many years and several competing formats have been proposed. These formats cover a broad range of complexity levels, from relatively simple encodings, such as PPM and BMP, to more complex standards such as JPEG (Pennebaker and Mitchell, 1993) and JPEG2000 (ISO/IEC, 2000).

These image representation vary widely in their design aims and applications use whichever format is deemed most appropriate. For example, applications that have no spare cycles to spend on decoding will opt for a less complex data representation like PPM which can be immediately used without any decoding operations. Conversely, an application that requires highly compressed image storage might opt for the more space efficient JPEG format.

Some image formats provide more than just a standard data organization specification. Lossy image formats, like JPEG, often provide a quality parameter which can be used by an application to determine how closely an encoded image should match the original. The quality parameter is used to configure the DCT-based transform coding algorithm. Additional encoding accuracy comes at the expense of additional storage space.

The quality parameter provides an important dimension of adaptive behavior at the point of encoding. However, decoding applications remain limited to access the image at the quality level specified at the time of encoding.

Other formats, such as the newer JPEG2000 standard, provide flexibility for both encoding and decoding applications. In particular, JPEG2000 uses a Wavelet transform which is inherently multi-resolutional. The transform creates a progressive ordering of the encoded image data. A decoding application can use any number of bytes, starting from the front of the image data stream, to decode the image. As more bytes are incorporated into the decoding process, the image quality improves. This provides a dimension of adaptivity to both the encoding and decoding processes.

Audio Representations

Audio encoding formats have also received a lot of attention from the standardization community. Early formats, such as the WAV standard (IBM Corporation and Microsoft Corporation, 1991), enabled high-fidelity digital audio representations but required significant data storage.

More recently, the MP3 standard (ISO/IEC, 1995b) has enabled a revolution in digital audio by reducing the storage requirement for audio files by an order of magnitude. This is done via carefully tuned transform coding and quantization techniques tailored to optimizing an encoding to match the abilities of the human auditory system.

With the advent of the Internet, additional audio formats have been developed aimed at delivery high quality audio over variable bit rate communication channels via streaming technologies. These efforts include proprietary formats from RealNetwork's Real Audio, Microsoft's Windows Media effort, as well as other techniques such as the open-source Ogg Vorbis and applications that stream MP3 data. I will cover the streaming of audio and other media from a networking perspective in Section 2.3.

Video Representations

As with audio formats, a huge effort has been made at standardizing video representations. However, even with video a wide range of formats have been developed, each of which have properties that make it a good choice for specific applications.

For example, low bit-rate video, as often used for video conferencing applications, can be encoded with the H.261 format (Union, 1993). Alternatively, higher bit-rate video can be encoded by one of the many generations of MPEG video formats (ISO/IEC, 1993; ISO/IEC, 1995a).

As with the image and audio formats presented earlier, these formats employ combinations of the basic encoding techniques to create application-specific data representations. They employ both transform coding and differential coding between sequential frames to create highly dependent representations that are extremely space efficient. This enables relatively small files sizes for the otherwise massive amounts of data required for video applications.

Multi-resolutional techniques have also been used for video encoding. In particular, layered video representations (Taubman and Zakhor, 1994) have been designed to produce video formats that allow access at several discrete quality levels.

Computer Graphics Representations

Standardized data representations have made significantly less inroads in the computer graphics application space. This is due in part to the wide range of application areas that require specific data representation properties.

Several data representations have been designed to accommodate flexibility in model

accuracy. The rendering of extremely complex geometric objects requires simplification techniques (Luebke, 2001) that generate multi-resolution models that can be selectively rendered by the application. For example, a simplification algorithm can generate discrete sets of multi-resolution geometric models (Lindstrom et al., 1996) that can be selected at render time to best allocate limited resources.

Alternatively, the progressive mesh algorithm (Hoppe, 1996) generates a progressive ordering of the geometric data similar in nature to the progressive code stream generated by the JPEG2000 image compression standard. An application utilizing this representation can choose how many bytes of the geometric model are needed at any given time. As more bytes of the data set are incorporated into the application, a more accurate geometric model is obtained.

IBR Representations

The field of IBR, a specialization of computer graphics, attempts to reconstruct virtual scenes using input sets of photographic samples rather than the traditional geometry-based models used most often in computer graphics applications. Even within this highly specific area, several competing data representations have been proposed, including the Light Field (Levoy and Hanrahan, 1996) and the Lumigraph (Gortler et al., 1996). Both of these systems develop a data representation for a set of images as a collection of two-dimensional slices of a four-dimensional portion of the plenoptic function. Surface light fields (Miller et al., 1998) provide yet another parameterization of the image samples and another algorithm for reconstructing the plenoptic function.

The Sea of Images (Aliaga et al., 2002) algorithm provides an alternative parameterization of the plenoptic function which allows user navigation along a horizontal plane through space, rather than the vertical plan typical of earlier IBR alogrithms. The Sea of Images algorithm allows greater user navigation than previous IBR systems because of the change in parameterization. The IRW representation (Gotz et al., 2002) also addresses the data representation issues associated with the Sea of Images approach to IBR based on epipolar geometry.

2.1.3 Data Representation Recap

Throughout this section, I have presented a long list of application-specific data representations designed for various media types. Each of these representations is unique in its details, but built from the same common set of tools: transform coding, differential

coding, and multi-resolution data organization structures.

The commonality in basic building blocks for each of these data representations suggests that a generic data representation can be designed that captures the underlying structural data relationships: data dependencies, adaptive structures, and multi-dimensional properties. The design of such a generic data representation is the topic Chapter 3 in this dissertation.

2.2 Adaptation

Data flow adaptation is an essential component of a vast array of networked media applications. The flow of information from source to receiver must often be controlled to effectively utilize a constrained communication infrastructure while continuing to meet application requirements in a timely manner.

Broadly speaking, there have been two approaches to the problem of adaptation. Previous work has largely addressed the problem of adaptation in the context of specific applications. These applications have employed ad hoc policies based on either simple heuristics or more complicated rule systems. Alternatively, a few recent efforts have attempted to identify the common elements of adaptation and develop a more general approach. In this section, I review several research projects in both of these areas: ad hoc policies and generalized adaptation.

2.2.1 Ad Hoc Policies

Ad hoc adaptation strategies have been employed in a number of specific application areas. These areas include multimedia systems, computer graphics, and visualization. These techniques attempt to develop systems of rules or heuristics that balance the competing needs of both application and system level requirements.

Multimedia

In multimedia, adaptation is an essential task. For example, video streaming adaptation (i.e. rate control) has been achieved through several different ad hoc mechanisms. These include controlling the signal-to-noise ratio in video coding (Kanakia et al., 1993), frame rate adaptation (Rowe and Smith, 1992), or combinations of the two methods (Ramanujan et al., 1997).

The layered video coding techniques discussed earlier in this chapter have also been used to achieve adaptive rate control (Rejaie et al., 1999). In this work, complex rules have been developed that govern when additional layers should be added or removed to maintain adequate performance..

More general approaches to multimedia adaptation include quality of service semantics (Walpole et al., 1999) and augmentation and substitution models (Boll et al., 1999). Both of these research efforts have attempted to move toward a more generic quantitative approach to expressing multimedia adaptation. They reduce the need to develop complex sets of rules, but fall short of the generic models covered in Section 2.2.2.

Of the work covered so far in this section, the effort by Walpole et al. to develop a quality of service model based on a spatial metric is most closely related to the generic framework for multi-dimensional adaptation I present in Chapter 4. In their model, Walpole et al. define a presentation space. This construct represents the space of all possible presentations of a particular media object. Within that space, they define a single static point as corresponding to the perfect quality presentation. They then define adaptation as an attempt to choose a presentation located most closely to the perfect quality location within the presentation space.

Computer Graphics

In computer graphics, the complexity of geometric models is often much greater than typical systems can handle for real-time rendering. As a result, significant effort has been spent developing techniques for geometric simplification (Luebke, 2001). At runtime, models of appropriate resolution are chosen based on any of a number of heuristics, including visibility estimates (Cohen-Or et al., 2001), screen-space error computations (Lindstrom et al., 1996), and rendering costs (Funkhouser and Sequin, 1993). These techniques are especially important for remote access to large, complex datasets where communication bandwidth is at a premium (Gotz and Mayer-Patel, 2005a; Rusinkiewicz and Levoy, 2001; Teler and Lischinski, 2001).

Visualization

Visualization systems must also deal with databases that contain far too much information to graphically display. As a result, adaptive mechanisms have been widely adopted. These include multi-resolution terrain models (Floriani and Magillo, 2002),

multi-resolution volumetric representations (Lamar et al., 1999), and multi-resolutional and multidimensional data query infrastructures (Beynon et al., 2000). The decision regarding allocation of constrained resources must carefully reflect application-specific requirements. For this reason, adaptation is most often performed via ad hoc solutions based on domain-specific rules and heuristics. This has led to a proliferation of ad hoc adaptation solutions.

2.2.2 Generalized Adaptation

While most adaptation techniques have been developed in the context of a specific application, there are some notable exceptions. Some researchers have proposed several design principles for adaptive systems (McIlhagga et al., 1998). These principles highlight several desirable properties of adaptive applications, including the need to identify the degrees of freedom for adaptive degradation of system performance. The adaptive framework presented in Chapter 4 is in line with many of these principles.

Other investigators emphasize the importance of data representation in adaptation (Policroniades et al., 2003). In their work, Policroniades et al. propose a data representation that considers files not as monolithic objects, but rather as collections of elements. These elements can then be chosen based on application-specific policies as part of content adaptation. The high-level design for this work is similar to portions of my proposed Representation Graph data structure covered in Chapter 3. However, their model is defined using simple composite relationships between elements, while the Representation Graph provides much more powerful structures that can express far more complicated data relationships which are essential for effective adaptation.

Formal adaptation models such as Adaptation Spaces (Bowers et al., 2000) have been developed to explicitly specify possible alternative application behaviors for reliable systems. This approach works best for systems with a small presentation state space where enumeration of all possible states is not onerous. The state space centric approach to adaptation is similar to the less general approach taken by Walpole et al. for multimedia presentation adaptation (Walpole et al., 1999).

In other work, resource-centric quality-of-service models (Chatterjee et al., 1997) are used to adapt resource allocations to compensate for changes in the system's state. Chatterjee et al. propose a rather complex set of rules for determining resource allocation at the time an application is invoked. However, it is their method for adapting resource allocation while the application is executing that is most closely related to the

adaptation algorithm of Chapter 4. They propose a multidimensional spatial benefit function used to guide ongoing resource adaptation. The benefit function is somewhat similar to the spatial utility metric proposed in my research. However, their function is static and predefined and is only used to compare the relative utility of alternative states generated by a resource manager. The utility metric proposed in my adaptation framework is highly dynamic, and based on the current application and system conditions.

2.3 Media Communication Techniques

.

This section reviews a selection of previous work most related to the Channel Set Adaptation approach to non-linear media streaming covered in Chapter 7. The review covers both linear and non-linear techniques.

Most of the existing research in the area of media communication technology deals with linear media streaming for audio and video. This section begins with a very brief overview of some of this work. It then covers the subset of linear media research that addresses efficient streaming techniques for large user groups.

This section then reviews several efforts at addressing the unique requirements of non-linear media streaming. These techniques have largely concentrated on single user systems due to the individualized data flows required by the nature of non-linear media. Finally, this section describes two efforts in scalable database access that explore issues similar to those faced in our work.

2.3.1 Linear Streaming Techniques

Streaming technologies for linear media objects have received a large amount of attention in recent years as media streaming has matured into a fixture on today's Internet. Several commercial technologies, including Real Network's RealAudio and Microsoft's Windows Media are now readily available and used to stream both audio and video content.

These technologies are based on several fundamental research efforts, including application-level framing, (Clark and Tennenhouse, 1990), forward-error correction (Rizzo, 1997), and early research initiatives in developing successful streaming protocols (Perkins et al., 1998). This has led to several protocol standards for supporting

real-time streaming, including RTP (Schulzrinne et al., 1996) and RTSP (Schulzrinne et al., 1998).

2.3.2 Linear Streaming to Large User Groups

The high bandwidth requirements for streaming audio and video have motivated several efforts to more efficiently support the distribution of linear media data to large groups of users. The multicast network model (Deering and Cheriton, 1990), where data streams are efficiently distributed to groups of interested users, was developed as an efficient alternative to unicast.

Multicast allows a user to join a group of receivers, all of whom receive an identical flow of data. A multicast server is then able to transmit a single stream to the entire group, rather than send individual streams to each user. The single stream is replicated as needed within the network and delivered to the interested participants.

Problems with deployment of IP Multicast, the standardized version of infrastructure multicast, have led to significant effort in developing Application Layer Multicast (ALM) (Banerjee et al., 2002; Begnoche et al., 2005; Castro et al., 2002; Chu et al., 2000; Chu et al., 2001). Rather than relying upon core network resources to perform group management and data replication, ALM performs these tasks at the application level using the very hosts that are participating in the multicast session.

Both IP Multicast and ALM techniques deliver identical flows to all receivers, making them ideal solutions for scalable linear media delivery. However, even for linear data, more flexibility is often required. Several researchers have explored novel uses of multicast protocols to provide limited flexibility in the time of access to linear media. For example, scalable video-on-demand can be accomplished through pyramid broadcasting (Viswanathan and Imielinski, 1996) and its many derivatives (Acharya et al., 1995; Hua and Sheu, 1997; Juhn and Tseng, 1997).

Similarly, other work has explored using layered media delivery via multicast to improve flexibility in the rate of data delivery. This includes the work on Receiver-Driven Layered Multicast (McCanne et al., 1996) which uses a layered media representation and multicast to support a heterogeneous set of linear media receivers. Subsequent work, such as the Thin Streams architecture (Wu et al., 1997), has improved on the performance of the original algorithm.

Despite this large body of previous work, scalable solutions have been largely limited to linear media objects. Many of these techniques depend upon the predictable access

patterns associated with linear media applications. In our work, we explore techniques that exploit multicast delivery for scalable and adaptive *non-linear* media streaming, where data access patterns are not known *a priori*.

2.3.3 Non-Linear Media Streaming

Several researchers have explored techniques for *single-user* streaming of non-linear datasets, particularly in the area of computer graphics. For example, streaming for complex 3D geometric models can be accomplished by selectively transmitting multi-resolution models of geometric objects based on the user's navigation of the scene (Teler and Lischinski, 2001). This work introduces an ad hoc benefit function that evaluates the relative utility of various models to drive the selective transmission.

Progressive mesh representations, which prioritize geometric information based on their importance to overall shape, have been used to develop geometric data streams that are resilient to lost packets during transmission (Al-Regib et al., 2002). The data encoding includes redundant copies of the low resolution geometric information to speed loss recovery.

Other researchers have explored single-user streaming for alternative computer graphics techniques. For example, selective transmission techniques have been applied to image-based rendering with concentric mosaics (Zhang and Li, 2001). This work has properties similar to standard remote file system architectures. Similar work has addressed the streaming techniques for point-based models (Rusinkiewicz and Levoy, 2001) by allowing clients to make individual requests for specific portions of the model.

These techniques, while supporting streaming access to non-linear media, are all based on individual user requests where the streaming server performs per-client work. As a result, the server workload and outgoing bandwidth requirements typically limit these solutions to very small user populations.

Recognizing the need for more scalable solutions, some researchers have explored support for broadcasting geometric data (Bischoff and Kobbelt, 2002) for scalable access. However, this work is limited to broadcast environments and does not allow any per-client control over the received data flow. All users receive the exact same flow of information, making it most applicable to small datasets where last-mile bandwidth efficiency is not a concern. Unfortunately, the Internet is not a broadcast medium and the last-mile links are often the primary communication bottleneck link for individual clients.

2.3.4 Scalable Database Access

The database community has explored scalable access frameworks that attempt to support large numbers of simultaneous queries. Certain aspects of the scalable non-linear media streaming problem have similarities to the problems faced by the database community. In particular, two of these efforts use solutions that draw on concepts that are closely related to our work.

The Datacycle Architecture (Herman et al., 1987) has been proposed for very high throughput database systems. In this architecture, the entire database is broadcast repeatedly over a local high-bandwidth communication network. Data filters attached to the network then work in parallel to search the stream of data and satisfy complex queries.

In more recent work, Broadcast Disks (Acharya et al., 1995) were developed for asymmetric communication environments where bandwidth is abundant for downstream transmission but expensive for upstream queries. Data is repeatedly broadcast over a single broadcast channel, and rates for repeating the broadcast of individual data elements are chosen to control their expected access times.

2.4 Summary

This chapter presented an overview of several areas of research related the research presented throughout the remainder of this dissertation. The related areas include digital media representations, adaptation techniques, and media streaming technologies.

The review began with an overview of related work in digital media representations and coding techniques. A large number of standard media representations were presented as well as a set of common techniques that form a tool box from which standard formats are built. Regardless of the specific encoding details, each representation creates similar data relationships and dependencies. This similarity is exploited in Chapter 3 to develop a generic data representation model.

The next area of related work covered in this chapter is adaptation. The review addressed both ad hoc adaptation techniques as well as research efforts aiming to develop a general model for the adaptation problem. The adaptation solutions covered here will help define the concepts behind the generic multidimensional adaptation algorithm presented in Chapter 4.

Finally, this chapter provided a very brief review of the extensive body of related

work in the area of media streaming. The review addressed work on both linear and non-linear media types. First, a survey of several underlying network technologies was presented. This was followed by an overview of tools that have been developed to support application-level streaming systems for both linear and non-linear media. In both cases, both single-user and scalable approaches were addressed.

The vast range of research topics covered in this chapter form the foundation for the research contributions presented throughout the remainder of this dissertation. My efforts in developing a general data representation, multidimensional adaptation framework, and scalable delivery architecture all build upon the many innovations and contributions of the research projects presented in this chapter.

Chapter 3

A Graph-Based Model for Data Representation

This chapter describes the *Representation Graph* (Gotz and Mayer-Patel, 2004), a graph-based data model that forms the underlying data structure in the adaptation algorithm and communication framework presented in later chapters. Using a single generic data representation provides a common vocabulary and allows for application-independent solutions to both the adaptation and communication problems. Solutions for specific applications can then be developed by providing a mapping between a specific data representation and the abstract Representation Graph model.

I begin by presenting a number of universal data properties that are common across a large class of adaptive multimedia applications. The discussion is focused on the properties most critical to the adaptation and communication tasks for which the data model is created. I then define the specific structures in a Representation Graph (RG). A RG is a graph-based structure composed of nodes and edges embedded within a multidimensional space. Throughout the definition, I illustrate how each component maps to the aforementioned universal data properties.

Finally, I present two examples in which I map application-specific data structures to the abstract model. In each of these examples, I take a well established application and provide a step-by-step mapping between the application-specific data representation and my graph-based abstract data model.

3.1 Universal Data Representation Concepts

There are a wide variety of adaptive media applications where flexible access to data is essential to system performance. As a result, a large number of application-specific data representations have been designed. These include, for example, progressive image coding for still pictures in JPEG2000 (ISO/IEC, 2000), layered video coding for motion pictures in MPEG2 (ISO/IEC, 1995a), and the progressive mesh representation computer graphics (Hoppe, 1996).

In this section, I review a number of universal properties common to most representations designed for adaptive multimedia applications. These properties include the representation dimensionality, elements of information, data clusters, and the representation index.

3.1.1 Dimensionality

Adaptive multimedia data representations, regardless of application, are typically hierarchical or multi-resolutional in nature. This derives from the fundamental reason for adaptation: limited resources must be allocated in response to changing system conditions. Without multi-resolutional representations there would be little flexibility in data access and adaptation would be difficult.

A data representation can be characterized by the number of dimensions along which adaptive data access is possible. I refer to this property as the representation's *dimensionality*.

A relatively simple representation may adapt across only a single dimension. This is the case, for example, in a layered video representation. In layered video, a single video stream is encoded as several incremental layers. The base layer contains a low resolution video encoding and each additional layer can be incorporated to improve the video's image quality. An application can adapt to changes in its available resources by changing the number of active layers. Because the number of layers is the only controllable parameter, this representation would have a dimensionality of one.

An alternative representation could contain several adaptive dimensions. For example, a video representation could provide direct control over three quality parameters: color depth, frame rate, and image resolution. Such a representation would have a dimensionality of three.

3.1.2 Elements of Information

A dataset can be considered to be a collection of individual *elements* of information. An element is an atomic unit of information at the application level. For example, in a layered video representation, one layer of one frame might be treated as an element.

Elements do not correspond to actual bytes of data, but only represent a conceptual unit of information. This is an important distinction because a single element might be encoded in several different ways. For example, a video stream, where each frame is considered an element, can be encoded using a variety of standard video formats. Each encoding will produce a unique data stream that represents the same sequential set of video frames. The elemental structure of the video stream, i.e. the set of frames, is the same regardless of the encoding.

Elements represent individual units of information, but they are not necessarily independent from other elements within the same data representation. For example, there are often encoding dependencies introduced between elements. Dependencies can be introduced for several reasons, such as for specific data structural reasons (for example, the dependence between layers in a layered video stream) or for storage efficiency (for example, predictive coding between video frames). Data dependencies between elements can be viewed as the *syntactic* relationship between elements.

There can also be *semantic* relationships between elements. For example, a multiple-descriptor video stream stored at several bit-rates has a group of independent elements for each frame. In this example, unlike a layered representation, each element is independently encoded and there are no syntactic dependencies between different bit-rate streams. However, there is still a logical relationship that relates the set of elements within the bit-rate dimension. This relationship is semantic.

3.1.3 Data Clusters

When information for each element is coded as bytes of data, it is grouped into *clusters*. I define a cluster as an access-level data structure which combines the encoded data of one or more elements. Clusters are atomic units of data that must be accessed as a single unit. For example, each quality layer in a video representation could be considered a cluster. During video playback, data is accessed by choosing a certain number of layers. This is in contrast to the elemental notion of individual video frames.

Given this view of a data representation, the RG models a dataset as a collection of elements. Elements may have either semantic or syntactic relationships with other

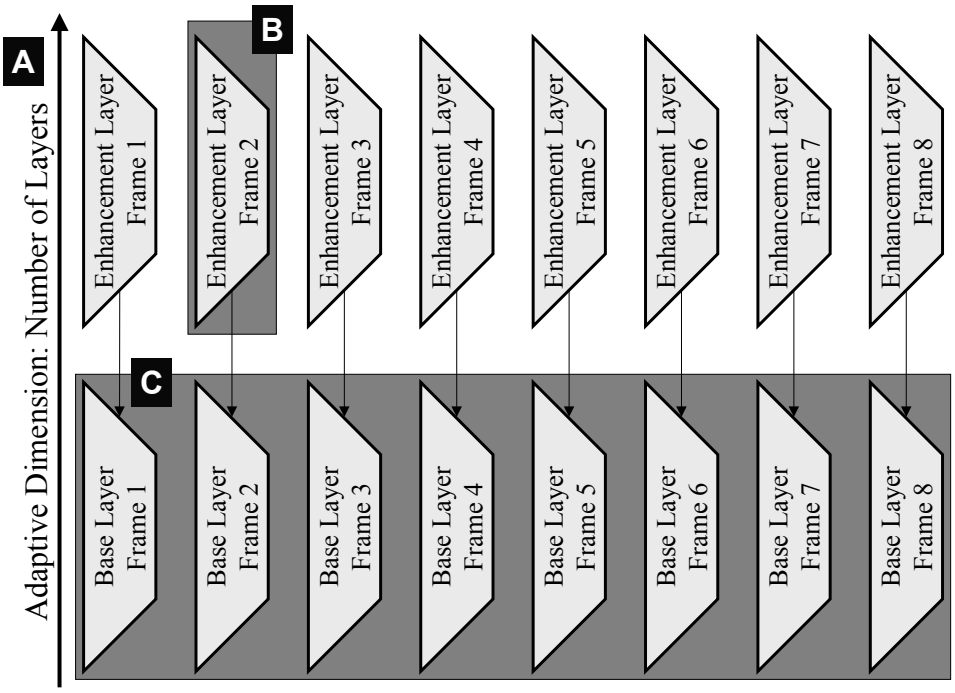

Figure 3.1: A simple layered video representation has a dimensionality of one. (A) The single adaptive dimension corresponds to the number of layers used during decoding. (B) Within each layer, there is a sequence of individual frames. These are the elements in this example. (C) Groups of frames form layers, which, as groups of elements that must be accessed together (or not at all), are considered clusters.

elements. Elements are encoded and stored, possibly with data from other elements, as data clusters. These relationships are illustrated in Figure 3.1.

3.1.4 Representation Index

Finally, it is important to note the distinction between the actual encoded data and the representation's data structure. The representation structure, defined by elements and their relationships, is used to decide which data is required during adaptation. I refer to this as the *representation index*. The index may be explicit or implicit, but it is always there. When explicitly defined, the structure has been given many names in different fields. For example, the structure has been called a scene-graph *skeleton* (Varadhan and Manocha, 2002) in graphics, a *description* (Gotz and Mayer-Patel, 2005a) in streaming complex media types, and a *sketch* (Policroniades et al., 2003) in wireless systems.

In the layered video example of this section, the representation index would include the specification of how each video frame relates the neighboring frames. This is in

contrast to the actual image data stored with each frame.

3.2 A Generic Representation Abstraction

In this section, I define the RG and its many components. The representation graph abstraction is a graph-based structure embedded within a multidimensional utility space. In the following subsections, I describe the abstraction and outline the mapping between individual components and the corresponding universal data representation properties outlined in the previous section.

3.2.1 Illustrative Example

Throughout this section, I illustrate each component of the RG via a simplified sample application. For each new concept, the formal definition is followed by a concrete example. The examples are taken from a simplified computer graphics application.

The example computer graphics system consists of a virtual user navigating along one dimension of a virtual scene. The scene is composed of a collection of geometric objects, each of which is stored at multiple resolutions. For example, imagine a user navigating through a virtual forest, as shown in Figure 3.2. Each object is represented by a set of multi-resolution models. These models are layered in the sense that lower resolution models are used as a basis for encoding higher resolution models.

At runtime, the application adapts the flow of incoming data to reflect both limited rendering resources and a moving viewpoint within the one dimensional scene. The goal is to render a forest as accurately as possible while working within the limited resources of the system. As a result, the system renders high resolution models for trees located close to the user's position and low resolution models for far away trees. The dynamic adaptive nature of this application can be seen by comparing Figure 3.2(A) and Figure 3.2(B).

In the remainder of this section, an representation graph for this example application is developed one component at a time. The example is intentionally simplistic and serves only to clarify the individual concepts in our framework. More detailed examples from common applications are illustrated in Section 3.3.

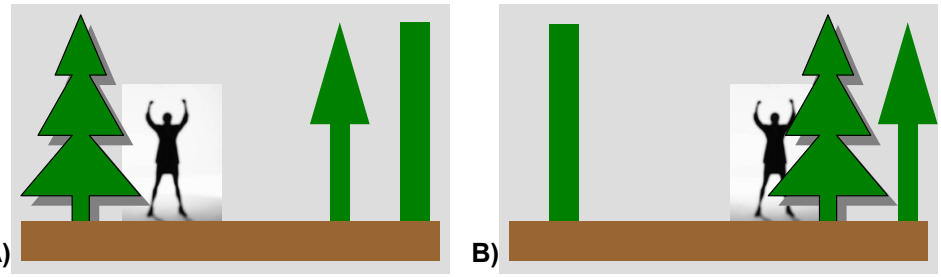

Figure 3.2: The illustrative example used throughout the generic representation definition is a simplified computer graphics application. In this application, a user moves a virtual person through a one-dimensional scene populated with a number of trees. Each tree is represented at multiple resolutions. (A) Trees close to the viewpoint are rendered with high resolution models while far away trees are rendered at low resolution. (B) As the position of the viewpoint changes, trees are rendered at different resolutions.

3.2.2 Utility Space

The first component of the RG is the *utility space*. The representation abstraction is embedded within a multidimensional utility space, noted as $\mathbf{U}$. The utility space is a linear space over the field $\Re$. The space $\mathbf{U}$ is an n-dimensional space where n corresponds to the dimensionality of the dataset to be represented.

For example, our sample computer graphics application has two dimensions: one to represent the navigable scene, X, and a second to represent an object's resolution layer, L. This defines a two dimensional utility space $\mathbf{U} = (X \times L)$, as shown in Figure 3.3(A).

Taxonomy of Dimension Types

In general, utility space dimensions can be categorized into a taxonomy of three types. For any instance of $\mathbf{U}$, there may be zero or more of each dimensional type. The three classifications are:

- **Navigable**: Dimensions in which an application maintains a dynamic point of interest. The application adjusts the position of the point of interest based on application conditions.

- **Static**: Dimensions in which there is a predefined and constant point of interest regardless of any dynamic system conditions. The coordinate value of this point

is called the *static value* for the dimension. A static dimension is analogous to a navigable dimension with a fixed point of interest.

- **Relational**: Dimensions which relate two or more distinct media subspaces (defined below) to form the global utility space $\mathbf{U}$. Relational dimensions provide a mechanism to express the relative importance between two or more independent media objects.

Of particular importance is the set of navigable dimensions. Within $\mathbf{U}$, the *navigable subspace*, $\mathbf{N}$, is defined as the region of space defined by the set of all navigable dimensions. It is within this subspace that the application can adaptively update the points of interest for each dimension. This will be a critical in feature in the adaptation algorithms presented in Chapter 4.

Consider, for example, the two-dimensional sample application. There is a single navigable dimension, X, which defines the spatial position of geometric objects within the scene. The second dimension, L, is an static dimension and is used to reflect the resolution level of each representation of an object. The static value is zero, representing the lowest possible level, because the system is interested in resolving low resolution data before high resolution information.

Furthermore, there is a one-dimensional navigable subspace of $\mathbf{U}$ defined as $\mathbf{N} = (X)$. There would be no relational dimensions in this example because there is only one media type within the application. The meaning of relational dimensions will become more clear during the discussion of media subspaces in the next subsection.

Media Subspaces

The overall utility space can be broken down into individual *media subspaces*, one subspace for each independent set of media objects. For example, a streaming application with separate audio and video tracks might contain two independent media objects. This would correspond to two media subspaces.

A single media subspace is noted as $\mathbf{M}_i$. For an application with n independent sets of media objects, a set of media subspaces is defined as $\mathbf{M} = \{\mathbf{M}_1, \cdots, \mathbf{M}_n\}$. By definition, each $\mathbf{M}_i$ is independent and shares no dimensions with any of the other media subspaces.

In the event where dimensions are logically equivalent across subspaces (i.e. time in the audio/video streaming application), each media subspace requires its own instanti-

ation of the dimension. For example, there could be a time dimension defined in both the audio and video subspaces of the streaming application.

The multiple time dimensions, one for each media subspace, are critical to adaptation. While the detailed adaptation algorithm is presented in Chapter 4, an intuitive explanation is provided here.

For example, a streaming application with both audio and video tracks would need to determine how to allocate the available bandwidth across to retrieve both audio and video data. For a particular application, it may be desirable to ensure that 10 seconds of future audio data has been buffered locally while only pre-fetching 5 seconds of video data. To allow this decision, the time dimension of the audio track must be treated differently than the time dimension of the video track, even though both dimensions reference the same notion of time.

The independence of each media subspace reflects the fact that each subspace corresponds to an independent set of media objects, even if they share logically common dimensions, such as time. Therefore, the intersection of any two media subspaces is always empty. This leads to the following equation for all $i, k \in \{1..n\}, i \neq j$:

$$\mathbf{M}_i \cap \mathbf{M}_j = \emptyset \tag{3.1}$$

However, an adaptive application must still choose how best to allocate resources across independent media objects. As a result, it will be essential to make utility comparisons across multiple media subspaces. These comparisons are made possible through the use of relational dimensions which are used to join together two or more independent media spaces into a common global utility space.

In this chapter, I simply introduce relational dimensions conceptually as added dimensions used to compare two or more independent media subspaces. A more detailed discussion regarding relational dimensions and their impact on the measurement of utility is presented in Section 4.6.

3.2.3 Nodes

The primary structural component within the RG model is a *node*. A RG is composed of a set of nodes, noted as S. Individual nodes correspond to the notion of elements and represent atomic units of information as discussed in Section 3.1.2. Each node $n_i \in S$ has a number of individual properties.

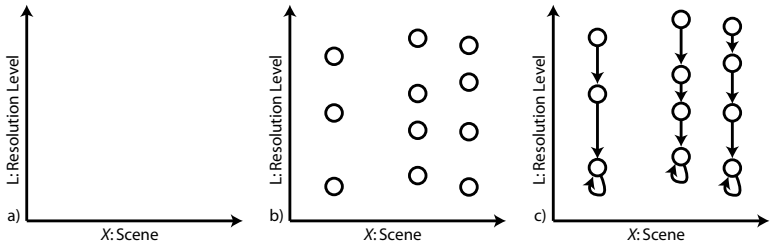

Figure 3.3: The Representation Graph (RG) for the illustrative application. (A) The RG is embedded within a two dimensional utility space $\mathbf{U} = (X \times L)$, where X is the spatial dimension and L is the resolution level dimension. (B) Individual models of geometric objects are represented as nodes whose position within $\mathbf{U}$ is determined by their position and resolution. (C) Edges are used to represent the coding dependencies between different resolution models of the same object.

Node Properties

Each node n_i is associated with a single media object and is located at a specific point $\mathbf{Pos}\{n_i\}$ within a single media subspace $\mathbf{M}_j$. A node has no defined position within all other media subspaces $\{\mathbf{M}_k\}, k \neq j$, and $\mathbf{Pos}\{n_i\}$ is therefore more correctly described as a hyperplane within $\mathbf{U}$ rather than a point. The relative positions between nodes are used to express the semantic relationship between individual elements of information.

For example, consider Figure 3.4, which shows three nodes positioned within a single resolution dimension. Each node corresponds to one of three tree models, each of a different fidelity. The relative positions of the nodes express their semantic relationship in terms of resolution.

In addition to position, each node is assigned one of four *states*. The state for a particular node, $\mathbf{State}\{n_i\}$, changes over time through *state transitions*. Node state is used to keep track of the flow of information during adaptation. For this reason, I delay covering states in detail until Section 4.9 in the chapter on adaptation. However, it is important to note here that this state is a node-specific property.

A node also maintains lists of both *arriving edges*, $\mathbf{Arr}\{n_i\}$, and *departing edges*, $\mathbf{Dep}\{n_i\}$. The number of edges in a particular list can be expressed as $|\mathbf{Dep}\{n_i\}|$. Edges, which connect two or more nodes, and restrictions on the number of edges in each list for a node are discussed in detail in the next section (Section 3.2.4).

Finally, specific applications may choose to define auxiliary meta-data for each node, $\mathbf{Meta}\{n_i\}$. Meta-data can be used to represent more specialized application information that is not expressible through the existing structures of the generic RG

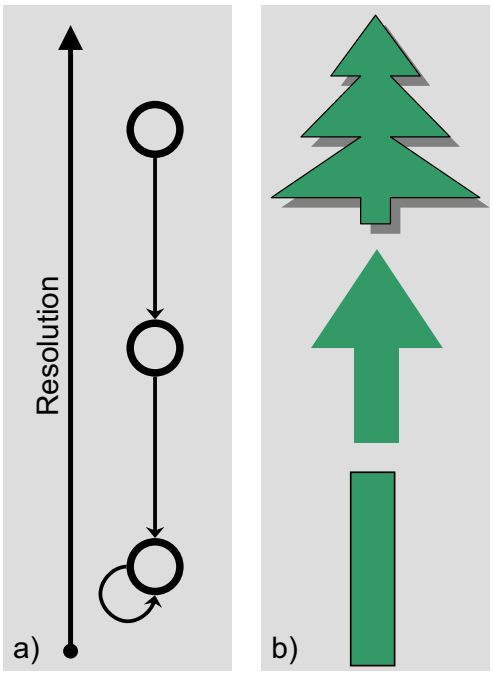

Figure 3.4: Nodes are used to represent individual elements of information. In (A), three nodes are shown arranged within a one-dimensional utility space. There are three nodes which correspond to the three tree models shown in (B). There is a one-to-one relationship between nodes and models.

model.

Nodes in the Illustrative Example

In the running example, nodes can be used to represent the model of an individual tree at a specific resolution. Due to the multi-resolutional dataset, each tree is represented by multiple nodes. Each node is positioned within **U** according to both its position in the scene and its resolution. All nodes for a given tree share a common position in the X dimension. However, they each have different resolution levels and are therefore positioned differently in the L dimension.

Figure 3.3(B), illustrates the set S for the sample application. For this example, the dataset contains a total of three trees, each with a unique position within the X dimension. The left-most tree is modeled at three discrete resolution levels. The remaining trees are modeled at four discrete resolution levels. This leads to a set S that contains eleven nodes, each of which is positioned within **U**. In the figure, you'll see three vertical arrangements, each of which represents a single tree at several resolutions.

Variable	Description
$\mathbf{U}$	Utility Space
$\mathbf{M}_i \subset \mathbf{U}$	Media Subspaces
$\mathbf{N} \subset \mathbf{U}$	Navigable Subspace
S	Set of all nodes
n_i	A node from the set S
$\mathbf{Pos}\{n_i\}$	The position of n_i
$\mathbf{State}\{n_i\}$	The state of n_i
$\mathbf{Arr}\{n_i\}$	The list of arriving edges at n_i
$\mathbf{Dep}\{n_i\}$	The list of departing edges at n_i
$\mathbf{Meta}\{n_i\}$	Application meta-data associated with n_i
B	The set of base nodes (nodes with a self-edge)
A	The availability front (all nodes in state *Available*)
E	Set of a edges
e_i	An edge from the set E
$\mathbf{Src}\{e_i\}$	The source node for e_i
$\mathbf{Dest}\{e_i\}$	The destination node(s) for e_i
$\mathbf{Clust}\{e_i\}$	The cluster to which e_i belongs
$\mathbf{Data}\{e_i\}$	The data associated with e_i
C	The set of all clusters
c_i	A cluster from the set C
$\mathbf{Edges}\{c_i\}$	The list of edges in c_i
$\mathbf{Cost}\{c_i\}$	The cost estimate for c_i

Table 3.1: The notation used to describe the Representation Graph (RG) model is summarized in this table. The symbols are grouped by concept.

Each node n_i has a position $\mathbf{Pos}\{n_i\} = (x, l)$ which uniquely identifies an individual element of information: a particular model of a tree.

3.2.4 Edges

Data dependencies between nodes are represented by directed *edges*. The set of all edges is noted as E. An edge $e_i \in E$ has both a source node, $\mathbf{Src}\{e_i\}$, and a list of destination nodes, $\mathbf{Dest}\{e_i\}$, as shown in Figure 3.5. For this discussion, the destination list is assumed to have just one member node. This is true for all edges except split-edges, which are defined later in this section.

An edge corresponds to a *depends-on* relationship between nodes, where $\mathbf{Src}\{e_i\}$ depends on $\mathbf{Dest}\{e_i\}$. Edges therefore express the syntactic relationships between nodes, regulating the order in which information can be resolved.

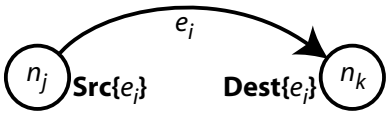

Figure 3.5: Edges are used to represent data dependencies. Typical edges express relationships between pairs of nodes. For example, edge e_i represents the data required to decode node **Src**$\{e_i\}$ given that node **Dest**$\{e_i\}$ is already resolved.

Edges serve double duty in the RG model. In addition to representing data dependence, an edge corresponds to the bytes of data needed to resolve **Src**$\{e_i\}$ when **Dest**$\{e_i\}$ has already been resolved. Recall that nodes, which correspond to elements, represent conceptual units of information. The actual data created by encoding a node is represented as an edge. The notation for the bytes of data associated with an edge is **Data**$\{e_i\}$.

While the data in a particular RG is represented by edges, it is often important to group edges into larger units to represent access-level restrictions. This is achieved using RG clusters, which are described in detail in Section 3.2.5. Each edge is assigned to a single cluster, noted as **Clust**$\{e_i\}$. Multiple edges may belong to the same cluster, but each edge belongs to exactly one cluster.

Data Versus Information

Nodes and edges can be used to represent a large number of complex data representation and encoding concepts. The definitions of nodes and edges in the RG model were derived to recognize the distinction between conceptual elements of information and the raw data used to resolve those elements.

The importance of this distinction is critical in the case, for instance, of multiple-descriptor media coding, where the same logical element of information is encoded multiple times. For example, consider the RG illustrated in Figure 3.6. Node n_i is drawn with two departing edges: e_j and e_k. The pair of edges represent two distinct paths for resolving the information at n_i. If the information for node n_j is already known, then **Data**$\{e_j\}$ can be used to resolve n_i. Likewise, if the information for n_k is already known, then **Data**$\{e_k\}$ can be used. The edges represent two different sets of raw bytes of data that can be used to resolve the same conceptual element of information.

In effect, the set **Dep**$\{n_i\}$ represents the set of options for decoding a node. In

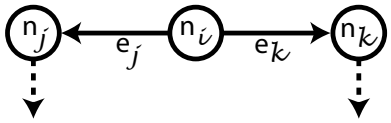

Figure 3.6: Nodes maintain a list of departing edges. For node n_i in this figure, the departing edge set $\mathbf{Dep}\{n_i\} = \{e_j, e_k\}$. The two edge represent paths for decoding node n_i. If both n_j and n_k are already known, either edge could be used to resolve n_i. If neither n_j or n_k are already known, then it is not possible to resolve n_i without first resolving one of the nodes on which it depends.

Figure 3.6, there are two options. In more complex data representations, there may be several alternatives for decoding a single node. However, for every node $n_i \in S$, there must be at least one decoding option. If not, the node would be unresolvable and should not be included in S. This observation leads to *Departing Edge Rule*, expressed in Equation 3.2. No such restriction exists on the number of arriving edges at a node, which can be zero or greater.

$$|\mathbf{Dep}\{n_i\}| \geq 1 \tag{3.2}$$

$$|\mathbf{Arr}\{n_i\}| \geq 0 \tag{3.3}$$

Self Edges

The Departing Edge Rule states that every node must have at least one departing edge because edges represent the actual data required to resolve a node. However, edges also represent dependencies between nodes.

This dual role leads to conflict for independent nodes that do not depend on any other node for resolution. An independent node n_i requires a departing edge, e_i, to comply with the the Departing Edge Rule. Clearly, $\mathbf{Src}\{e_i\} = n_i$. But to which node $\mathbf{Dest}\{e_i\}$ should e_i point?

Using the *depends-on* formulation of an edge, the correct answer should be no node at all because n_i is independent. However, every edge must have a destination node. To resolve this conflict, the RG model calls for $\mathbf{Dest}\{e_i\}$ to be defined as equal to $\mathbf{Src}\{e_i\}$. This is called a *self edge*, stemming from the fact that the edge implies that node n_i depends upon itself. A self-edge is shown in Figure 3.7(A).

Self-edges are required for nodes with no other edges due to the Departing Edge Rule. However, as with any node, a node with a self-edge can include in its departing

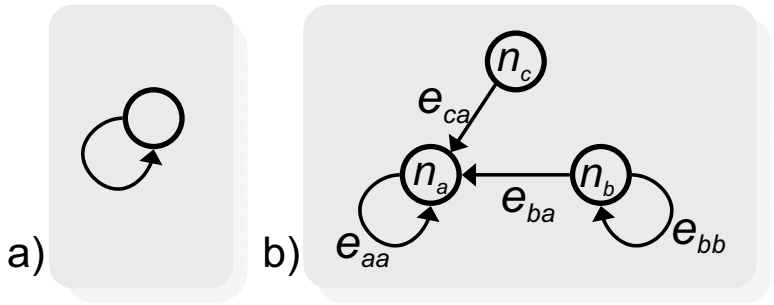

Figure 3.7: (A) A self edge is an edge where the source and destination nodes the same. (B) The set of departing nodes for an node can have a mixture of normal edges and self edges, as shown by n_b in the figure. If n_a is already known, then n_b can be resolved by either edges $e_b a$ or $e_b b$. If n_a is not already known, then $e_b b$ must be used.

edge list any combination of additional self-edges and traditional edges.

Figure 3.7(B) shows an example RG where node n_b has two edges in $\mathbf{Dep}\{n_b\} = \{e_b b, e_b a\}$. The self-edge $e_b b$ represents the data needed to resolve n_b without any prior knowledge. The edge $e_b a$ represents the data needed to resolve n_b assuming n_a is already resolved. This relationship could be common, for example, in applications that use predictive coding. Using node n_a as the predictive base for edge $e_b a$ might be more efficient and require less data than self-edge $e_b b$ for resolving n_b. However, it requires that n_a be previously resolved via edge $e_a a$. Including both edges provides the application with a choice between two paths of resolution.

A node that contains at least one self-edge within their departing edge list is considered a *base node* because it can be resolved without any prior knowledge. The RG defines the set of base nodes, $B \subset S$, as the set of all nodes with at least one self-edge. It is defined formally in Equation 3.2.4.

$$B = \{n_i\} : \forall n_i \in S | \exists e_j \in \mathbf{Dep}\{n_i\} | \mathbf{Src}\{e_j\} = \mathbf{Dest}\{e_j\} \tag{3.4}$$

The set of base nodes, B, is an important concept in maintaining node state. The implications on state maintenance for nodes within B are covered in Section 4.9 where adaptation via state transition is presented in detail.

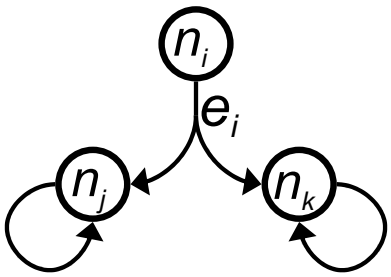

Figure 3.8: The RG model uses split edges to represent multiple predictor coding relationships. The split edge e_i has two destination nodes. As a result, decoding node n_i requires the information for both n_j and n_k, as well as the data assigned to edge e_i.

Split-Edges

Edges are used to represent the dependencies between nodes within an RG. As presented to this point, edges have been used to connect only pairs of nodes, with individual source and destination nodes. However, some representations may include more complex relationships, where multiple predictors are used in combination to encode a single node. I refer to this as *Multiple Predictor Coding (MPC)* .

MPC contrasts with multiple-descriptor coding in one very critical property. MPC uses multiple predictors in combination to form a single set of bytes for encoding a node. Multiple-descriptor representations include multiple encodings for a single node, each using independent predictive bases. As has already be presented, multiple-descriptor representations are supported by maintaining a list of departing edges for each node. However, MPC requires additional support.

The RG model uses *split-edges* to represent MPC. A split-edge e_i maintains two or more destination nodes for a single edge. This leads to the formal definition of $\mathbf{Dest}\{e_i\}$ as a list of nodes with one or more members. A split edge is illustrated in Figure 3.8, which shows edge e_i expressing the fact that node n_i can be resolved with $\mathbf{Data}\{e_i\}$ only if both nodes n_j and n_k have already been resolved.

Edges in the Illustrative Example

In the example computer graphics application, edges represent dependencies between nodes. Recall that nodes correspond to different resolution models of each tree. The sample dataset of multi-resolution trees has eleven nodes, and includes a single departing edge for each node. As shown in Figure 3.3(C), this leads to eleven edges.

For each tree, the model with the greatest error is fully encoded without any data

dependencies. This leads to the inclusion of a self-edge for nodes representing trees at the coarsest level. In this example, there are three nodes with self-edges. These three nodes make up the set of base nodes, B.

The dataset uses the nodes in B as predictive bases for encoding the more accurate models of each tree. The predictive relationships between nodes are expressed through the eight remaining directed edges that point from higher resolution nodes to lower resolution nodes. The final RG shows a series of three columns of connected nodes. Each column corresponds to the predictively coded models for a single tree, represented at multiple resolutions.

3.2.5 Cluster

Edges are used to represent blocks of data at a very fine granularity. A typical dataset contains a large number of nodes and the data required to resolve each node is represented as a separate edge. However, representations often restrict access to individual edges and only allow data access at a coarser granularity for performance reasons. Larger and less frequent data queries can usually be achieved far faster than a corresponding set of small and frequent queries.

The RG model allows representations to specify the granularity of data access through the use of *clusters*. Each cluster represents an atomic unit of data at the access level.

Cluster Properties

A cluster is defined as a group of one or more edges. Each cluster c_i contains a list $\mathbf{Edges}\{c_i\}$ of all edges assigned to it. The list must have one or more member edges. Any given edge belongs to exactly one cluster.

In addition, a cluster maintains a cost estimate $\mathbf{Cost}\{c_i\}$ which measures the cost of accessing the associated edges. This is the composite cost of accessing all edges in $\mathbf{Edges}\{c_i\}$ because the cluster requires that all data associated with it be loaded as a single unit. Costs are typically based on the size of the data associated with the cluster, but may also reflect other resource resource requirements associated with accessing the cluster.

Clusters are a critical component in the adaptation framework presented in Chapter 4 because they represent data at access level granularity. Adaptation is performed by deciding which cluster is the best to access at any point in time. For this reason,

the grouping of edges into clusters can play an important role in the performance of adaptive systems.

Clusters in the Illustrative Example

In the illustrative example application, the data for each edge can be accessed by the application independently. As a result, the application is provided with very fine-grained data access. The RG model expresses this design with eleven clusters, one for each edge in the representation. Each edge is assigned to a unique cluster.

In a more complete example application, it might be desirable to group edges together into larger clusters. Clusters are used more aggressively in the sample mappings provided in Section 3.3 to model to some more realistic application scenarios.

3.3 Mapping Application-Specific Data Structures to the RG Model

The RG model is not intended to serve as a replacement for existing data representations. Rather, it is designed as a generic and flexible model upon which the adaptation algorithms of Chapter 4 are defined. For that reason, it is expected that existing representations will be mapped to the RG model during the design stage of application development.

In this section, I discuss in detail the possible mappings for two pre-existing data representations. The first example presents a mapping for a more realistic computer graphics representation. The second mapping explores an audio and video streaming representation with two media subspaces.

3.3.1 Computer Graphics Models

Up to this point, example mappings for the RG model have been restricted to a simplified computer graphics application. In this subsection, a more realistic and fully featured computer graphics scenario is considered.

A typical computer graphics application contains a large number of computer graphics objects dispersed throughout a three-dimensional model space. The model space is canonically defined using three-dimensional Cartesian coordinates, (X, Y, Z), often

called the model space. Individual models, located at specific positions within the model space, represent physical objects using a series of geometric primitives (i.e. triangles).

Models are often stored using multi-resolution representations, such as levels-of-detail (Luebke, 2001) or the progressive mesh (Hoppe, 1996). This added dimension of adaptation allows the application to manage geometric complexity to match the available computational resources during rendering. In this example, it is assumed that the data representation contains several discrete models for each object at a number of levels-of-detail. Furthermore, it is assumed that the various resolution models for a given object are encoded using predictive coding techniques which improve data compression rates by introducing encoding dependencies between the models.

Utility Space Definition

The process of mapping an existing graphics dataset as described above to the RG model begins by defining the multidimensional utility space, $\mathbf{U}$. In this example, there is a four dimensional utility space defined by the union of the 3D model space and the resolution level of the individual geometric models. Defining the resolution level dimension as R, the 4D utility space becomes $\mathbf{U} = (X, Y, Z, R)$.

The three spatial dimensions (X, Y, Z) are all navigable because the application will maintain the user's viewpoint within the scene and will need to render geometric models based on the position of that viewpoint. This defines the navigable subspace, $\mathbf{N} = (X, Y, Z)$. The resolution dimension $\mathbf{R}$ is a static dimension with a static value of zero representing the lowest possible resolution for a geometric model. There is only one media type in this example, so there is only one media subspace, defined as $\mathbf{M_1} = \mathbf{U}$.

Node Definition

The second step in defining the RG is to specify the set of nodes, S. In this application, a node is defined for every resolution model of every geometric object. These nodes must also be positioned within $\mathbf{U}$.

Assuming there are n levels of resolution for each object, the RG would include columns of nodes, stacked vertically in the R dimension. For a given geometric object located at position (x, y, z) within the application's model space, there would be a series of nodes n_i, one for each of the n levels of resolution. The position of each node would be $\mathbf{Pos}\{n_i\} = (x, y, z, r_i)$ where r_i is the resolution value associated with node n_i.

Edge Definition

The encoding dependencies between different levels-of-detail of the same geometric object are represented using edges. Therefore, there is a group of edges for each vertical column of nodes within S. Each group contains a chain of edges pointing from the very highest resolution node down to the lowest resolution node. This corresponds to the chain of dependencies introduced by the predictive coding techniques used in storing the multi-resolutional models.

The node corresponding to the lowest resolution model for each object has a self edge indicating that it forms the predictive base for the entire column of nodes. The collection of all lowest resolution nodes forms the set of base nodes, B.

Every node $n_i \in S$ has exactly one departing edge. This property is common and reflects a data representation that does not utilize any multiple descriptor techniques. However, the number of arriving edges for each node varies. For all nodes $n_i \in B$, $|\mathbf{Arr}\{n_i\}| = 2$. For nodes corresponding to the highest resolution models, the arriving edge list is empty. For all other nodes, the arriving list contains just one edge.

Cluster Definition

The grouping of edges into clusters can be highly variable depending upon the expected data access patterns by the application. While every edge could be assigned to its own cluster, in may be desirable to group edges of similar resolution models that are located very close to each other within the application's model space. This would enable the application to more easily resolve groups of common resolution models from the same location in the virtual scene. However, it would also prohibit the application from individually resolving a specific node in the representation graph because clusters are considered atomic.

3.3.2 Audio and Video Streaming

Quality adaptation for audio and video has often been achieved using layered representations (McCanne et al., 1996; Rejaie et al., 2000). This case study considers a multicast-based application with independent video and audio streams, each of which are represented using a layered encoding.

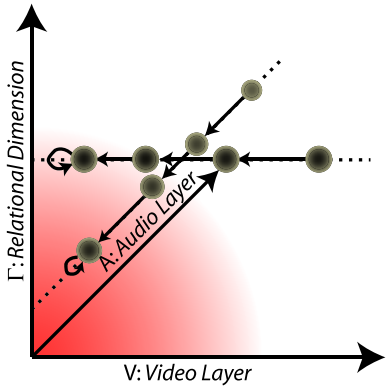

Figure 3.9: A Representation Graph (RG) designed for the audio/video sample application. The RG consists of eight nodes embedded within a three-dimensional utility space. The relational dimension Γ is used to manage cross-media adaptation.

Utility Space Definition

The RG for this application is defined within a three dimensional utility space, **U**. The first two dimensions symbolize the adaptive dimensions of the two media streams. One dimension corresponds to the video stream's quality layer dimension, V, while a second dimension corresponds the audio stream's quality layer dimension, A. Both of these dimensions are static because a user does not actively navigate through these dimensions. Typically, a user will always receive the lowest quality layer first with as many additional incremental layers as possible.

The two static dimensions are each independent and form a pair of one-dimensional media subspaces: $\mathbf{M_1} = V$ for the video stream and $\mathbf{M_2} = A$ for the audio stream. The media subspaces are combined via a single relational dimension, Γ. The relational dimension is used to facilitate inter-media trade-offs within the utility space.

The two static dimensions and single relational dimension form the overall utility space $\mathbf{U} = (V \times A \times \Gamma)$. This utility space is somewhat unique in that none of its dimensions are navigable. As a result, $\mathbf{N} = \emptyset$ and **U** is considered *unnavigable*. An unnavigable utility space is considered a special case for the RG model and, as will be covered in Chapter 4, allows far less flexibility during adaptation.

Node Definition

In this example, both streams are encoded independently using a layered encoding. The RG contains one set of nodes for each stream, each of which are located within a

single media subpace of **U**. Assuming the video stream has n_V layers, there would be n_V nodes positioned within M_1. Similarly, assuming the audio stream has n_A layers, there would be n_A nodes positioned within M_2. Within each subspace, there would be a single column of nodes arranged within the corresponding layered dimension (V for M_1, A for M_2).

Edge Definition

The layered encoding method used for each stream introduces a set of edges for each media subspace. As described in Section 3.3.2, there is a set of nodes within each individual subspace M_i. Within a subspace, these nodes are arranged in a column. Nodes within a column are chained together with edges which correspond to the encoding dependencies between neighboring layers of the media stream.

The node corresponding to the lowest quality layer for each media stream has a self edge indicating that it forms the predictive base for the entire set of nodes. Following the rules of layered encoding techniques, each stream has only a single base layer. The set of all base layer nodes (i.e. nodes with self edges) forms the set of base nodes, B.

As in the previous example, every node $n_i \in S$ has exactly one departing edge. The number of arriving edges for a node varies depending on which layer the node represents. For all nodes $n_i \in B$, $|\mathbf{Arr}\{n_i\}| = 2$. For nodes corresponding to the highest quality layer, the arriving edge list is empty. For all other nodes, the arriving list contains one edge.

Cluster Definition

Each quality layer must be made independently accessible to the application. This allows the greatest amount of flexibility in choosing how many layers to access any any point in time. The RG model expresses this design requirement by assigning each edge to a unique cluster.

3.4 Summary

In this chapter, I described the RG data structure abstraction. The RG is a graph-based data model that can express both the syntactic and semantic relationships between the many individual elements of information that, as a whole, compose a larger multimedia object.

The RG model employs a graph, composed of nodes and edges, embedded in a multidimensional utility space. The dimensions of the utility space correspond to the adaptive dimensions inherent in the media object. The position of the RG nodes are determined by the semantic relationships between nodes within the dimensions of the utility space. Edges are used to connect dependent nodes based on syntactic relationships, such as encoding dependencies. In addition, edges correspond to the actual data required to resolve their associated nodes. The RG model also defines the notion of a cluster as a group of semantically similar edges. Clusters are units of data that are accessed atomically.

The RG model is a powerful abstraction that can model a large number of multimedia datasets. It is highly flexible and capable of representing a wide variety of data relationships. At the same time, it is able to represent a number of core data representation concepts that are integral to the adaptation process presented in Chapter 4. I highlighted this flexibility by presenting two case studies where two different real-world data representations were mapped to the RG model.

In the following chapters, the RG model is used as the foundation on which to build the algorithms required for both adaptation and scalable delivery. The RG is ideally suited to these two tasks due to its power to model the semantic and syntactic data relationships, and its generic definition which makes it relevant to a wide range of data structures.

Chapter 4

Adaptation

Data adaptation is a critical system task in any non-linear media streaming system. The flow of data from source to receiver must be controlled to satisfy both the receiver's application-level requirements and the limited communication resources available to supply the receiver with the needed data.

In this chapter, I first discuss the general problem of data flow adaptation and enumerate many of the challenges in performing adaptation in complex systems. I then present my general framework for expressing multidimensional adaptation (Gotz and Mayer-Patel, 2004). This framework regards adaptation as a maximization problem in which the goal is to maximize the utility of the received data while simultaneously minimizing the access cost. The framework provides efficient algorithms for determining the appropriate behavior, as well as mechanisms for expressing application-specific data requirements.

I begin by framing adaptation as a maximization problem. I then define the primary inputs to that maximization problem: a set of adaptation structures that allow for the expression of both data availability and application-level data requirements. After covering the adaptation structures, I present the concept of cost and utility metrics. These metrics are used to evaluate the utility and cost of access to individual elements of information, respectively.

The cost and utility metrics are both used to define the Utility-Cost Ratio (UCR). The UCR is a quantitative measure whose value serves as the quantity to be maximized by the adaptation algorithm. I will present both a formal definition of the UCR, as well as the iterative maximization algorithm that drives the adaptation process.

Following the discussion of the UCR, I review the tools and structures that allow applications to express application-level preferences which can be incorporated into the

computation of the UCR. I then discuss node states and state transitions, which are used to keep track of the dynamic system state as data is acquired by the adaptive application.

Finally, I present a detailed review of the iterative adaptation algorithm. This algorithm brings together all of the concepts and structures presented throughout this chapter to efficiently compute the appropriate adaptive behavior based on current system and application requirements.

4.1 Challenges in Multidimensional Adaptation

Advances in storage and processing technologies now allow scientists to capture, simulate, or create immense collections of data. Unfortunately, advances in networking and display technologies, while impressive, have not kept pace. Although one can build multi-gigabit networks, achieving those line speeds end-to-end remains almost impossible. Similarly, high-resolution display technologies have improved but essentially remain within a factor of ten of the mega-pixel displays one has enjoyed for nearly twenty years.

As the gap between the amount of data one can capture, store, and process and the resources one has to transmit and view that data increases, the problem of adaptation becomes increasingly central to the performance of high-bandwidth and high-volume applications such as data visualization, tele-immersion, and media streaming.

The challenge that system engineers face when tackling the adaptation problem in whatever context it appears is twofold. First, how can one compactly and intuitively specify adaptation policy to support specific user-level goals? Second, given a particular adaptation policy and set of user-level goals, how can one efficiently evaluate that policy relative to available resources and the way the data are represented and organized?

4.2 Ad Hoc Solutions to Adaptation

Adaptation techniques, algorithms, and frameworks have largely been developed within the context of a specific application or data type. For example, Geographic Information Systems (GIS) applications use multi-resolutional data for visualization and representation of terrain information (Floriani and Magillo, 2002). In graphics, multi-resolutional geometry information is used to create Hierarchical Levels of Detail (HLODs) (Luebke, 2001). These are used to dynamically adjust model complexity in order to achieve a

target rendering rate. In multimedia, layered media encodings are used to dynamically scale media bit-rates to match current network conditions (Rejaie et al., 1999).

While ad hoc application-specific methods may be effective when the dimensionality of the adaptation problem is limited, the task can become overwhelming as the number of dimensions increases. The same is true when an adaptation decision must negotiate between data sources of fundamentally divergent natures, as is often required in multimedia applications.

4.3 General Models for Adaptation

The complexity of ad hoc solutions to the problem of adaptation can become very difficult to manage in the face of both heterogeneous data types and growing degrees of freedom in the adaptive process. Expressing adaptation policy in a rule-based manner, for example, becomes painful as the number of possible trade-offs and permutations grows.

For these reasons, several researchers have begun exploring more general models of adaptation. This work ranges from a series of design principles for adaptive systems (McIlhagga et al., 1998) to fully developed algorithmic models computing adaptive behavior (Walpole et al., 1999). A more detailed review of previous work in developing general models of adaptation is included in Chapter 2.

In this chapter, I present a novel framework for multidimensional adaptation that fits within this class of general models of the adaptation problem. The framework allows for the compact expression of adaptation policies, and provides algorithms for dynamically and efficiently evaluating utility with respect to changing system conditions.

4.4 Adaptation as Maximization

In general, data adaptation is a process that, given a set of constraints and preferences, attempts to obtain the most useful data available at the lowest possible cost. It is in essence a maximization problem, where the term to be maximized is the utility of information over the cost of accessing it.

The aim in my framework is to define an algorithm to efficiently and interactively evaluate both the cost and utility of the available information. The evaluation must be interactive because the system's constraints and preferences are dynamic and their changing values must be incorporated into the adaptation algorithm.

Framing adaptation as a maximization problem is not the only possible approach. For example, many systems have developed complex sets of rules to determine a system's proper adaptive behavior as conditions change. However, a maximization-based solution provides a critical benefit: reduced complexity.

The framework I present for maximization-based adaptation allows a system designer to bypass the complex enumeration of rules that govern proper behavior at all possible system states. This task can quickly become onerous in complex systems where the number of possible states may grow exponentially.

Instead, I employ mathematical functions defined to measure both the utility and cost of information in a far more compact and intuitive manner. At runtime, the proper adaptive behaviors can be determined simply by iteratively evaluating quantitative metrics, then choosing whichever action maximizes the utility of received information per unit of access cost.

4.5 Adaptation Structures

The adaptation process determines which data is must useful and available at the lowest cost given a set of given a set of constraints and preferences. The process must therefore have access to two classes of information. First, the adaptive process must be aware of the overall data structure to know which data can be resolved given the current set of known information. This depends on the syntactic structure of the data representation. Second, there must be data structures that express system preferences and requirements to determine which of the available data is most useful given the current system state. This depends on the semantic relationships between data elements on the current state.

There are a number of structures used to represent these two classes of information. Each of these structures, used as input to the adaptation algorithm, are described in this section. Coverage of the adaptation algorithm itself is presented in Section 4.10.

4.5.1 Data Structures

The adaptation algorithm, which decides which data to obtain given current system conditions, must have a detailed description of the available data in order to properly function. The description is used to determine what data is available and what its semantic and syntactic relationships are at the current point in time. Given this information, the adaptation algorithm can then make an intelligent choice about which of

the available data elements would be best to obtain.

I use the RG abstraction, defined in Chapter 3, to represent the overall dataset with which the adaptation algorithm must work. The RG models both the semantic and syntactic relationships between individual elements of information. These relationships are critical inputs to the adaptation algorithm.

The adaptation process must examine the semantic value of each data element in the dataset, deciding which is most relevant to the current application interests. This is made possible by the embedding of semantic information within the RG. Semantic information is expressed by the position of individual RG nodes within the utility space.

Similarly, syntactic relationships between data elements may limit the order in which data elements can be obtained. These relationships must be respected by the adaptation algorithm. Again, this is made possible by the embedding of syntactic information within the RG using edges.

4.5.2 Preference and Constraint Structures

The primary motivation for developing adaptive systems is the need to match the flow of information to dynamic application preferences and constraints. Therefore, it is critical to provide structures that both (1) allow an application to express preferences and constraints, and (2) allow the adaptation algorithm to incorporate them into its decision process.

There are two structures provided by my adaptation framework for expressing application preferences. The first structure, the prediction vector, is used to keep track of the current application requirements and, if available, predictions of future requirements. The second structure is the alpha vector, used to manage trade-offs across adaptive dimensions.

The Prediction Vector

An adaptive application will typically exhibit dynamic data requirements that change over time. These changing requirements are communicated to the adaptation framework through the *prediction vector*, noted as $\vec{p}$.

The prediction vector is a list of one or more values that represent the current and future application requirements. Each element $\vec{p}[i]$ of the prediction vector pairs a point $\mathbf{Pos}\{\vec{p}[i]\}$ in $\mathbf{U}$ with a confidence value $\mathbf{Con}\{\vec{p}[i]\} \in (0, 1]$.

***Point of Interest*:** The first element in the vector, $\vec{p}[0]$, has a special significance

Variable	Description
RG	Representation Graph
$\vec{p}$	Prediction Vector
$\vec{p}[i]$	Element i of $\vec{p}$
$\vec{p}[0]$	Point of Interest
$\mathbf{Pos}\{\vec{p}[0]\}$	Position of $\vec{p}[i]$
$\mathbf{Con}\{\vec{p}[0]\}$	Confidence value for $\vec{p}[i]$
$\vec{\alpha}$	Alpha Vector
$\vec{\alpha}[i]$	Element i of $\vec{\alpha}$

Table 4.1: The notation used to describe the structures used by the adaptation framework is summarized in this table. The symbols are grouped by concept.

and is termed the *point of interest*. This is the only required element in the prediction vector and is used to represent the current application preferences. The preferences are expressed by positioning the point of interest within the semantically meaningful utility space. Because $\mathbf{Pos}\{\vec{p}[0]\}$ corresponds to the current system conditions, $\mathbf{Con}\{\vec{p}[0]\}$ is always set to one to signify full confidence.

For example, in the illustrative application first described in Section 3.2.1, the point of interest would be located within the two-dimensional utility space. The position of $\vec{p}[0]$ would be determined by the user's current viewpoint within the navigable dimension X, as well as the static value of zero for the static dimension L. By definition, $\mathbf{Con}\{\vec{p}[0]\} = 1$. As the user's viewpoint moves through the scene, the application would update the value of $\mathbf{Pos}\{\vec{p}[0]\}$ to reflect its changing point of interest.

***Predictions*:** While the point of interest is sufficient to express the current application preferences, is often desirable to anticipate future application needs. Depending on the application, it may be possible to exploit knowledge of likely future preferences to more effectively adapt the flow of information to meet future needs.

The adaptation framework supports the incorporation of predicted future points of interest as part of the prediction vector. An application can have zero or more predictions. For a prediction vector of length n, there are $n-1$ predictions $\vec{p}[i]$, where $1 \leq i < n$. The prediction vector is ordered by confidence such that Equation 4.1 is satisfied.

$$\mathbf{Con}\{\vec{p}[i]\} \leq \mathbf{Con}\{\vec{p}[i-1]\}, \forall i \in \{1, \cdots, n-1\} \tag{4.1}$$

As with the point of interest, each $\vec{p}[i]$ has an associated position within $\mathbf{U}$. However,

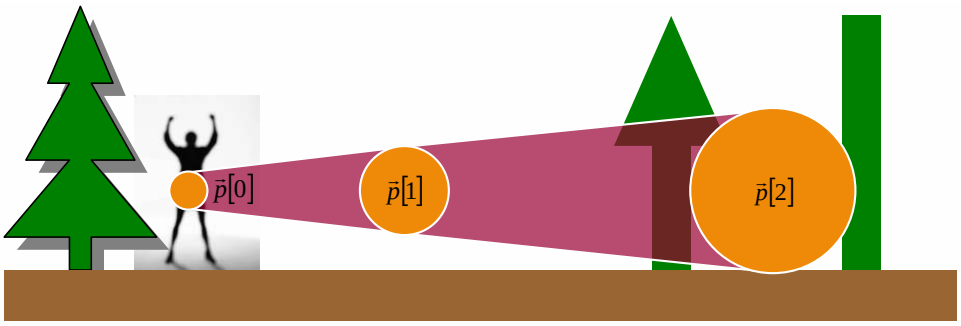

Figure 4.1: A prediction vector of length three would contain the current point of interest, followed by two locations that correspond to predicted points of interest in the future. The point of interest, $\vec{p}[0]$, is known with absolute certainty. The predictions $\vec{p}[1]$ and $\vec{p}[2]$ are given with increasing uncertainty, illustrated by the larger circles in the figure.

unlike the point of interest, the confidence value paired with each prediction is given a value in the range $(0, 1]$ to reflect certainty in the accuracy of the prediction.

In the illustrative application, in which a user is navigating through a one dimensional computer graphics scene, the future position of the viewpoint might be predicted from the direction and speed of the user's movements. This technique may yield highly accurate results when predicting the viewpoint location over a short time window, but less accurate results at a broader time scale.

Such a system could incorporate both the long and short term predictions into the adaptation process by adding two entries onto $\vec{p}$. The position of the short term prediction is more accurate, and following Equation 4.1 belongs in $\vec{p}[1]$. Conversely, the long term prediction is less accurate and the position and confidence values are stored in $\vec{p}[2]$.

This example therefore has a prediction vector of size three which contains both the current point of interest and two predictions of future requirements. Figure 4.1 illustrates the use of $\vec{p}$ in the sample application.

Alpha Vector

Adaptive applications must often manage trade-offs between multiple dimensions of adaptivity. For example, a video streaming application may be able to adjust quality in two ways: lower the frame rate and reduce the video's resolution. In practice, an application may have a large number of options for adapting the flow of data.

The number of adaptive dimensions for a particular application depends on the

nature of the dataset. For data that is modeled using the RG model from Chapter 3, the number of dimensions corresponds directly with the dimensionality of the RG's utility space. The adaptation framework takes advantage of this correspondence and takes a spatial approach to managing inter-dimensional trade-offs.

The utility space **U** of a RG defines a spatial representation of the dimensions of adaptivity. As will be presented in Section 4.6, the metric that measures the usefulness of individual data elements within the RG is defined as a function which operates on the space **U**. For this reason, inter-dimensional trade-offs are expressed as spatial operations that scale the representation graph within **U**. In this subsection, I describe the *alpha vector*, the structure used to store the dimensional scale factors. The power of the alpha vector will become more clear after the discussion of utility and cost metrics in 4.6.

The alpha vector, noted as $\vec{\alpha}$, is a list of linear scale factors to be applied to the utility space **U** prior to the calculation of the utility and cost metrics. As such, $\vec{\alpha}$ contains one element for every dimension in **U**. A RG with a dimensionality of n will have an alpha vector of the same length. Each element of the vector can be set to any real number.

The identity alpha vector, where $\vec{\alpha}[i] = 1, \forall i$, produces no scaling effects on **U**. Any changes to individual elements of $\vec{\alpha}$ scales **U** and introduces a change in the relative positions the RG's nodes. Because a node's position defines its semantic meaning, the alpha vector essentially warps the semantic nature of the data set to reflect the relative importance of each dimension in **U**. A more through description of the alpha vector's impact on adaptation is presented in Section 4.8.

4.6 Utility and Cost Metrics

The adaptation framework defines the process of adaptation as a maximization problem. The goal of adaptive data systems is to obtain the most useful data available at the lowest possible cost. Therefore, two critical components of any adaptive system are the metrics that define what data is useful and how much it costs to retrieve that data.

While extremely critical, these metrics are necessarily application-specific. Even two very closely related applications may have vastly different criteria for evaluating the utility or cost of information. For example, a video streaming application for NASA transmitting information from a rover on Mars might require metrics that are quite different from a video streaming application to transmit feature films for viewing on a

personal computer.

Recognizing the need to allow individual systems to define application-specific metrics, the adaptation framework specifies general models for both *utility metrics* and *cost metrics*. These models define both the interface and conceptual design of the abstract metrics, while leaving the specific implementation details to be dealt with by an application designer.

4.6.1 Utility Metric

The adaptation framework defines an abstract utility metric, $UtilMetric$, used to evaluate the relative usefulness of individual elements of information. Recall that the RG model expresses elements of information as individual nodes. I can therefore define the utility metric as a function of a single node, n_i. The value returned corresponds to the utility of the information corresponding to the node in question given the current system requirements and preferences. Given such a metric, the adaptation algorithm can compare the relative utility of all nodes.

There are several other inputs to the $UtilMetric$ function. It evaluates the utility of a single node as a function of the node itself, the overall utility space ($\mathbf{U}$), the set of all nodes in the RG (S), the prediction vector ($\vec{p}$), and the alpha vector ($\vec{\alpha}$). The metric returns a scalar value $u \in \Re$.

$$UtilMetric(n_i, \mathbf{U}, S, \vec{p}, \vec{\alpha}) = u \in \Re \tag{4.2}$$

Utility and Distance

While the implementation of a utility metric is necessarily application-specific, the $UtilMetric$ abstraction is based on a general concept of measuring utility as a distance within the space used to define the RG. This is the reason that the space which contains the RG is called the *utility space* (see Section 3.2.2).

Recall that every element of information in a particular dataset is represented by a unique node, n_i, positioned within the utility space, $\mathbf{U}$. For an individual node, $\mathbf{Pos}\{n_i\}$ places the node within $\mathbf{U}$ based on the semantic meaning of the data associated with the node.

At the same time, the prediction vector, $\vec{p}$, consists of a series of points, similarly located within $\mathbf{U}$, which specify the current application data interests within the semantically meaningful space. In general, the distance between a node and the prediction

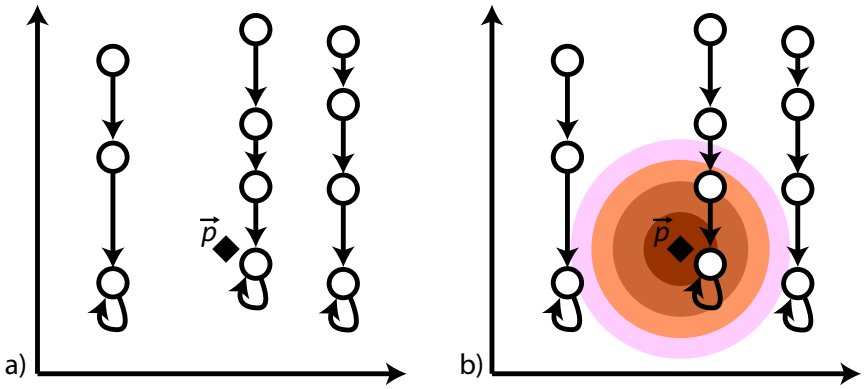

Figure 4.2: (A) The representation graph for the illustrative application includes 11 nodes in a two-dimensional utility space, **U**. A prediction vector $\vec{p}$ is positioned within **U**. (B) A simple utility metric can be defined on the euclidean distance between $\vec{p}$ and each node n_i. Nodes located closer to $\vec{p}$ are considered more useful.

vector is considered inversely related to utility of the node.

The specific metric implementation can vary widely between applications. For example, an application may employ simple Euclidean distance to measure utility. Alternatively, the metric may be based on a complex non-linear function of the positions for both n_i and $\vec{p}$.

However, regardless of the specifics of a particular utility metric, the general model remains the same. The $UtilMetric$ returns a scalar value produced as a result of an application-specific function defined over the utility space, **U**.

For example, consider the illustrative application discussed throughout both this chapter and Chapter 3. A RG has already been defined, including a utility space (Section 3.2.2) and set of nodes (Section 3.2.3). In addition, assume that the application is using a prediction vector of length one, containing only the current point of interest and zero predictions. Figure 4.2(a) illustrates both the RG and $\vec{p}$.

Recall that $UtilMetric$ measures the utility of an individual node, n_i. The simplest utility metric for the sample application would be a Euclidean distance measure based on the distance between the point of interest, $\vec{p}[0]$, and n_i. The example $UtilMetric$ is shown in Equation 4.3, where the $Dist$ function returns the Euclidean distance between two points.

$$UtilMetric(n_i, \mathbf{U}, S, \vec{p}, \vec{\alpha}) = \frac{1}{Dist(\mathbf{Pos}\{n_i\}, \mathbf{Pos}\{\vec{p}[0]\})} \tag{4.3}$$

There are several important components of the proposed $UtilMetric$. First, the metric returns the reciprocal of the distance, rather than the distance itself. The inversion is due to the fact that the conceptual model of the utility metric specifies an inverse relationship between distance in $\mathbf{U}$ and utility. Nodes positioned close to the prediction vector are of the highest value, while nodes far away are of significantly lower utility.

A second observation regarding the example metric is simplicity of the equation. In the sample metric, utility is calculated only locally, ignoring the location of all other nodes in S. Similarly, the equation does not make use of the alpha vector or any predictions in $\vec{p}$.

However, for more sophisticated applications, these additional $UtilMetric$ parameters can be very powerful tools for expressing adaptive policies. A more thorough discussion of these structures and techniques for incorporating them into the calculation of the $UtilMetric$ is presented in Section 4.8.

Relational Dimensions

The utility metric is defined as a spatial function over the overall utility space for a given dataset. As described in Section 3.2.2, a multimedia dataset's overall utility space, $\mathbf{U}$, can be broken down into individual media subspaces, $\mathbf{M}_i$, where each subspace contains the representation graph for an independent set of media objects.

For example, an audio/video application could be represented with an RG containing a utility space composed of two media subspaces, $\mathbf{M} = \{\mathbf{M}_{audio}, \mathbf{M}_{video}\}$. Each media type is represented in a separate media subspace, which is by definition independent from all other subspaces in $\mathbf{M}$. The independence of each media subspace reflects the fact that each subspace corresponds to an independent set of media objects.

However, the utility metric must be capable of determining the relative utility of nodes in the RG across media subspaces. For example, an audio/video application would need the ability to measure the relative benefit of obtaining additional video data versus the benefit of additional audio data.

The essential task of cross-subspace utility evaluation is made possible by including *relational dimensions* within the utility space. For a utility space with multiple media subspaces, one or more relational dimensions, $\mathbf{R} = \{\mathbf{R}_1, \cdots, \mathbf{R}_j\}$, are required to specify the relative utilities of the competing media objects.

Because they share no dimensions, individual media subspaces are detached from one another and exist wholly independent from each other. However, the utility eval-

uation is a global metric that must determine the most useful node across all media subspaces. To tie the set of independent media subspaces into a single unified space, the set of nodes for each media subspace is given a position along a common relational dimension.

In effect, this extra dimension ties together the subspaces $\mathbf{M_i}$ into a unified utility space $\mathbf{U}$. We can therefore define $\mathbf{U}$ as the union of all subspaces in $\mathbf{M}$ and all relational dimensions $\mathbf{R}$.

$$\mathbf{U} = \mathbf{M} \cup \mathbf{R} \tag{4.4}$$

Relational dimensions are especially important because they offer a direct method for managing the often dynamic trade-offs that between the set of possible media types that must be managed by an adaptive multimedia application.

For example, in the audio/video application discussed above, an adaptive decision must be made by choosing to obtain additional video data or additional audio data. The relative importance of audio versus video could be highly dynamic. Consider an application that sends the audio and video of a classroom environment across a network. When an instructor is writing on the board, video data will be highly critical. When the instructor turns to face the class and begins lecturing, video data will become less important and audio data will become critical.

Relational dimensions make it easy to express these highly dynamic relationships between media subspaces. An application can simply map a change in relative utility to a change in the position along the relational dimension of each subspace's RG. The new positions along the relational dimension directly effect the distances measured within the utility space, thereby influencing the utility metric itself.

Figure 4.3 illustrates the use of a relational dimension to control the cross-media adaptation policies of the audio/video application example. In Figure 4.3(A), the audio subspace is located closer to the point of interest than the video subspace. This would lead an application to bias its adaptation toward retrieving additional audio data. In contrast, Figure 4.3(B) shows the video subspace located closer to the point of interest, indicating an application-level preference for video data.

4.6.2 Cost Metric

The adaptation framework defines an abstract cost metric, *CostMetric*, used to evaluate the cost of obtaining the data needed to resolve an individual element of informa-

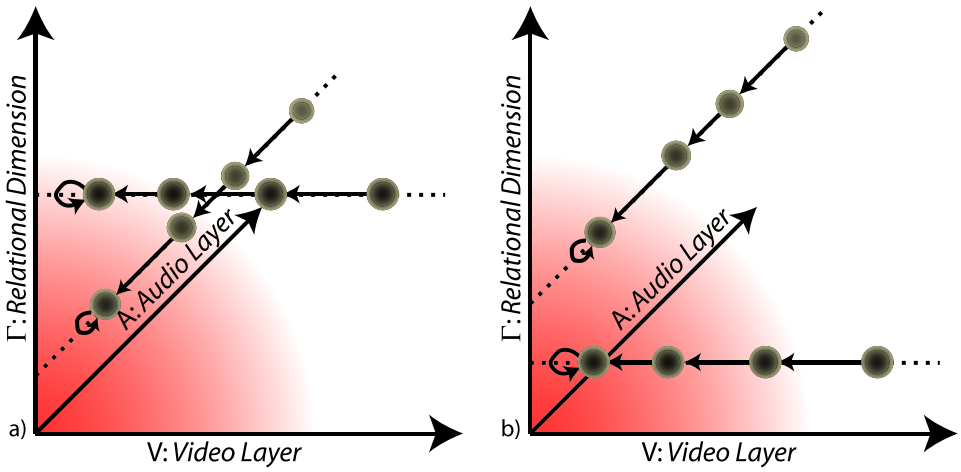

Figure 4.3: The RG in this figure shows a three-dimensional utility space containing eight nodes with a point of interest located at the origin. The RG is defined for an audio/video application. On the left (A), the audio stream nodes are positioned closer to the point of interest, biasing the utility metric toward themselves. On the right (B), the positions of the two media subspaces within Γ have swapped, leading to a utility bias toward the video nodes.

tion. The RG model expresses elements of information as individual nodes. In order to resolve the information associated with a given node n_i, the data associated with the node must be obtained by traversing one of the edges in the set of departing edges $\mathbf{Dep}\{n_i\}$.

The metric is therefore defined as a function of a single node, n_i. The value returned corresponds to the cost of accessing the data required to resolve the information corresponding to the node in question.

The specific implementation of this abstract metric is application specific. For example, one application may measure cost in terms of the raw number of data bytes to be read. In contrast, another application may be concerned about monetary costs and would strongly prefer data from lower cost content providers.

However, regardless of the specific formulation of the cost metric, it returns a scalar value $c \in \Re$ which measures the relative cost of accessing the data required to resolve node n_i.

$$CostMetric(n_i) = c \in \Re \tag{4.5}$$

Any application-specific implementation of the cost metric must take into account

the possibility of multiple-descriptor coding techniques. The RG model supports multiple-descriptor coding by allowing an individual node n_i to maintain multiple departing edges. Each edge $e_j \in \mathbf{Dep}\{n_i\}$ corresponds to a unique unit of data which can be used to decode n_i.

The cost of resolving node n_i will vary along each edge. Furthermore, the most cost effective edge may change dynamically with changes to the system's state. As a result, the *CostMetric* must evaluate the range of decoding options represented by the set $\mathbf{Dep}\{n_i\}$, and returns a scalar value reflecting the alternative with the minimum cost value.

4.7 The Utility-Cost Ratio

The adaptation process is guided by both the utility and cost metrics present earlier in this chapter. The two metrics are combined into a larger metric which is used to drive the maximization-based adaptation algorithm. The combined metric, which produces a single scalar measure to drive adaptation, is called the *Utility-Cost Ratio*. This section provides a formal definition of the UCR metric and outlines the iterative maximization algorithm used to drive adaptation.

4.7.1 UCR Formulation

Adaptation is a process that must satisfy two competing aims. First, an adaptive algorithm must maximize the utility of the information delivered to the application. Second, it must minimize the cost of accessing the information to allow the application to perform in the face of limited resources.

The previous section defined abstract utility and cost metrics which provide mechanisms for evaluating the both utility and cost of accessing an individual element of information. In this section, the two metrics are combined to form the UCR, as shown in Equation 4.6.

$$UCR = \frac{UtilMetric(n_i, \mathbf{U}, S, \vec{p}, \vec{\alpha})}{CostMetric(n_i)} \tag{4.6}$$

The UCR is used as the scalar value to be maximized by the adaptation algorithm. The UCR is simply the ratio of the utility of information over the cost of accessing it. The UCR expresses the fundamental need for adaptation: to achieve the goal of obtaining the most useful information at the lowest possible cost.

The UCR is an effective metric for adaptation because it produces a numerical expression of the cost-to-benefit trade-off that must be negotiated by any adaptive application. For example, the UCR will give a high score to a useful and inexpensive element of information. At the same time, an element of information that has both a low utility value and high cost of access will be scored very poorly.

More importantly, the UCR will provide relatively similar scores to both a useful but expensive element, and a cheap but less useful element. In this event, the maximization-based adaptation algorithm will simply choose to resolve whichever element was scored highest by the UCR.

In a rule-based adaptive system, this case would require specific rules designed to decide between the two adaptive options. For large systems with multiple dimensions of adaptive behavior, the rule set could grow exponentially in complexity. In contrast, my computational model for adaptation simply computes the UCR for all possible adaptive behaviors and performs that task with the highest score.

4.7.2 Iterative UCR Evaluation

The UCR is a dynamic measure which changes over time to reflect changes in an application's internal state and the available system resources. It must therefore be computed iteratively, where each evaluation provides an estimate of both the utility and cost of a potential adaptive step based on a momentary snapshot of the system's status. A detailed description of the iterative adaptation algorithm is provided in Section 4.10 following the definition of node states and state transitions in Section 4.9.

4.8 Expressing Application-Level Preferences

Application-level preferences, such as trade-offs between different media objects or changes in data interests, can be expressed via several of the data structures presented earlier in this chapter. These include the prediction vector and the alpha vector. This section presents an overview of the power of these two structures in the context of the UCR.

4.8.1 Using the Prediction Vector

The prediction vector, $\vec{p}$, represents both the current application preferences as well predictions of future needs. The vector's first element $\vec{p}[0]$, knows as the point of

interest, corresponds to the immediate system conditions.

The point of interest along with the remainder of the prediction vector serve as the primary structure through which the application's system conditions are mapped to the spatial metaphor used to evaluate the utility of information. As described in Section 4.6.1, the utility metric is defined as a spatial function that measures how valuable individual elements of information are to the application as a function of the distance between the current prediction vector and the position of the information's corresponding node.

There is great power in the primacy of the prediction vector in calculating the utility of information. The relative utility calculated for an enormous set of RG nodes can be quickly altered simply by changing the position of the point of interest. In addition, the inclusion of future application preferences can be exploited to alter the utility metric so that nodes more likely to be needed in the future are located closer to the prediction vector.

For example, consider the prediction vector designed for the illustrative application in Section 4.5.2. The proposed $\vec{p}$ includes a point of interest located within the two-dimensional utility space corresponding to the position of the user's current viewpoint within the navigable dimension X. The position of $\vec{p}[0]$ within the static dimension L is equal to the static value of zero. In addition, $\vec{p}$ contains two predictions of future viewpoint positions with decreasing confidence values.

The sample application's prediction vector, positioned within the utility space, is illustrated in Figure 4.4. Notice the similarity between this figure, showing the spatial utility model used by my adaptation framework, and the pictorial representation in Figure 4.1. In Figure 4.4(a), the user's viewpoint is positioned to the left of the scene, making the left-most nodes closer to $\vec{p}$. As a result, the left-most nodes would be scored with a higher utility value.

In Figure 4.4(b), the viewpoint has transitioned to the center of the scene, increasing the utility of the nodes near the center while decreasing the utility of the nodes on the left. The inclusion of predictions that show that the viewpoint will continue to move to the right allows the utility metric to favor nodes to the extreme right over nodes to the extreme left.

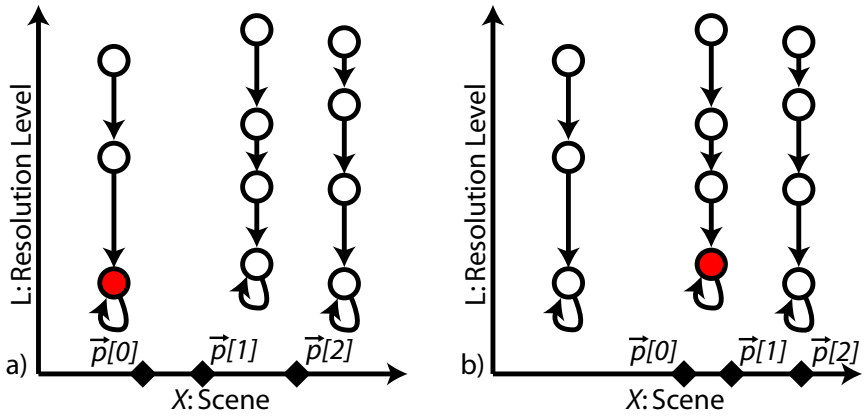

Figure 4.4: The sample application's prediction vector $\vec{p}$ is located within the two-dimensional utility space. On the left (A), $\vec{p}$ is positioned to reflect a user on the left side of the scene moving toward the right. On the right (B), $\vec{p}$ shows the user has moved to the middle of the scene and continues to move rightward. In each figure, the most useful node is shaded gray

.

4.8.2 Using the Alpha Vector

The spatial nature of the utility metric can also be exploited to manage inter-dimensional trade-offs. This is done via the alpha vector. As presented in Section 4.5.2, $\vec{\alpha}$ is a list of scale factors used to transform the utility space prior the calculation of the utility metric. By geometrically transforming the space in which the nodes are embedded, the alpha vector can directly the impact utility metric by changing the relative importance of each utility dimension within the overall calculation.

Consider the illustrative application and the utility space shown in Figure 4.5(a). The figure shows both the RG and $\vec{p}$ drawn using an identity alpha vector, where $\vec{\alpha}[i] = 1, \forall i$. The iso-utility line shown in the diagram corresponds to a distance from $\vec{p}$ at which all nodes would have an equal utility value.

Changes to the alpha vector alter the positions of nodes within $\mathbf{U}$, thereby changing the locations of nodes with respect to the iso-utility line. For example, Figure 4.5(b) shows the same RG and $\vec{p}$ using $\vec{\alpha} = \{1, 2\}$. Note that the altered iso-utility line places a far greater importance on data in the X dimension with respect to the L dimension.

The alpha vector $\vec{\alpha} = \{1, 2\}$ therefore increases the utility of geometric objects located at nearby positions in the scene while decreasing the utility of higher-resolution data. Conversely, an alpha vector of $\vec{\alpha} = \{2, 1\}$ would alter the utility calculation to

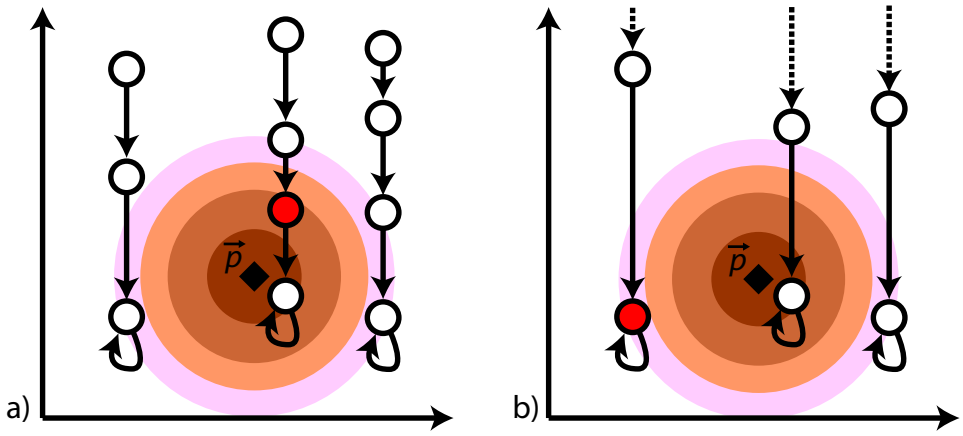

Figure 4.5: The alpha vector can be used to manage trade-offs between adaptive dimensions. (A) An alpha vector of $\vec{\alpha} = \{1, 1\}$ places equal value on each of the two dimensions. (B) Changing the alpha vector to $\vec{\alpha} = \{1, 2\}$ transforms the RG so that the second dimension is given less weight in the utility metric. This biases the utility metric by altering the distances between nodes and the prediction vector.

favor resolution over spatial position.

In practice, the specific values assigned to the alpha vector have an enormous impact on how adaptation is performed. For this reason, the values must be chosen carefully by application designers. However, proper settings for the alpha vector are necessarily application specific. As a result, targeted user studies may be appropriate to determine the most effective alpha vector settings for a specific application.

4.9 Node States and State Transitions

Throughout this chapter, data structures and metrics have been defined to evaluate the utility and cost of individual nodes of information given specific application needs and system conditions. However, these tools alone are not enough to support the behaviors of interactive adaptive systems.

Adaptive systems must know more than only the relative utility and cost of individual nodes, They must also know if the information corresponding to individual nodes has already been received, is in the process of being received, or is available to be received. The task of maintaining this information is accomplished through the use of node states and state transitions.

This section describes the space of all possible node states. Several invariants regarding the valid state space are also defined. State transitions, used to effect the change of a node's state upon the arrival or disposal of information, are discussed in this section as well. Finally, I define the availability front: the set of nodes from which the adaptation algorithm must choose its next target for retrieval.

4.9.1 Node States

Each node in a RG is assigned to one of four possible states. A node's state may change over time, but it is assigned just a single state at any particular moment in time. A node's state reflects the current status of the information represented by that node. For a node n_i, the state is noted as $\mathbf{State}\{n_i\}$.

For example, some nodes may correspond to information that has already been received by the application and is currently stored in a local cache. Alternatively, other nodes may correspond to information that has not yet been received. These properties of a node are represented by a node's state.

There are four valid node states: idle, available, active, and resolved. The set of possible states is considered ordered to allow relational comparisons between node states in the adaptation algorithm to be presented in Section 4.10. The state order is defined in Equation 4.7.

$$Resolved > Active > Available > Idle \tag{4.7}$$

In the following subsections, I define each of the four node states. For clarity, I present them in decreasing order, beginning with the *Resolved* state.

The Resolved State

The highest ranking state is that of *Resolved*. A resolved node is any node for which the application has received enough data that it can resolve the information associated with the node. For example, in the illustrative computer graphics application of Section 3.2.1, a node is said to be resolved when the geometric data required to render the object associated with a node has been received and cached by the application.

The Active State

A node is in the *Active* state after a request has been issued to initiate the delivery of data associated with the node, but before all of the requested data has been received. For example, in the illustrative application, a node enters the *Active* state when the application issues a request to load the data for a particular geometric object. The node remains in the *Active* state until all of the geometry data has been loaded. At that point, the node would transition to the *Resolved* state. State transitions are covered in more detail in Section 4.9.4.

The Available State

Nodes in the *Available* state are nodes that have not yet been targeted for resolution by the application, but that are possible to resolve without resolving some other node first. Because edges represent a depends-on relationship, **State**$\{n_i\}$ is *Available* if either (1) $n_i \in B$ (i.e. has a self-edge), or (2) at least one of the the edges in the set **Dep**$\{n_i\}$ points to a node (or group of nodes for split-edges) that is either *Active* or *Resolved*.

For example, consider the illustrative computer graphics application which makes use of multi-resolution geometric models of trees. Suppose that at a particular moment in time, the illustrative application has resolved only the node n_{lowRes}, corresponding to the lowest resolution model of a particular tree in the virtual forest. The node corresponding to the medium resolution tree model, n_{medRes}, would be marked as *Available* because it is neither *Resolved* nor *Active*, and it depends upon a resolved node, n_{lowRes}.

The Idle State

The lowest ranking node state is the *Idle* state. Nodes in this state have not yet been targeted by the application for resolution. In addition, *Ilde* nodes are not even available to be targeted because none of the nodes on which they depend are *Resolved* or *Active*, nor do *Idle* nodes have a self-edge which would allow them to be decoded without any prior knowledge. Such nodes are *Idle* because they can not be resolved without resolving some other node first.

Once again, consider the illustrative computer graphics application. As in the *Available* state example, suppose that at a particular moment in time, the illustrative application has resolved only the node n_{lowRes}, corresponding to the lowest resolution model of a particular tree in the virtual forest. It was shown in the last subsection

that the node n_{medRes}, corresponding to the medium resolution tree model, would be marked as *Available*.

In contrast, the node $n_{highRes}$, corresponding to the highest resolution tree model, would be marked as $Idle$ because the node on which it depends, n_{medRes}, is not *Resolved* or *Active*, nor does $n_{highRes}$ contain a self-edge. The $Idle$ state for $n_{highRes}$ indicates that before resolving the node, the application must fist resolve the middle resolution tree model on which the high resolution model depends.

4.9.2 The Availability Front

Every node within a RG is assigned to one of the four states. However, the set of nodes assigned to the *Available* state are of particular importance in our adaptation algorithm. These nodes, which form the *availability front* (noted as A), form the pool of possible nodes from which the adaptive algorithm must choose its next target for resolution.

The availability front has special significance during the utility evaluation stage of the adaptation algorithm. The utility metric need only be computed for each of the nodes within A. The adaptation algorithm then initiates an operation to promote the node within A whose UCR is highest.

4.9.3 Node State Invariants

The overall state of an application can be described in part by the states assigned each of the nodes in the representation graph. Taken as a whole, the state of the RG describes which data is already resolved, actively being resolved, or available to be resolved in the immediate future.

The ability of node states to describe the availability and resolution of data makes them ideal for expressing adaptive data operations. The adaptation algorithm maps both requests to activate individual nodes and the receipt of edge data to changes in node state.

While maintaining the dynamic states in the RG, the adaptation algorithm must adhere to three *node state invariants*. These invariants are defined as part of the adaptation framework and are designed to ensure that the node state space remains consistent throughout the life of an adaptive session.

Each of the three invariant conditions are required to be true at all times. Following any node's change in state, these invariants must be reinforced. Only state transitions

which reestablish these conditions are considered legal in our framework.

In this subsection, I cover each of the three invariants: (1) the Self-Edge Invariant, (2) the Coherent Front Invariant, and (3) the Active Predictor Invariant.

Self-Edge Invariant

The first invariant is the *Self-Edge Invariant*. This rule stipulates that any node with a self-edge may not be in the $Idle$ state. Expressed formally, the invariant states:

$$\mathbf{State}\{n_i\} > Idle, \forall n_i \in B \tag{4.8}$$

This invariant follows directly from the definition of the $Idle$ state. A node is $Idle$ only when a node requires that some predictor be resolved before being activated. Nodes with a self-edge, by definition, do not depend on any predictors, and can therefore be activated at any time. Such nodes may therefore never be assigned to the $Idle$ state.

It is because of this invariant that the set of base nodes ($B \subset S$) was defined in Section 3.2.4. At the very start of a session, when an application has yet to activate or resolve any of the nodes within its RG, all nodes are placed in the $Idle$ state except, because of the Self-Edge Invariant, those nodes in the base set B.

It is the Self-Edge Invariant, therefore, that cleanly initializes the availability front (A) at the start of an adaptive session to include a initial set of nodes in the *Available* state. The initial availability front, known as the set of base nodes (B), forms the original pool from which the adaptive application can begin targeting individual nodes for resolution.

Coherent Front Invariant

The availability front defines the span of nodes within the RG from which the adaptive application must choose a new target for resolution. An important requirement is that the front remain coherent throughout the large number of dynamic state transitions that occur during adaptation.

The *Coherent Front Invariant* stipulates that a node's state must be greater than or equal to the state of all of its dependents. This rule ensures that the algorithm maintains a coherent front through the representation graph based on the data dependence relationships expressed by edges. Expressed formally, the invariant states:

$$\mathbf{State}\{n_i\} \geq \mathbf{State}\{\mathbf{Src}\{e_j\}\}, \forall e_j \in \mathbf{Arr}\{n_i\} \tag{4.9}$$

Active Predictor Invariant

The third and final invariant is the *Active Predictor Invariant*. This rule stipulates that a node can not be idle if it has one or more predictors that are in state *Active* or greater. This again follows directly from the definition of the $Idle$ which says a node n_i can only be idle if all of its predictors are $Idle$. Expressed formally, this invariant states:

$$\textbf{State}\{n_i\} > Idle, \forall n_i | \exists e_i \in \textbf{Dep}\{n_i\} | \textbf{State}\{\textbf{Dest}\{e_i\}\} \geq Active \tag{4.10}$$

The implication of the Active Predictor invariant is that an $Idle$ node *must* transition to *Available* upon the transition of any of its predictors to the *Active* state. The enforcement of this invariant within the RG has the potential to set off a series of secondary transitions as covered in Section 4.9.4.

4.9.4 State Transitions

Data adaptation is performed by transitioning individual nodes between states. As data is requested and received by the adaptive application, nodes are promoted beginning from the $Idle$ state eventually arriving at the *Resolved* state. As data is discarded, nodes are demoted back down toward the $Idle$ state.

This behavior defines two principal categories within which we can classify all legal state transitions: (1) *promotion transitions* and (2) *demotion transitions*. State transitions can be further categorized into one of two subcategories. *Primary transitions* occur as a direct result of an adaptive operation. Following a primary transition, *secondary transitions* may ripple through the representation graph as invariants are reinforced.

In this subsection, I describe the set of valid state transitions, categorized as either promotion or demotion transitions. For each transition, I discuss the conditions under which it may be triggered. For clarity, I introduce the notation $Trans(n_i, StateA, StateB)$ to mark a transition of node n_i from $StateA$ to $StateB$.

Promotion Transitions

At the start of a session, every node in a RG is set to the $Idle$ state except those nodes in B, which are initialized to *Available*. Throughout the life of the session, nodes are

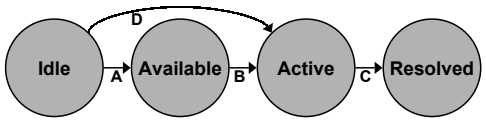

Figure 4.6: Transitions A-D are the four valid promotion transitions. Nodes are typically promoted by one state at a time (A-C). However, nodes can skip from *Idle* to *Active* under certain conditions (D).

targeted for resolution by the adaptation algorithm, triggering state transitions that promote a node's state from *Idle* up through *Resolved*. There are four valid promotion transitions, as shown in Figure 4.6.

Nodes in the *Idle* state can be promoted along one of two paths. First, the transition $Trans(n_i, Idle, Available)$ occurs as a secondary transition when one of n_i's predictors is promoted to *Active*. These secondary transitions do not occur explicitly, but rather are performed implicitly as the active predictor invariant is enforced following the transition of one of n_i's predictors to the *Active* state.

The second valid promotion transition from the *Idle* state is $Trans(n_i, Idle, Active)$. This transition is also a secondary transition and occurs when the adaptation algorithm requests the data assigned to a cluster to which any edge in $\mathbf{Dep}\{n_i\}$ belongs. This transition occurs because the RG data structure considers all edges within a cluster as an atomic unit of information. Therefore, when the adaptation algorithm determines that a specific edge should be received, it must receive data for all edges in the cluster, even edges for which the source node is in the *Idle* state. The activation of a cluster c_j results in the promotion of the source node for every $e_k \in \mathbf{Edges}\{c_j\}$. Figure 4.7 illustrates a scenario in which this transition would take place.

The third promotion transition is $Trans(n_i, Available, Active)$. This is a primary transition and indicates that the adaptation algorithm has targeted an *Available* node for resolution along some edge. The transition occurs when the adaptation algorithm begins receiving data to resolve the information represented by node n_i along one of the edges in $\mathbf{Dep}\{n_i\}$. This transition has the potential to trigger a number of secondary transitions due to the Active Predictor Invariant. This transition can even occur as a secondary transition itself, under under the same circumstances as $Trans(n_i, Idle, Active)$.

The final promotion transition is $Trans(n_i, Active, Resloved)$. This occurs when a node receives enough data to be be resolved and is no longer actively receiving data. It is a primary transition and does not trigger any secondary transitions.

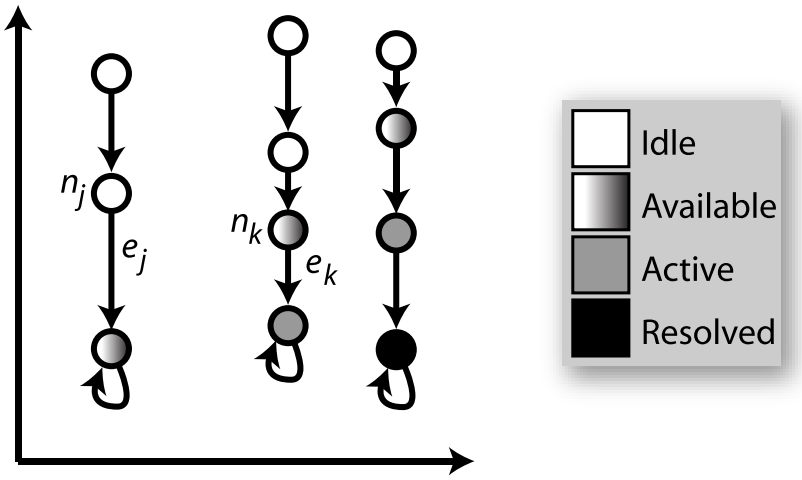

Figure 4.7: Suppose that edges e_j and e_k belong to the same cluster. As the adaptation algorithm targets n_k for resolution, the cluster **Clust**$\{e_k\}$ is activated. As a result, the source node for edge edge in **Clust**$\{e_k\}$ must be promoted to at least the *Active* state. For n_j, this leads directly to $Trans(n_j, Idle, Active)$.

Demotion Transitions

During the life of an adaptive application, the primary goal of adaptation is to obtain the most useful information given changing system and application requirements. The receipt of information is modeled via the promotion transitions described in the section above.

However, because data storage resources are finite, it is not possible to continuously receive additional information without simultaneously disposing of older and less useful data. The adaptation algorithm must therefore support not only promotion transitions, but also *demotion transitions*. For example, demotion transitions are used to reflect the flushing of data from a local cache or the deactivation of clusters which the adaptation algorithm deems no longer useful.

Unlike promotion, which limits the valid paths through which nodes can transition the state space, there are no restrictions for demotion. This is illustrated in Figure 4.8 which shows that all six possible paths are valid.

As the figure indicates, a node n_i in the *Resolved* state can transition to any of the three other states. The specific transition is determined by the state invariants. For example, $Trans(n_i, Resolved, Active)$ is performed if a cluster containing any of the edges in **Dep**$\{n_i\}$ is currently loading. Otherwise, n_i may transition to either $Idle$ or *Available*. The specific transition to occur is determined by the Coherent Front

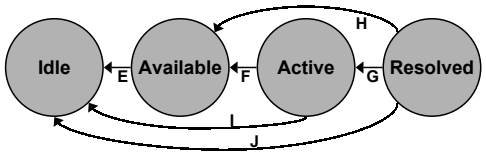

Figure 4.8: Transitions E-J are demotion transitions. Unlike promotion transitions, it is legal to be demoted along any of the six possible transition paths.

Invariant.

The demotion transitions presented so far (i.e. transitions from the *Resolved* state) are primary transitions that occur as a result of flushing a node's data or deactivating a cluster. The remaining demotion transitions are all secondary transitions and occur as side effects as state invariants are reinforced. Following any demotion transition of node n_i, the nodes linked to n_i through edges $\mathbf{Arr}\{n_i\}$ must all be checked to ensure compliance with the state invariants. For highly dependent datasets with a high degree of connectivity between nodes, this has the potential to trigger a wave of state transitions as the invariants are reinforced.

4.9.5 Proof of State Transition Stability

The potential for a chain reaction of state transitions raises an important question surrounding the stability of state transitions: is the process of managing state transition stable, or can a single primary transition trigger a series secondary transitions that never concludes? The proof in this subsection shows that even in the worst case, the sequence of state transitions is finite and the process of reinforcing invariants is guaranteed to terminate.

As outlined in Section 4.9.4, demotion transitions trigger only other demotion transitions. Nodes are strictly ordered, meaning there are at most three demotion transitions a single node can undergo without a promotion. Since the number of nodes in S is finite, the number of possible node transitions is bound by $(3|S|)$. Since (1) a finite bound on the number of demotion transitions exists and (2) demotion transitions can only trigger further demotion transitions, the process must terminate.

The same argument applies to promotion transitions because they can only trigger secondary promotion transitions. Once again, the maximum possible number of promotion transitions is bound by $(3|S|)$, ensuring termination.

4.10 Iterative Adaptation Algorithm

The building blocks for an efficient iterative adaptation algorithm have been presented throughout this chapter. In this section, these pieces are tied together into a common algorithm that performs adaptation as a greedy maximization process. The following paragraphs describe the algorithm depicted in pseudocode in Figure 4.9.

4.10.1 The Active Cluster Set

The adaptation algorithm is written using the notion of an active channel set, noted as *activeChannelSet* in the pseudocode. The active channel set corresponds to the set of clusters in the RG that are actively being received by the application at a particular point in time. The size of the active cluster set can change over time based on both system conditions and the nature of the application itself. The size of the set can vary from zero to *activeClusterLimit* inclusive, where the latter is an application-specific parameter. This flexibility allows the same adaptation algorithm to be used across a wide variety of adaptive applications.

For example, some applications may require the serialized arrival of data. In these applications, all the data from a single cluster must be received before moving on to the next most useful cluster. In this case, *activeClusterLimit* would be set to one, and the application would be restricted to a active cluster set size of either zero or one.

In other applications, such as adaptive streaming applications, it may be desirable to allow the activation of multiple clusters in parallel. In this case, the *activeClusterLimit* would be set to a larger value, more than one cluster to be active at the same time. The exact value of the *activeClusterLimit* could change dynamically as system conditions change. For example, a distributed application may choose to adjust the limit in response to changing network conditions.

As shown in Figure 4.9, the *activeClusterSet* is initialized to the empty set (line 1) before entering the main adaptive loop (line 2). The main adaptive loop has three clauses, all of which operate by updating the membership of the *activeClusterSet*.

4.10.2 The Adaptive Control Loop

The iterative adaptation algorithm repeats a common control loop throughout the life of an adaptive application. The body of this loop contains two sections. The first section handles cluster activation and contains three clauses: (1) the reduction clause,

(2) the growth clause, and (3) stability clause. All three clauses are similar in that they enable a change in the contents of the *activeClusterSet*. However, they differ in how they handle the *activeClusterLimit*. The second section handles data arrival.

The Reduction Clause

The first clause in the cluster activation section of the algorithm is the reduction clause. This clause, shown in lines 4-7 of Figure 4.9, is executed when the size of the active cluster set needs to be reduced. A decrease in the *activeClusterLimit* corresponds to a reduction in the number of active clusters. This clause will typically take place when the available resources are insufficient to maintain the current number of active clusters. For example, a network streaming application could reduce the number of active clusters to reduce demand for bandwidth in the face of growing network loss.

When the system reports that a reduction in the *activeClusterLimit* is required, the adaptation algorithm responds by first determining the least useful active cluster (Figure 4.9, Line 5). The algorithm uses the UCR, the RG, the alpha vector, and the prediction vector to determine which of the clusters currently within the *activeClusterSet* is least useful at the current point in time.

The least useful cluster, c_active, is then deactivated (i.e. the request for data is canceled) and the cluster is removed from the *activeClusterList* (Figure 4.9, Lines 6-7). The deactivation is mirrored via node state transitions that can affect every node with an edge in c_{active}. These transitions are demotion transitions that move nodes from the *Active* state down to either *Available* or $Idle$.

The Growth Clause

The second clause in the cluster activation section of the algorithm is the growth clause. This clause, shown in lines 8-11 of Figure 4.9, is executed when the size of the active cluster set needs to be increased, allowing an additional cluster to be activated together with the already active clusters in the *activeChannelSet*. The *activeClusterLimit* increases when system resource are plentiful enough that an additional cluster can be handled by the system. For example, a network streaming application could increase the *activeClusterLimit* if it detects that additional communication resources have been made available.

When the system reports that the *activeClusterLimit* can be increased, the adaptation algorithm responds by finding the most useful inactive cluster, $c_{inactive}$, requested

that it be activated, and adding it to the active cluster list. Once again, the UCR, RG, alpha and prediction vectors are used to determine which cluster is most useful.

The activation of $c_{inactive}$ initiates state transitions for nearly every source node of every edge in $c_{inactive}$. Each of these nodes in either the *Idle* or *Available* states is promoted to the *Active* state using the appropriate transition. In addition, secondary $Trans(n_j, Idle, Available)$ transitions can occur due to the Active Predictor invariant.

The Stability Clause

The final clause in the cluster activation section of the algorithm handles the case where the size of the active channel set remains constant. This case, called the stability clause, is shown in lines 12-20 of Figure 4.9. In this event, the algorithm determines both the most useful inactive cluster ($c_{inactive}$) and the least useful active cluster (c_{active}). When $c_{inactive}$ is found to be more useful than the currently active c_{active}, the algorithm swaps membership in the *activeChannelSet* to ensure that at any point in time, only the most useful channels are included in the *activeChannelSet*.

As with both the reduction and growth clauses, any changes in the *activeChannelSet* during the stability clause are mirrored by node state transitions that allow the adaptation algorithm to determine which portions of the representation graph are currently *Idle*, *Available*, *Active*, or *Resolved*.

Data Arrival

Following the management of cluster activation, the adaptation algorithm must handle the data received from the currently activated clusters. This section of the adaptation algorithm is shown in lines 23-29 of Figure 4.9. Every unit of data received belongs to an individual edge e_i in the RG. This data can then be used to decode the information represented by node $n_i = \mathbf{Dep}\{e_i\}$. Once decoded, node n_i undergoes the ultimate promotion transition: $Trans(n_i, Active, Resolved)$.

Once node n_i is resolved, the decoded information is added to a cache made available to the adaptive application. However, because storage is finite, the algorithm must ensure that room remains in the cache for the newly resolved node's information (Figure 4.9, Line 25).

If there is room in the cache, the information for node n_i is simply added to the cache and the algorithm moves on to the next piece of received data. If the cache is full, the least useful cached node, n_{evict}, is evicted to make room for the new information.

Upon eviction, n_{evict} undergoes a demotion transition from the *Resolved* state. The exact transition depends on system conditions and n_{evict} could end up in any one of either the *Idle*, *Available*, or *Active* states.

4.11 Summary

In this chapter, I described my solution for multidimensional data adaptation. Adaptation, which controls the flow of information from a data source to receiver, is a critical component in any non-linear media streaming system. Data adaptation must adapt the stream of data to reflect both the receiver's application-level requirements and the limited communication resources available for transmitting the data.

This chapter describes a general framework for expressing multidimensional adaptation. The framework regards adaptation as a maximization problem in which the goal is to maximize the utility of the received data while simultaneously minimizing the access cost.

The primary inputs to the adaptation algorithm are data structures that allow for the expression of both data availability and application-level requirements. These include the RG, the prediction vector, and the alpha vector. The RG describes the overall structure of the dataset and defines the dimensions along which the application can adapt. The prediction vector describes both current and possible future application-level requirements. The alpha vector is used by the adaptation algorithm to manage inter-dimensional trade-offs.

These data structures are then used to define the UCR metric, a ratio of the utility of information and the cost of data access. The UCR is a function built on top of application specific cost and utility metrics. This layer of abstraction allows for the definition of a general adaptation algorithm that can be customized for specific applications by implementing specialized cost and utility metrics.

As data is retrieved by the adaptation algorithm or flushed from an application's cache, it is critical that the current state of individual elements of information be maintained. This is accomplished in the adaptation framework by augmenting nodes in the RG structure with a state field. Throughout the life of an adaptive application, the states of individual nodes go through state transitions. These transitions can be either promotion transitions, which occurring during the resolution of a node, or demotion transitions, which occur when a node is flushed from an application's cache. In order to maintain an a coherent node state within a RG, several node state invariants were

presented.

Finally, this chapter provided a detailed algorithm for data adaptation. The algorithm employs the concepts presented throughout this chapter to efficiently and flexibly drive adaptation. The algorithm allows for the adaptive behavior to be carefully controlled by current system and application requirements by incorporating these constraints into the UCR evaluation.

```
1   activeClusterSet = {}
2   Repeat forever:
3       // Manage Cluster Activation
4       If (conditions warrant decrease in activeClusterLimit)
5           c_active = GetLeastUsefulActiveCluster(RG, activeClusterSet)
6           CancelRequestFor(c_active)
7           Remove c_active from the activeClusterSet
8       Else if (conditions warrant increase in activeClusterLimit)
9           c_inactive = GetMostUsefulInactiveCluster(RG, activeClusterSet)
10          RequestDataFor(c_inactive)
11          Add c_active to the activeClusterSet
12      Else
13          c_inactive = GetMostUsefulInactiveCluster(RG, activeClusterSet)
14          c_active = GetLeastUsefulActiveCluster(RG, activeClusterSet)
15          If UCR(BestNode(c_inactive)) > UCR(BestNode(c_active))
16              CancelRequestFor(c_active)
17              Remove c_active from the activeClusterSet
18              RequestDataFor(c_inactive)
19              Add c_active to the activeClusterSet
20          End-if
21      End-if
22      // Manage Data Arrival
23      For every Data{e_i} received
24          Use Data{e_i} to decode node n_i = Dep{e_i}
25          If cache is full
26              Make room in cache by evicting cached node with lowest UCR
27          End-if
28          Store decoded n_i in cache
29      End-for
30  End-repeat
```

Figure 4.9: Pseudocde for the iterative adaptation algorithm. The algorithm attempts to maximized the UCR via an iterative greedy algorithm.

Chapter 5

The Generic Adaptation Library

In this chapter, I describe GAL, a library for multidimensional and multimedia adaptation (Gotz and Mayer-Patel, 2005b). GAL is a realization of the the conceptual framework presented in Chapter 4. Implemented in C++, GAL is a middleware library for adaptive non-linear media applications that provides a fully functional implementation of the iterative multidimensional adaptation algorithm and provides an Application Programming Interface (API) through which applications can express data requirements and adaptation preferences.

This chapter begins with a review of several underlying assumptions in the GAL design. Second, a detailed discussion of the layered system design of GAL is presented. Third, this chapter outlines the layer-to-layer interfaces which define the API made available to applications employing the library. Finally, the plug-in architecture for cost and utility metrics is described.

Throughout this chapter, I provide a detailed overview of a practical implementation of the conceptual data representation and adaptation framework presented in earlier chapters. The GAL implementation is then used as the basis for the experimental prototype and evaluations presented in the remainder of this dissertation.

5.1 Assumptions

GAL is designed to support a broad class of adaptive systems. As such, it is applicable to a wide range of application models and configurations. However, our design makes a few assumptions which are critical to the design of the library. In particular, GAL assumes a particular network model as described in this section.

The network model for which GAL is designed is a subscription based service where

individual clients can access data from a server through subscription operations. A client, therefore, can employ two functions to support data access. Both of these functions are performed on a specific communication channel, noted as c_i. The two functions are:

- **Subscribe:** A request to receive data is performed via a *subscribe* operation. Noted as **Sub**$\{c_i\}$, a subscribe operation initiates a flow of data to the receiver.
- **Unsubscribe:** A request to stop receiving data is performed via a *unsubscribe* operation. Noted as **Unsub**$\{c_i\}$, an unsubscribe operation halts the flow of data to the receiver.

In addition, GAL assumes only best-effort delivery of data packets. Therefore, out-of-order delivery and lost packets can be handled within the library. Any subscription-based network model can be employed. This includes both IP Multicast and any of several ALM variants. In addition, the layered library design of GAL allows for the design of application-layer subscription models to be implemented over many additional network models, including both TCP/IP and UDP/IP. As a result, GAL can be widely deployed across most of today's networks.

5.2 Layered System Design

The GAL library implements the multidimensional adaptation framework from Chapter 4 using a multi-layered system design. The overall philosophy in GAL's design places all adaptation operations on the client, which in turn drives requests for additional data to be delivered by the server. Therefore, the GAL library sits entirely on the client side. This design decision places all per-client work on individual clients and alleviates the need for a server to maintain state information for each of its clients.

Given this system architecture, GAL introduces an extra system layer between the application and communication layers. This middle layer performs adaptation with input from neighboring layers in order to incorporate both application and network measurements into the adaptive algorithm.

GAL's design consists of three primary layers: (1) the application layer, (2) the adaptation layer, and (3) the communication layer. A single client application is composed of the union of these three layers. The top layer consists of the adaptive application that requires GAL's services. The middle layer is the GAL middleware library

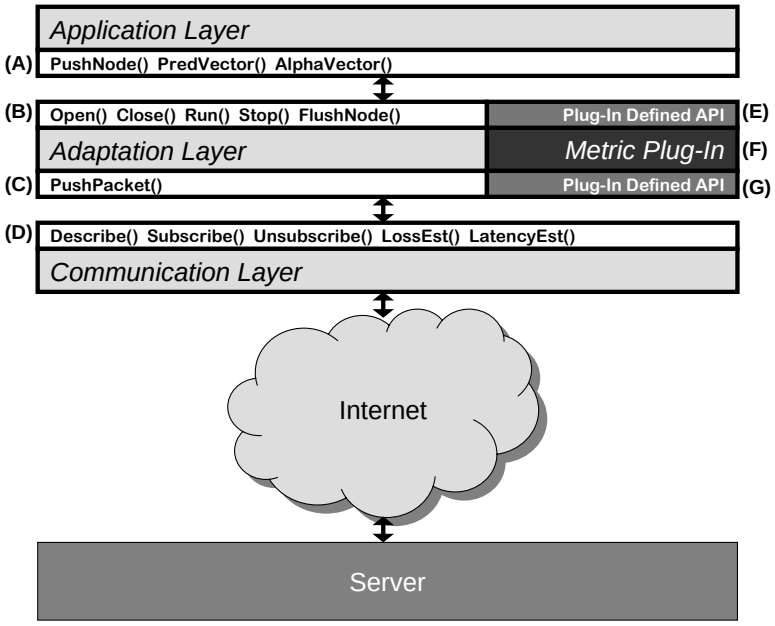

Figure 5.1: The system model consists of three layers: the application layer, adaptation layer, and communication layer. The Generic Adaptation Library, or GAL, is an implementation of the adaptation layer. GAL defines a number of application programming interfaces, labeled with letters A-G in the figure. Custom cost and utility metrics are supported as plug-ins.

itself. The bottom communication layer is a protocol-specific wrapper around the core networking functionality. This wrapper is responsible for implementing the exact subscription API required by GAL on top of the native network services. The layered design is illustrated in Figure 5.1.

5.3 Layer-to-Layer Interfaces

The GAL library provides two primary application programming interfaces (APIs). One API defines the application-to-adaptor interface. The second API defines the communicator-to-adaptor interface. In addition, GAL provides templates for both the Application and Communication layers that define a set of minimal functionality that must be supported by each layer.

5.3.1 Application-Adaptor API

The application-to-adaptor interface defines a set of functions used by the application to interact with the adaptor library. A simplified API is illustrated in Figure 5.1(B). An

application developer will use this API to interact with GAL and integrate the library into the overall application..

The API includes several functions that allow the application to control the adaptation layer. These include functions that control the status of the thread of execution responsible for adaptation, as well as a function for notifying the adaptation layer that data has been discarded from the application's memory space.

The first two functions provided by the Application-Adaptor API are **Open**{} and **Close**{}. These functions initiate and terminate a new session, respectively. As an application starts up, it calls the **Open**{} function to open a new session by initializing the adaptation layer and telling it which data source the application wishes to access adaptively. During initialization, the RG data structures are created for the specific data set being opened and the entire node state space is properly initialized.

As it shuts down, an application calls the **Close**{} function to close the application's adaptation session. This function tells the adaptation layer that the application no longer requires its services and allows the adaptation layer to free its allocated resources (such as the RG data structures) and tear down any open network connections.

After a session has been opened, the application must call the **Run**{} function. This function tells the adaptation layer to enter the main iterative adaptation loop. Following a call to **Run**{}, the adaptation layer iterates indefinitely, continually evaluating UCR value for available nodes and deciding which data should be requested for resolution.

The adaptation layer continues iterating until the application calls the **Stop**{} function. This function causes the adaptation layer to leave the iterative loop and cease making new requests for data. However, the RG data structures remain intact and the current node states are preserved. This allows the application to restart the iterative adaptation algorithm from where it left off. The state information is not destroyed until the application calls the **Close**{} function.

The final function in this portion of the API is **FlushNode**{}. This function is used to notify GAL that a specific node's data has been flushed from the application. While the adaptation thread is executing, data is constantly arriving from the communication layer. After the incoming data is processed and recorded within GAL, it is passed up to the application via the **PushNode**{} function. At that point in time, GAL marks the state of the node as *Resolved*.

However, the data does not remain with the application indefinitely. Data may be discarded by an application for one of two reasons. First, the data may no longer be

needed. This could occur because an application decides that the data is no longer relevant or perhaps the data has become invalid due to changing system conditions. Second, there may be limited storage on the client. In this case, the client will maintain some sort of local cache and due to the limited size, will have to evict old data to make room for data that has newly arrived.

In either case, the application must notify GAL that the information for a node is about to be removed from memory. This is done via the **FlushNode**{} function. Once notified, the GAL library initiates the proper demotion transitions within the RG representation to reflect application's action.

Together, these five functions provide the bulk of the functionality required for an application to employ GAL for adaptive data access. Along with the functions required by the application layer template (see Section 5.3.3), an application can create a new session, start the adaptation algorithm, express changes in preferences and requirements, handle the arrival and disposal of data, and terminate the adaptation algorithm and application. The pseudocode in Figure 5.2 shows how a typical GAL-based application would employ these functions.

5.3.2 Communicator-Adaptor API

The communicator-to-adaptor interface defines a set of generic functions used by the adaptor to interact with the adaptor library. A simplified portion of the API is illustrated in Figure 5.1(C). This portion of the API is particularly simple because the services required are trivial. In response to subscription operations, the communication layer must simply deliver packets to the adaptation layer.

Therefore, the API specifies only one key function which handles the arrival of new data packets. The **PushPacket**{} function is used as a callback function by the communicator every time a new packet arrives from the network. The communicator can call this function for packets in any order, even in the presence of lost packets. This flexibility relieves the communication layer from any responsibility for packet reordering or reliable delivery. As a result, the communication layer can be built on top of any best-effort protocol with or without reliable delivery, including both TCP and UDP.

5.3.3 Application Layer Template

The application layer is an abstraction for the application-specific code required by any actual system. As such, this layer will vary greatly for each application. However,

an application must define a minimal set of functionality to work correctly with GAL. This interface, illustrated in Figure 5.1(A), is used by GAL to exchange data and to monitor changing application conditions.

The first function in the application layer template is **PushNode**{}. This function handles data exchange between the adaptation and application layers. As a node enters the resolved state within the adaptation layer, the data corresponding to the node is pushed to the application layer via the **PushNode**{} function. An application's implementation of this function determines what happens to data once it becomes available to the application.

The remaining functions in the application layer template are used to provide the adaptation layer with access to the current application conditions. The application is responsible for maintaining both a prediction vector (see Section 4.5.2 and alpha vector 4.5.2 and updating the vector values based on their own requirements and preferences. These vectors are then exposed to the adaptation layer for incorporation into the UCR calculation as part of the iterative adaptation algorithm.

The vectors are shared between layers via two functions. The **PredVector**{} function returns a reference to the current prediction vector, while the **AlphaVector**{} returns a reference to the current alpha vector. Note that both of these functions pass references to, rather than copies of, the vectors. This is important because both the prediction vector and alpha vector are highly dynamic. Using references allows any changes made by the application to have an immediate impact on the UCR calculation without requiring extensive copying of data.

Taken together with the Application-Adaptor API, the application layer template provides all the functionality required for an adaptive application to make use of GAL for multidimensional adaptation. The pseudocode for a typical adaptive application is shown in Figure 5.2.

5.3.4 Communication Layer Template

As with the application layer, the GAL library provides a template for the communication layer. The template defines the minimum functionality required by GAL and specifies the interface through which GAL interacts with the underlying network service.

The communication layer is an abstraction for the portion of the system that handles the low level network functionality. This abstract layer is designed to allow GAL to

```
1   Application Initialization
2   Call Open{} to create a new adaptive session
3   Additional application initialization
4   Call Run{} to initiate the iterative adaptive algorithm
5   While (Application is running) {
6       Update prediction vector if needed
6       Update the alpha vector if needed
6       Process new data arriving via PushNode{}
7       Signal data disposal by calling FlushNode{}
8       Perform other application tasks
9   }
10  Call Stop{} to halt the iterative adaptive algorithm
11  Application Tear Down
12  Call Close{} to terminate the adaptive session
13  Additional Application Tear Down
```

Figure 5.2: Pseudocde for an application built on top of the GAL library.

work equally well with a large set of transport level network protocols. GAL depends on at least best-effort packet delivery and uses a subscription metaphor for data access. Based on the template's set of functions, a new communicator can be defined for each new network protocol to be used by an application.

The first function defined in communication layer template is the **Describe**{} function. This function must be defined to utilize the available network resources to obtain a *description* of the utility space and representation graph from a specified server. The description is a formal XML-based file format used to enumerate the RG's properties, including the nodes, edges, and utility space specifications.

An application's initial call to GAL's **Open**{} function includes information about the server and dataset that the application wishes to access. In response to the **Open**{} call, the GAL library calls the **Describe**{} function of the communication layer to obtain the formal description of the RG being accessed. This information is then used to create the appropriate data structures within the adaptation layer.

Two other functions included in the communication layer template specification are **Subscribe**{} and **Unsubscribe**{}. These functions are used to provide the required subscription-oriented interface to GAL. The underlaying implementation of these two functions depends on the natively supported network protocol. A communicator therefore is responsible for bridging the gap between the services provided by the network (i.e. as socket-based TCP or best-effort UDP) and the subscription model used by

GAL.

When the adaptation layer chooses to activate a specific cluster (based on the UCR calculation as part of the adaptation algorithm), it uses the **Subscribe**{} function to initiate the data request. A corresponding call to **Unsubscribe**{} terminates the request. In a typical TCP-based communicator, these functions could correspond to the setup and tear down of a network socket connection.

Finally, there are two network measurement functions required by GAL. These functions are used as input to the the adaptation algorithm so that the adaptation layer can effectively monitor network performance and adapt accordingly. This feature of the communication layer template is critical for implementations based on network protocols without congestion control (i.e. UDP). In these cases, the adaptation algorithm can vary the number of simultaneously active clusters to maintain satisfactory network performance levels.

The first of the two performance monitoring functions is **LossEst**{}. This function provides a loss rate estimate for the underlying network. Using loss as an implicit signal of congestion, the adaptation algorithm is designed to reduce its number of concurrent data requests when loss levels rise.

The second performance monitoring function is **LatencyEst**{}. This function provides an estimate of the leave latency, defined as the time between the initiation of a unsubscribe request and the actual cessation of data flowing over the network. Given that the adaptation algorithm can attempt to manage congestion via changes in the number of concurrent data requests, it must have an idea of how quickly a change in subscription level will impact the loss rate. The latency estimation function provides that metric.

5.3.5 The Metric Plug-ins

The adaptation algorithm defined in Chapter 4 relies on abstract cost and utility metrics in its definition of the UCR. These metrics are critical to GAL because the adaptation algorithm performs adaptation as a maximization problem where the UCR is the value to be maximized. However, the adaptation framework is careful to leave the actual implementation of the cost and utility metrics undefined because individual applications will need to measure the cost of data access and the utility of information in application-specific ways.

The GAL library follows the same philosophy by providing a plug-in architecture

through which applications can either reuse existing metric plug-ins, or develop their own based on their individual requirements.

New metric plug-ins can be defined to implement any desired utility or cost metric. The plug-in is given access to the internals of the adaptation layer, including the representation graph. Therefore, a particular plugin can make full use of the data description, utility space definition, and the entire node state space in determining cost or utility. GAL's plug-in architecture is illustrated in Figure 5.1(F).

Standard inputs to the cost and utility metrics, such as the prediction and alpha vectors or network measurements, can be accessed through the previously presented APIs, similar to the adaptation layer itself. However, some metrics may need additional information from either the application or communication layers. A custom metric plug-in may therefore define its own API to be exposed to both the application and communication layers. This is shown in Figures 5.1(E) and 5.1(G).

5.4 Summary

This chapter described GAL, a library for multidimensional and multimedia adaptation. GAL is an actual implementation of the adaptation framework and data representation presented in earlier chapters of this dissertation. In addition, GAL forms the foundation of the experimental prototype used throughout the evaluations presented later in this dissertation.

This chapter reviewed several underlying assumptions in the GAL design, including the subscription-based network model to which it is designed. In addition, this chapter detailed layered system design of GAL, including the application layer, the adaptation layer, and the communication layer.

In the middle, there is an adaptation layer that contains the actual GAL implementation. The adaptation layer is responsible for maintaining the RG data structure, keeping track of node state changes, and performing the iterative adaptation algorithm.

The topmost layer is the application layer. This layer is an abstraction that represents an adaptive application that employs the GAL library. An application layer template is defined that specifies the interface thorough which GAL interacts with the application. Similarly, an API is defined as part of the adaptation layer which specifies the interface through which the application interacts with GAL.

The bottommost layer is the communication layer. This layer is a wrapper around the underlying network service that provides a subscription-oriented interface to GAL.

The GAL library interacts with the communication layer through the functions defined in the communication layer template. Similarly, the communication layer interacts with GAL through an API defined as part of the adaptation layer. GAL can be ported to a new underlying network protocol simply by implementing a new communication layer that maps the underlying network services to GAL's subscription-based model.

Chapter 6

Experimental Prototype and Evaluation

In this chapter, I describe the experimental prototype application built to evaluate the design and performance of GAL, as well as the underlying RG representation and multidimensional adaptation framework. The prototype application is an stream-based image-based rendering application where a server distributes a large set of high resolution images to a heterogeneous set of remote clients, each of which requires a unique set of images.

This chapter begins with a detailed description of the prototype application. In addition to coverage of the application domain itself, this chapter includes a discussion of both the streaming challenges faced in designing the prototype application and how these challenges are representative of a broad class of non-linear media streaming applications.

Second, a review of the experimental testbed and methodology is presented. This review includes a formal definition of The Summed Utility Metric (SUM), the performance metric used in many of the evaluation experiments.

Finally, the results from a number of experiments are presented that evaluate the prototype's performance using TCP data delivery. As part of this presentation, additional challenges related to large scale performance are highlighted. These challenges will serve as motivation for the solutions presented in the remainder of this dissertation.

6.1 Prototype Application

The experimental prototype application used throughout the evaluation presented in this dissertation is an image-based rendering streaming application, known as Streaming Walk-throughs of Image-based Models (SWIM). This target application is a client-server version of the Sea of Images (SOI) algorithm (Aliaga et al., 2002) designed to transmit digitized spaces to large groups of users. For example, consider a digital museum that digitizes a famous location and wants to share the image-based model with its virtual visitors.

This SOI-based application is a good match for the adaptation framework presented earlier in this dissertation because it is a high-bandwidth application with several dimensions of adaptability and a dynamic point of interest.

This section begins with an overview of the motivating application as well as the original SOI algorithm. It then presents two designs for a streamed version of SOI capable of serving several clients from a centrally stored image database. The first design is the naive approach using existing tools that has two critical performance flaws. The second design is the approach taken by the prototype application and utilizes the GAL library as part of a more comprehensive solution.

6.1.1 Motivating Example

As a motivating example, imagine a digital museum sharing a large IBR model of a famous location for virtual visitors connected to the Internet (Gotz and Mayer-Patel, 2005a). In such a scenario, the museum curators would first need to digitize the space using IBR techniques. Then, they would place the digitized data on-line and make it available to legions of on-line visitors to navigate interactively and independently.

To support this vision, the prototype must extend an IBR algorithm to function in a distributed environment where there is a centrally stored data repository with a large set of independently operating clients that have unique operating requirements as they explore the digitized space.

6.1.2 Sea of Images

The prototype system is based on the SOI algorithm (Aliaga et al., 2002) by Aliaga et al. The SOI algorithm is an image-based rendering algorithm that allows for a user to virtually navigate through a digitized scene. The algorithm takes as input a large set

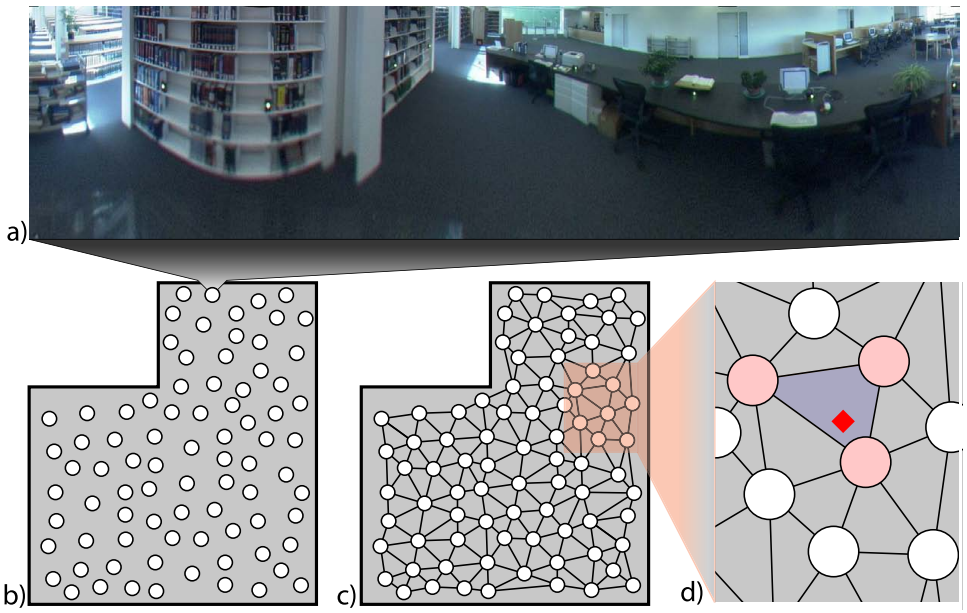

Figure 6.1: The Sea of Images rendering algorithm proposed by Aliaga et al. takes as input a large set of panoramic images. Using the database of images, novel views of the scene can be constructed by interpolating between triplets of stored images. (A) Each panoramic image in the input dataset contains a full 360° view of the scene. (B) The input images are captured by a camera positioned along a plane at eye-level. (C) At runtime, a Delaunay Triangulation of the stored image positions is maintained. (D) As the user moves through the scene, illustrated by the red diamond, the triangle containing the user's position determines the appropriate triplet for interpolation.

of high resolution cylindrical images captured at various points along a eye-level plane. This can be done, for example, with a camera attached to a motorized cart.

A Delaunay Triangulation of the image positions is computed to build a triangle mesh. At run time, a user can navigate a virtual viewpoint along the eye-level plane. A virtual rendering of the scene is synthesized by interpolating between the three nearest pictures found within the input dataset, as determined by the triangle containing the viewpoint. The process is illustrated in Figure 6.1.

The original SOI algorithm is designed to run locally with both the image database and the rendering engine located on the same machine. Even in this configuration, the frame-rate at which reconstruction takes place can have trouble maintaining interactive rates. In addition, the input dataset is often too large to be loaded entirely into memory. As a result, the three closest images may not always be available. Therefore, the triangular mesh is calculated dynamically using only the set of loaded images. The

mesh is maintained throughout the life of the application as images are loaded into or removed from memory.

6.1.3 Streaming SOI

The motivating application requires not only IBR reconstruction, but that large groups of remote clients have interactive access to a centrally stored model through which they can navigate. There are two possible approaches to this streaming problem.

A Streaming Approach Using Existing Technologies

The experimental prototype is a SOI-based IBR streaming application. The simplest approach to supporting such a system would be to perform the entire SOI algorithm on a server and stream the reconstructed imagery to individual clients. In this design, a client application would translate user interaction events into movements of the virtual viewpoint within the scene before reporting the change in viewpoint to the server. The server would then determine the three closest images, reconstruct the new view of the scene, and transmit the images back to the client.

In this design, the server is tasked with performing the entire SOI algorithm. The reconstructed views of the scene produced by the server could then be streamed to the client using existing video adaptive streaming technologies. This in effect would translate the inherently non-linear problem of streaming the actual images to a traditional linear challenge of streaming video.

However, there are two critical flaws in this design which would severely limit the system's interactive experience. The first problem is latency. The delay between a user's interaction and the associated change in generated imagery has been shown to be a critical factor in achieving a sense of presence within virtual environments (Meehan et al., 2003). While no study has been undertaken to study the impact of latency specifically for streaming IBR models, similar results can be expected.

The latency introduced by the design presented here can be quite severe. From the moment of interaction, when a user first initiates a change in viewpoint, to the time where the change becomes evident to the user is directly related to the network's round-trip time between client and server. As shown in Figure 6.2, a change in viewpoint must first be sent from the client to the server. This is followed by a delay while the server reconstructs an updated image. Finally, the resulting image is sent back to the client. Even in the ideal case, where reconstruction can occur instantly and no buffering is

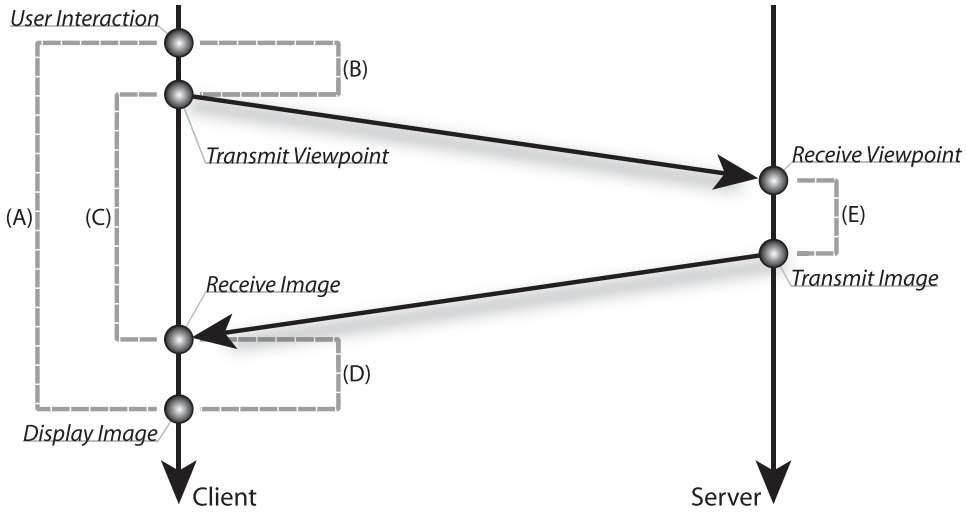

Figure 6.2: A naive approach to streaming a streaming IBR application based on the SOI algorithm would perform reconstruction on a central server and transmit the novel imagery as a video stream. However, this technique performs poorly due to (A) the latency introduced between user interaction and changes in the imagery. Client-side activities introduce (B) latency in translating user interaction to viewpoint movement and (D) latency from buffering the video stream. Server-side activities introduce (E) latency due to reconstruction. Even if these system latencies were performed instantaniously, (C) a latency of one round trip time would be incurred.

required to smoothly stream the resulting video, a full round-trip time is the smallest latency possible.

The second critical flaw in this design is the inability to support a large number of simultaneous users. As described in Section 6.1.2, the SOI algorithm a significant amount of computation. Combining the work required to support several simultaneous clients navigating independently would reduce performance below acceptable levels.

The SWIM Approach

An alternative approach can alleviate both the scaling and latency problems encountered in the previous solution. This approach, rather than leverage video streaming technologies for data transport, attempts to transmit the raw image data to individual clients which are each responsible for their own reconstruction. The approach, named SWIM for Streaming Walk-throughs of Image-based Models, removes the CPU bottleneck away from the server by placing the onus for all per-client reconstruction tasks on the clients themselves.

Moving reconstruction to the client allows for the asynchronous retrieval of image data from the server which removes the round-trip-time latency from between the time of interaction and the time of rendering. Instead, the rendering loop continuously renders novel views of the digitized scene with whatever data is present. In parallel, data is being adaptively requested to meet the client's current requirements.

It is this approach which is used in the experimental prototype referenced in the remainder of this dissertation. The adaptive data access to the raw images is supported by the GAL library. In this approach, the prototype is designed as a client-server system with a single image server that can transmit streams of images to any interested clients. Each client is able to navigate through the space independently along their own unique paths. Each client expresses their individual needs for data using the GAL library's prediction and alpha vectors.

Simple Server Design Philosophy

The SWIM design choice of moving the task of reconstruction from the server to the client is the first example in this dissertation of the *Simple Server Design Philosophy*. This philosophy holds that in order to be truly scalable, a server should not perform any per-client work. In non-linear media applications, where every client requires custom data flows, the work load associated with performing any per-client task will inherently grow along with the size of the user population. By removing all per-client tasks from a server, a design becomes inherently scalable where no additional work is required as new users connect to the service.

The SWIM approach presented so far improves dramatically the server's performance for large user groups. However, as will be shown in Section 6.3.6, there are still bottlenecks that result from per-client server-side work. In the next chapter, a new technique will be presented that takes the simple server design philosophy to another level, leading to truly group-size-independent performance.

6.1.4 Data Representation

Because the prototype employs the GAL library, it is important to specify a mapping between the data set's underlying representation and the RG representation abstraction. It is therefore critical to understand the data representation used to represent the input data set for the prototype.

The input data set to the SWIM application consists of a set of cylindrical panoramic

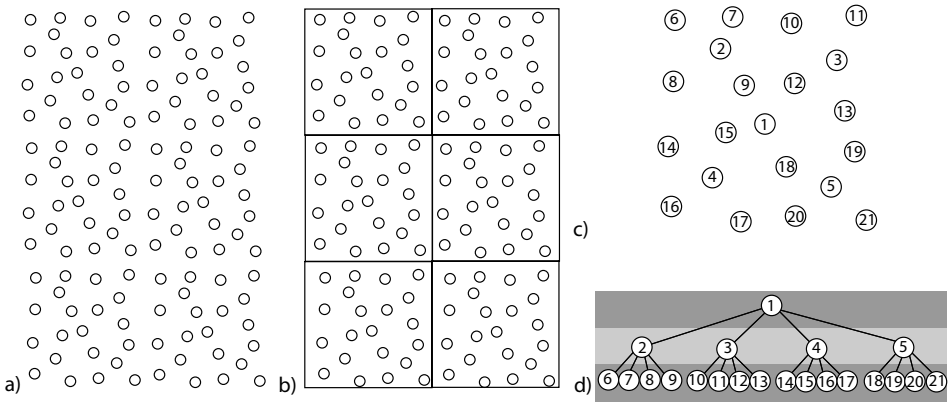

Figure 6.3: The dataset is organized across multiple dimensions. (a) The data consists of a collection of images arranged in a plane. (b) The first step in building the representation is to partition the data into spatial regions. This figure shows a regular grid partitioning. (c) The next step is to order the individual images by spatial density. (d) The ordering is determined by a breadth-first traversal of a quad-tree built using the image positions in the plane.

images. Each image in the input set has an associated position along a 2D plane at eye level. This position corresponds to the location of the camera when the image was captured. The images are encoded using a configurable multi-stage process. The two major stages are segmentation and multi-resolution encoding. Both stage of the process are configurable to allow a high degree of control over the final encoding.

Segmentation

The first step is in the encoding process is *segmentation*. This process splits the input dataset into coherent units of data along four dimensions. These dimensions correspond to four of the five dimensions of adaptive data access provided by the representation. These are the spatial position of the images (2D), the view angle of the images, and the spatial density of the images. The fifth adaptive dimension is provided by the next state of the encoding process.

The data is first partitioned based on the $2D$ planar position of each image. Conceptually, this partitioning structure is designed to allow a client to navigate the database by jumping from one partition to the next as the user's viewpoint changes. Typically, a client will be interested in a neighborhood of partitions surrounding the viewpoint.

The encoding process for the prototype uses a regular grid partitioning algorithm as illustrated in Figure 6.3(b). However, variable resolution partitioning mechanisms

can be used as well. Partitions are used to group images together for predictive coding. Images placed in the same partitions can be used to improve the representation's storage efficiency with the expense of greater inter-image dependence. The exact size of the partitions is configurable and can be used to control the granularity which which the data is encoded.

After partitioning, the segmentation process continues by splitting the dataset along view direction. Each image is captured as a 360° panorama. The input images are broken up into n pieces pieces, each with a $\frac{360°}{n}$ field-of-view. Once again, the value of n is a configuration parameter. At this stage, each segment of the dataset contains only potions of images oriented in the same view direction and captured from the same general location.

Segments are further broken apart along the dimension of spatial density. This step first requires that images be arranged by order of importance as measured by their spatial proximity. This is done by building a progressive ordering of images via quad-tree decomposition, storing the image closest to the center of each region in the corresponding node of the quad-tree. A breadth-first traversal of the quad-tree determines the order. Figures 6.3(c) and 6.3(d) show this process.

The ordering creates a progressive data representation, reflecting the fact that the SOI algorithm can make use of sparsely located images to produce poor quality reconstructions while more dense data continues to arrive. The image groups are further segmented by grouping the images along the progressive ordering. This creates segments of the database at various spatial resolutions.

Every step within the four dimensional segmentation process is configurable to control the size of the segments. In the extremes, a very large segmentation size can group the entire set of images together into a single segment. Conversely, an extremely small segmentation size will place every image into its unique segment. This feature will become particularly relevant in Chapter 7.

Multi-resolution Encoding

Following segmentation, each individual image is encoded at multiple resolutions. The provides the fifth and final dimension of adaptivity available in the prototype's data representation. The encoding process uses the JPEG2000 (ISO/IEC, 2000) image compression standard which uses a wavelet-based encoding process. Wavelet-based coders inherently encode an image at multiple resolutions. The progressive JPEG2000 code stream is split into multiple parts, as was done with the progressive image ordering

during segmentation.

Each image is encoded as either an index frame or a delta frame. Index frames are fully encoded as standard images. Delta frames are stored as differences from previously stored frames. For images with a closely matching predictor, delta encoding takes less storage space.

Images are encoded by segment and in progressive order. For each image, the encoder begins by searching for the best possible predictor among previously encoded images within the segment. If no good predictor is found, the image is encoded as a index frame. Otherwise, the difference is computed and encoded as a delta frame. The compressed images are split into multiple resolution layers as each image is encoded.

6.1.5 Mapping the Dataset to GAL

A critical component of the prototype design is the mapping between the data representation and GAL's RG data model. The first step in developing this mapping is to identify the dimensions of adaptability. These dimensions define the RG's utility space within which the nodes and edges that make up the RG will be embedded.

The data representation described in Section 6.1.4 creates five adaptive dimensions. Two dimensions ($X \times Y$) are defined by the two-dimensional plane from which the images were captured. A third dimension, θ, corresponds to am image's view angle. The fourth dimension, δ, corresponds to the spatial density of the images. The fifth and final dimension, ρ corresponds to the resolution of the image. Together, these define a five dimensional utility space $\mathbf{U} = (X \times Y \times \theta \times \delta \times \rho)$.

Within $\mathbf{U}$ there are several nodes and edges that correspond to the individual images and their encoding relationships. Each resolution layer of each image has an associated node $n_i \in S$, where S is the set of all nodes within the RG. The location of each node, $\mathbf{Pos}\{n_i\}$, is determined by its location within the five dimensions as determined by the encoding process.

Edges are used to represent the data dependencies introduced by the predictive coding methods and wavelet-based transforms used during the encoding process. Edges are added to the RG that point from high resolution images to their lower resolution counterparts in recognition of the layered dependency relationship between different resolution versions of the same image. In addition, edges are used to connect delta frames with their predictive base. Finally, self-edges are added to all nodes representing the lowest resolution layer of index frames. These are the only nodes that can be

immediately decoded without any data dependencies.

Finally, clusters are introduced to the RG by placing all edges that correspond to the same segment of the database into a single cluster. In this way, data corresponding to images located nearby in the utility space is grouped together into a single atomic unit of data. By controlling the parameterization of the segmentation process during encoding, the size of these clusters can be widely varied. They can be large enough to encompass every edge in the RG, or small enough to force every edge into its own cluster. In essence, the number of partitions controls the granularity with which data is accessed via GAL.

6.1.6 Merging SWIM with GAL

As dictated by GAL's layered design, the prototype application must be configured with an application layer, a communication layer, and a server. The application must also specify cost and utility metrics as well as prediction and alpha vectors.

The Application Layer

The application layer is built around GAL's application template and contains the rendering and reconstruction code. The application also manages the prediction and alpha vectors for expressing changes in requirements. The prototype maintained a prediction vector with just a single point corresponding to the point of interest.

The position of the prediction vector was dynamically managed to map to the client application's viewpoint position. The alpha vector was used to manage the trade-off between the five adaptive dimensions. It was updated dynamically based on the speed of the viewpoint's movement through the scene. During periods of slow movement, the alpha vector was set to favor the resolution of nearby high resolution images. When the viewpoint moved quickly, the alpha vector was set to favor lower resolution images located further from the viewpoint.

The Communication Layer and Server

The communication layer is built around GAL's communication layer template and provides a subscription-based wrapper around, in this chapter, TCP/IP. The GAL library will ask the communicator to subscribe to a particular cluster. In response, the communicator users TCP/IP to communicate with the remote image server to

request all image data associated with the specified cluster. The communicator provides congestion control and reliable, in order delivery to GAL because it is based on TCP/IP.

Cost and Utility Metrics

The cost metric employed by the prototype application is very straightforward. The cost function assigns a cost to each edge equal to the number of bytes in the cluster to which it is assigned. Because data is accessed atomically by cluster, the overall size of the cluster determines the amount of data that must be transferred to the client before decoding the edge.

The utility metric is designed to mathematically express several assumptions about the IBR application. Primarily, images located closer to the viewpoint are most important because reconstruction quality is highest when the source images used as input are closest to the viewpoint. In addition, low resolution images are most critical as input to the reconstruction process because they provide the most information. Additional resolution layers of the images can be employed to provide additional information as they become resolved, but they are treated as a luxury by the utility metric.

In effect, these assumptions about the utility of data express the notion that far away low resolution images are about equal in utility to nearby high resolution images. This relationship can be easily expressed via a Euclidean distance metric which is the metric utilized in the prototype.

Prediction and Alpha Vectors

The prediction vector in the application prototype consists of just one entry. An individual prediction vector is maintained by each client. The single entry corresponds to the client's current point of interest. No predictions were used to drive the adaptation process so that only the current application conditions were taken into account. A more sophisticated prototype could make predictions of future locations of the point of interest based on user speed and direction of movement.

The application prototype's alpha vector consisted of five values chosen to manage the trade-offs between each of the five dimensions of the utility space. The alpha vector is configured in part to reflect the units associated with node positions inside the RG. For example, the prototype's default alpha vector is $\vec{\alpha} = \{50, 50, 3, 1, 1\}$ for the utility space $\mathbf{U} = (X \times Y \times \theta \times \delta \times \rho)$. The prototype assigns far greater alpha values to the spatial dimensions X and Y. This reflects the vastly different scale at which values are

assigned to node positions in these dimensions.

The prototype application's alpha vector is not static. The values are adjusted when the application detects that the user is moving faster than a specified threshold. When this condition is detected, the alpha vector is altered to place greater utility on the X and Y dimensions. This is accomplished by decreasing the alpha values associated with these dimensions. The lower alpha values bias the utility metric in favor of far away images.

Measuring the Effort of Application Implementation

The overall coding required to implement the application prototype was relatively small. As a rough measure of complexity, one can examine the lines of code associated with each component of the application. For example, the GAL library itself consists of roughly 15,000 lines of code.

In contrast, the application layer of the prototype contains approximately 1,000 lines of code (not including the actual SOI rendering module). The relatively modest size of this component shows the power of using GAL to manage the data management, adaptation, and communication portions of the application.

6.1.7 Specifics for the Input Dataset

The input dataset for the prototype consists of about 2,000 cylindrical color images, each with a resolution of 2,048×512. For all experiments in this dissertation, the RG used to represent the dataset consisted of 15,568 edges connecting 15,568 nodes embedded within the five dimensional utility space defined in Section 6.1.5. The number of clusters varied widely between experiments, ranging from 16 to 15,568.

6.2 Experimental Testbed and Methodology

The evaluation of the prototype application is designed to quantify the effectiveness of the GAL library and its underlying data representation and adaptation algorithm at satisfying the data requirements for a non-linear adaptive media streaming application serving several independently operating clients. This section describes the experiment testbed and the methodology behind the evaluation. This includes the network testbed used during evaluation, the network model that lies at the foundation of the evaluation,

the network topology used for the experiments, and the SUM performance metric by which the experimental results are measured.

6.2.1 The Emulab Testbed

The evaluation experiments were performed using the Emulab network emulation testbed (White et al., 2002). Emulab uses network emulation to recreate the operating conditions of a real network in a laboratory environment. The Emulab testbed provides a more realistic environment than simulation technologies because it transmits real packets over real network hardware to be received by real applications. However, unlike experiments performed on the open Internet, Emulab experiments are performed in a closed environment where competing network traffic can be carefully controlled.

6.2.2 Network Model

Because the experiments will be performed in a laboratory environment, several assumptions about the underlying network must be made when designing a custom topology. In particular, the assumption is made that bandwidth bottlenecks in a client-server application occur near the individual clients. Under this assumption, a server is provisioned with a rather large but constant level of network connectivity. Similarly, individual clients are provisioned with relatively narrow last-mile links which form the narrowest bandwidth portion of the path between client and server. Finally, the intermediate links between server and client (i.e. neither the first-mile nor last-mile links) are assumed to be adequately provisioned for whatever traffic is sent over them. The network model is illustrated in Figure 6.4.

6.2.3 Experiment Topologies

The Emulab testbed allows for the specification of custom topologies for network experiments. In all of the experiments presented in this chapter, a topology was designed based on the network model presented in Section 6.2.2.

The topology includes a single server provisioned with a 100Mbps network connection. This is the top speed supported by the Emulab testbed and therefore corresponds to a server that is maximally provisioned by a content provider aiming to distribute an IBR dataset to a large group of users.

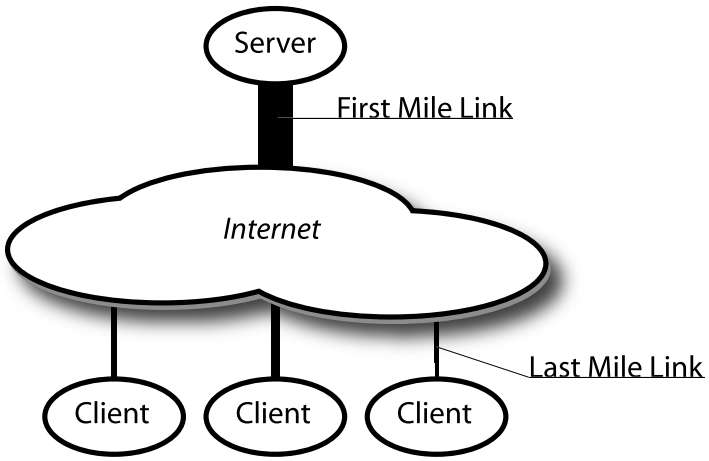

Figure 6.4: The network model for evaluation makes two key assumptions. First, the server is assumed to be provisioned with a large but fixed size first-mile link. Second, individual clients are assumed to be provisioned with relatively narrow last-mile links which form independent bandwidth bottlenecks. Finally, the core network is assumed to be adequately provisioned to handle all traffic sent over it.

The network model assumes that all bandwidth bottlenecks occur within the "last mile" for each client. All core links within the topology were therefore modeled with the same 100Mbps bandwidth as the server. Once again, the bit-rate is determined by the maximum rate supported by the testbed.

The core links connected interior routers of the topology in a tertiary tree. This created four links connected to each interior router: one upstream link and three downstream links. Once again, this design was dictated by the Emulab environment. In order to perform experiments at large scale, the topology had to make maximal use of the four network interfaces provided by Emulab at each node. For example, the 60 client topology illustrated in Figure 6.5 requires 131 nodes from the Emulab. Were the topology to model a more complicated interior network, even more nodes would be required, surpassing the capacity of Emulab. The fact that Emulab perform emulation using real machines and real networks provides an accurate look at network performance. However, the same fact forces the topology to reflect the real world limits of the testbed.

Finally, links connecting clients to the core network were configured as last-mile bottleneck links. This was achieved by provisioning edge links with a fixed bandwidth of between 0.1Mbps and 10Mbps depending on the experiment. Figure 6.5 shows a graphical depiction created of one experiments topology as created by the Emulab testbed.

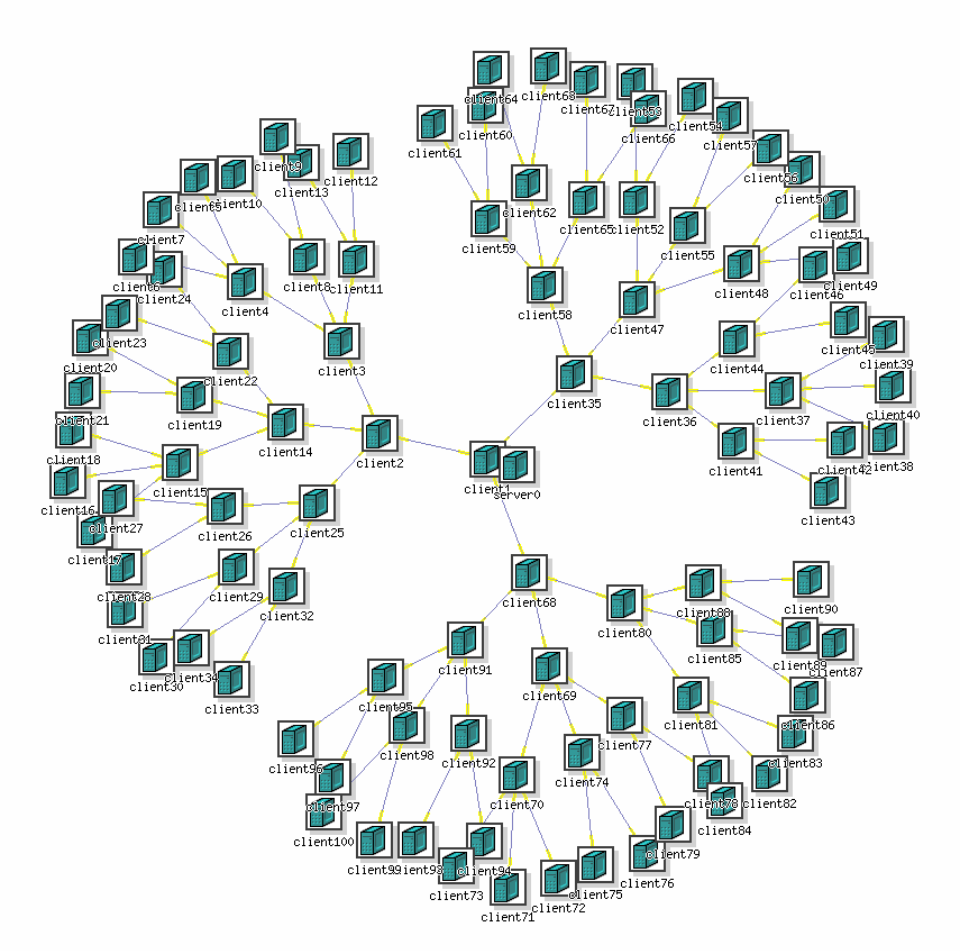

Figure 6.5: An experiment topology with 60 clients is depicted in this image taken from the Emulab testbed's web interface. This topology, even though it uses a tertiary tree for efficiency, nearly surpasses Emulab's resources by requiring 131 PCs to be provisioned.

6.2.4 The SUM Metric

The experiments presented later in this chapter are designed to measure the adaptive performance of GAL in supporting the prototype application. In support of these experiments, a performance metric must be defined that can evaluate a system's adaptive behavior.

An important property of the performance metric is that it must be application independent. This is because the goal is to measure how well GAL supports the adaptive policies of the application, not how well the application's cost and utility metrics map to user satisfaction. Such a metric can be defined by as a function of the adaptation

layer, itself an application-independent concept. Stated broadly, the performance metric measures the performance of GAL by measuring how well it can adapt the flow of data to match the needs of the application independent of how well the application can express its needs.

The proposed performance metric, the SUM, is a metric that measures system performance as a function of the current state of the representation graph. The SUM is designed to provide a numerical measure of system performance at a given point in time using only information that is available to the adaptation layer. This includes the representation graph, the alpha and prediction vectors, and the utility metric.

The SUM is derived from the notion that the adaptation layer's performance can be measured by the utility of the data it has obtained at any given point in time. This can be measured by applying the current utility metric to every resolved node in the representation graph. We then sum all of the resolved node utility values to find the SUM. We formally define the SUM metric in Equation 6.1.

$$\mathbf{SUM} = \sum_{n_i} UtilMetric(n_i) : n_i \in S \wedge \mathbf{State}\{n_i\} = Resolved \tag{6.1}$$

It is important to note that the SUM is a measure of performance at a single point in time. To capture a reliable measure of system behavior, the SUM metric must be evaluated repeatedly over a period of time.

6.3 Performance Evaluation using TCP Data Delivery

In this section, the results of several performance experiments are presented that highlight the proper adaptive behavior achieved by GAL. In addition, the experiments highlight that even the SWIM approach to streaming IBR, which performs reconstruction on the individual clients, scales poorly in support of large user groups. The scaling problems highlighted in the experiments shown here form the motivation for the Channel Set Adaptation (CSA) technique presented in Chapter 7.

6.3.1 Experiment Description

For each experiment presented in this chapter, the network topology consisted of one central image server with a constant number of clients. The exact number varied by

experiment. Throughout the life of an experiment, clients autonomously navigated along a ten minute path through the IBR dataset.

During the ten minute execution time, the SUM metric was computed once per second. When presenting average SUM values, only the second five minutes of SUM data were considered. This policy ensured that only steady-state behavior was included in the results presented below and that transient start-up behavior had no impact on the evaluation. However, in practice the results generally held even when the initial time period was included.

6.3.2 Negligible Impact of Path Choice on Performance

Throughout all of the experiments, the path along which the clients navigated the dataset included a variety of movement patterns. The movements included both fast and slow movements as well as changes of direction. In addition, movements included forward, backward, and sideways translations of the viewpoint. The varied nature of the path choice is designed to limit the impact of specific path properties on the overall results.

However, the specific path choice has a negligible impact over the presented results for several reasons. First, the input dataset is relatively uniform in density across the navigable dimensions. This leads to little variation in performance between different paths.

Second, most of the experiment results are presented as averages of the performance metrics. Because the characteristics of the specified path varies widely over the course of each ten minute session, any transient advantages due to path choice are averaged out over the course of each session.

Finally, the independence of the experimental results from the navigation path can be seen by closely examining the non-averaged results which plot performance over time. These results show that even over limited time windows, during which the path exhibits more homogeneous characteristics, the relative performance of various system configurations remains similar to the overall average results. For example, the plot in Figure 6.7 was calculated by averaging the results from 6.6 over the entire session. Averages computed over more limited time windows exhibit similar characteristics.

6.3.3 Adaptation Performance Over Time

The first experiment in this chapter examines the adaptation performance of the prototype over the course of session. Performance was measured using the SUM metric as defined in Section 6.2.4. The experiment used a single client TCP-based topology as described in Section 6.2.3. The client machine in the topology was instrumented to capture SUM values once a second for the entirety of a ten minute session.

Plotted directly against time, the SUM value can fluctuate dramatically as application conditions change and data is resolved. For example, a sharp change in position of view direction of the user's viewpoint can make the entire set of resolved data immediately less useful to the application than it had been just seconds earlier.

The cumulative SUM value can be used to represent the SUM values more smoothly when viewing them as a function of time. If we consider the sequence of SUM values taken from time 0 to time t, the cumulative SUM can be defined as in Equation 6.2. The cumulative SUM creates a monotonically increasing metric that avoids the traditional SUM's fluctuations in value and allows for a cleaner illustration of a system's behavior over time.

$$\textbf{CumulativeSUM}(t) = \sum_{i=0}^{t} SUM(i) \tag{6.2}$$

The results of several sessions are summarized in Figure 6.6. In each session, the SUM values were recorded at intervals of one second and plotted using the **CumulativeSUM** metric. The ten plots in the figure correspond to ten unique configurations of the experiment. Each session was performed using a different last-mile bottlenck bit-rate, ranging from 0.10 Mbps to 10 Mbps. In all of these sessions, every edge in the was placed in its own cluster, yielding 15,658 unique clusters. This configuration allows the most flexibility of data access and therefore the greatest adaptive power.

The results indicate that the GAL library performs well in several important ways. First, they show that despite dramatically different behavior during various time periods, the cumulative SUM value continues to grow steadily. This indicates that under all conditions, such as changes of viewpoint direction, changes of viewpoint speed, and changes of viewpoint orientation, the GAL library is able to reliably deliver useful data to the application layer, as judged by the utility metric.

Second, the similar shape but disparate slope for each plot shows that GAL ap-

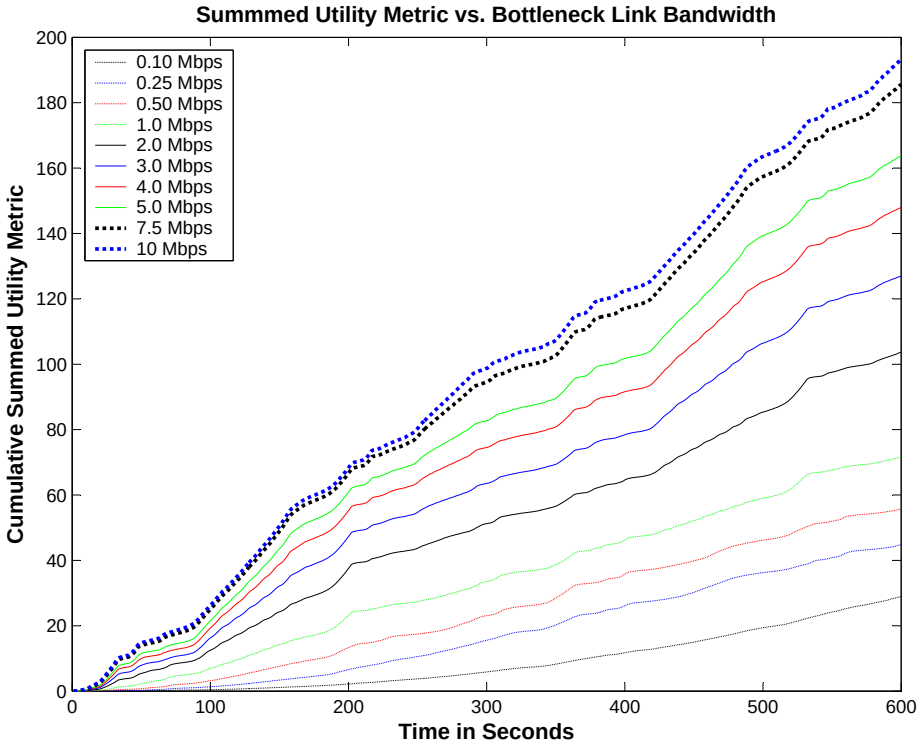

Figure 6.6: This graph shows three plots of cumulative summed utility values over the course of a ten minute session. The plots correspond to three different bottleneck link speeds. The steady growth of the three plots shows that over time, GAL continues to provide data of high utility. The difference in slope shows that GAL properly utilizes any available bandwidth to improve utility as much as possible.

propriately makes use of additional bandwidth to improve the overall utility of the set of resolved nodes. Closer attention to the differences in slope shows that the benefit in terms of SUM does not grow linearly with the speed of the bottleneck link. This behavior is explored in more detail in the next experiment.

6.3.4 Bottleneck Link Impact on Adaptation Performance

As hinted at in the previous experiment, the benefit of additional bandwidth at the bottleneck link is sublinear. This fact can be illustrated clearly by analyzing average SUM values as a function of the bottleneck link speed.

This experiment again used a single client topology where the client was instrumented to capture SUM values once a second for the entirety of a ten minute session. In this experiment, the series of SUM values was averaged to yield a single numerical measure of the adaptation performance over the life of the session. In all of the ses-

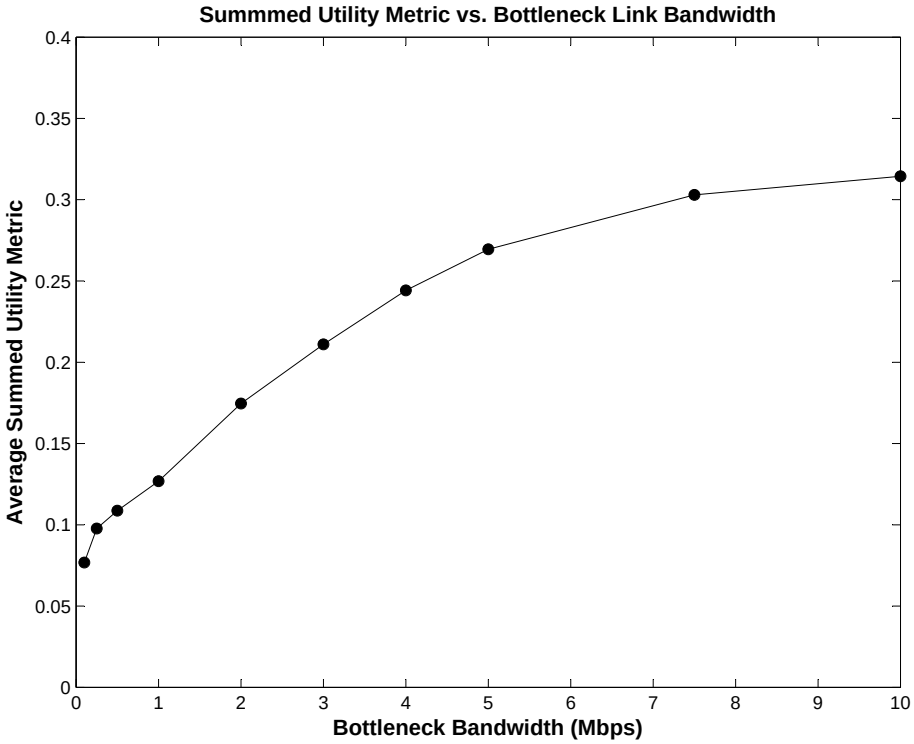

Figure 6.7: This figure shows the average summed utility metric value for several sessions. Each session was executed with a different bottleneck link speed. As expected, the plot shows an increase in utility for higher link speeds. However, the shape of the curve is not linear. This highlights the diminishing marginal benefits of additional bandwidth.

sion for this experiment, the RG was configured to place each edge in its own cluster, yielding 15,568 unique clusters. The results are shown in Figure 6.7.

This experiment confirms the conclusions drawn from the first experiment: that GAL correctly takes advantage of excess bandwidth to improve the utility of the set of resolved nodes. However, additional focus is placed on the marginal utility gained by additional bandwidth at the bottleneck link.

The behavior is illustrated by the shape of the curve in the plot. The steepest gains in utility are found when increasing bandwidth from a very low value. The gains tend to plateau as the bottleneck link speed grows faster and faster.

While not necessarily intuitive, the decrease in marginal utility at higher bit-rates is the expected result. For example, suppose that with a bottleneck bit-rate of x, a system using GAL is able to resolve the n nodes with the highest utility values. In this scenario, not a single unresolved node has more utility than any of the n nodes that the system was able to resolve.

Meanwhile, assuming all nodes are the same size, a system with a bottleneck bit-rate of $2x$ (twice the previous example) would be able to resolve those same n nodes plus n additional nodes. However, if GAL is operating correctly then the additional n nodes would never be as useful as the first n. Therefore, while the system with twice the bandwidth was able to resolve twice as much data, the additional data was not as useful. System engineers who need to double the performance of an adaptive system must more than double the bandwidth available for adaptation.

In summary, the results of this experiment shows that there is a decreasing marginal utility for the added bandwidth. Played out over several bottleneck link speeds, the results show the clearly sub-linear growth in utility as the bit-rate grows. In fact, the plot grows asymptotically toward an upper bound corresponding to the highest possible SUM value. In the extreme, where the bottleneck bit-rate is infinite, the best performance possible would be to resolve all nodes in the RG. The SUM value corresponding to this ideal state serves as the upper bound on performance.

6.3.5 Impact of Clustering on Performance

An important engineering parameter when mapping a specific dataset to to the abstract RG data model is the specification of clusters. Under the RG model, clusters are groups of edges that form units of data that are accessed atomically. For a given dataset, the range of options is quite wide.

At one extreme, every edge can be place within its own cluster. This allows each edge to be individually addressed, providing the maximal amount of data access flexibility. However, there is an increase in the required overhead in managing all of the cluster subscriptions.

At the other extreme, all edges are grouped into a single cluster. This is equivalent to simply downloading a single large file with a static data ordering. This does not provide any data access flexibility, but it dramatically reduces the amount of management overhead because there are no choices to be made.

In practice, it will be best to choose a middle ground between these two extremes that provides adequate flexibility without incurring too much overhead. This is particularly true for network models that have large subscription penalties such as IP Multicast. However, the network model employed in this chapter is based on the unicast protocol TCP/IP which has the smallest subscription delay possible: one round trip time.

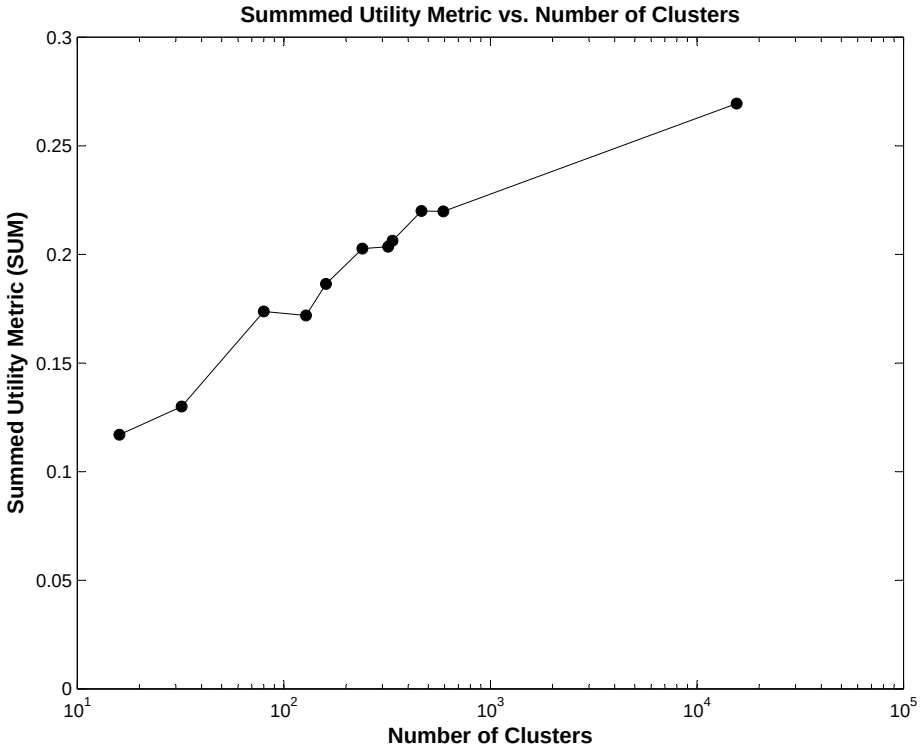

Figure 6.8: This figure shows the impact of clusters on overall performance. As the number clusters decreases, so does the flexibility of data access patterns, resulting in a drop of the average SUM value. We call this the clustering penalty. However, the clustering penalty is not evenly distributed. The penalty is greatest when there are a very small number of clusters. As the number of clusters grows, the penalty gets smaller and smaller. This leads to the linear shape of the plot using a logarithmic scale for the number of clusters.

The experiment presented here was designed to explore the impact of clustering on performance. This experiment used a single client topology where the client was instrumented to capture SUM values once a second for the entirety of a ten minute session. The series of SUM values was averaged to yield a single numerical measure of adaptation performance over the life of the session.

The number of clusters in the RG was varied for each session. At one extreme, every edge in the RG was placed in its own cluster, resulting in 15,568 clusters. At the other extreme, the dataset was split into just 16 clusters, averaging nearly 1,000 edges each. This greatly restricted the data access patterns available to the application, but lowered the overhead needed to make subscription requests. For example, only 16 subscriptions requests need to be made to resolve the entire dataset. The results are shown in Figure 6.7.

The results show that as the number of clusters drops, the average SUM value decreases. For all sessions, there was a single client with a fixed bottleneck bandwidth of 5Mbps. The only setting that changed between sessions is the number of clusters, and the drop in utility is a direct result of this clustering parameter. I refer to the drop in utility as the *clustering penalty*.

It is important to note that the graph shows the number of clusters using a logarithmic scale. Therefore, the linear shape of the plot indicates a substantial clustering penalty only for configurations with relatively few clusters. Additional clusters will improve quality by increasing the flexibility of data access. However, there is a decrease in the marginal benefit of increasing the number of clusters as the cluster count grows. This result implies that a small amount of clustering to reduce communication overheads can be used without seeing a dramatic impact on system utility.

For the TCP/IP-based application presented in this chapter, there is little benefit from reducing the number of clusters. With just one client, as in this experiment, there is no benefit as the session with the greatest number of clusters performed best. However, as more and more users access the data simultaneously, decreasing the cluster count can decrease the number of data requests arriving at the server. In addition, reducing the number of clusters may be desirable for implementations that utilize alternative network models that have substantial delays when servicing a subscription request.

6.3.6 Scaling to Large User Groups

The GAL design places the burden of adaptation on the client side. This is particularly critical when there are a large number of users simultaneously accessing the server. If the server were responsible for adaptation, the server's CPU load would grow linearly with the number of clients. However, the client-side adaptation architecture means that the server's CPU doesn't perform any adaptation tasks. Instead, the number of users is limited only by the amount of outgoing bandwidth provisioned to the server.

To test this property, this section presents an experiment based on several sessions that were run with a variety of user group sizes. For all sessions in this experiment, there was a single server connected via the tertiary tree topology to a group of clients, each of which were provisioned with a 5Mbps bottleneck link. In addition, each client was allowed to navigate independently through the IBR dataset. The RG was configured to place each edge in its own cluster, yielding 15,568 unique clusters.

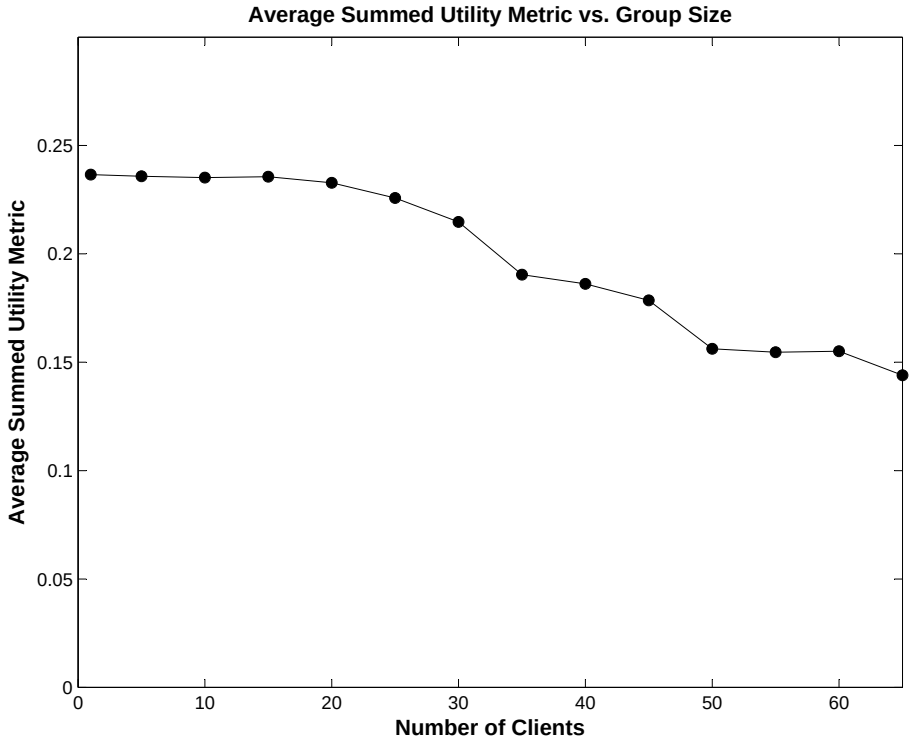

Figure 6.9: This figure shows the average summed utility metric for several sessions. The sessions were performed with a variety of user group sizes ranging from one to sixty. Given the server bandwidth allocation of 100Mbps and a per-client bottleneck speed of 5Mbps, the drop off in performance at 20 clients is expected. This is because the system's overall performance is bandwidth limited.

The results from these sessions are shown in Figure 6.9. As the plot illustrates, there is essentially no drop-off in performance until the number of clients surpasses twenty. The drop-off point is where we would expect given the server's link speed of 100Mbps and a per client link speed of 5Mbps. This illustrates that the scaling limit for the TCP/IP prototype is the server-side first-mile link speed.

While the client-side adaptation architecture alleviates the CPU bottleneck on the server, it does little to limit the bandwidth bottleneck. The only control over bandwidth is the number of clusters. However, as the experiments in this chapter have shown, performance remains best at high cluster counts despite the added number of subscription requests.

In the next chapter, a novel technique is presented that targets the bandwidth bottleneck that plays such a key role in limiting the number of simultaneous users. Together with the client-side adaptation architecture presented in this chapter, the new technique enables truly scalable streaming for non-linear media.

It is interesting to note that even as the group size grows past 60, more than triple the drop-off point of 20, the drop in performance is perhaps less than expected at first glance. While the bit-rate per client will drop by two thirds when there are 60 clients, the decline in the SUM is less than 50 percent. This relationship, however, is indicative of proper behavior and is a result of the same phenomenon discussed in detail in Section 6.3.4.

6.4 Summary

This chapter presented the details of the experimental prototype used throughout this dissertation. The prototype is a media streaming system for IBR datasets based on the SOI algorithm developed by Aliaga et al. As a motivating example application, the prototype is designed to meet the needs digital museum that wants to share digitized IBR models of important spaces to large groups of independently operating virtual visitors.

After describing the motivating application and the SOI algorithm itself, this chapter presented two approaches to the IBR streaming problem and explained the client-side adaptation technique adopted in the prototype. This option was chosen for its scalable properties, such as the transfer of per-client work to the client in adherence to the simple server design philosophy.

Following the IBR streaming design discussion, the chapter detailed the data representation used in our prototype and described its mapping to the GAL libraries RG data model. This included the definition of a five-dimensional utility space and a 15,568 node RG.

Finally, the results from four experiments were presented. The experimental methodology and testbed were described, including details on the Emulab network emulator, the network topology used throughout the experiments, and the SUM performance metric.

The experiments evaluated several of the prototype's performance properties. These included measurements of the adaptation algorithm's performance over time, its performance over varying bottleneck link speeds, and the impact of clustering on adaptation performance. The final experiment highlighted that even the client-side adaptation architecture could alleviate the CPU bound to system performance. The server-side bandwidth remained as the limiting factor in how many simultaneous users could be supported.

Chapter 7

Scalable Delivery Architecture

The experimental prototype evaluated in the previous chapter performed well for relatively small groups of users. For one or a few simultaneous clients, the system architecture based on TCP/IP's unicast network model was able to support effective adaptation and deliver a stream of data to each user that matched its individual data requirements. The client-side adaptation approach moved both adaptation and reconstruction to the client and removed the CPU bound on server performance. This enabled the support of a handful of clients from a single IBR server.

However, as the final experiment in Chapter 6 highlighted, the solution was unable to support large groups of simultaneous users. Because individual data streams are delivered to each client, the server-side bandwidth requirement grows linearly with the number of clients. This leads to a bottle-neck on the first-mile link over which the server must distribute all of the outgoing data streams. For very large groups, the performance quickly drops below acceptable levels. Unfortunately, the motivating digital museum application for this technology requires that very large numbers of independently operating clients be able to satisfactorily access the centrally stored IBR data set.

In this chapter, I describe an architecture for scalable data delivery that alleviates the server-side bottleneck, enabling IBR streaming that scales effectively to support very large user groups (Gotz, 2004). The architecture, called CSA (Gotz and Mayer-Patel, 2005c), takes the simple server design philosophy outlined in the previous chapter to the next level by removing literally every per-client task from the server. This is accomplished by employing a group-based channel oriented network model in a novel way to allow non-linear media streaming to large groups of users.

This chapter begins with an overview of the design decisions relevant to creating a

scalable delivery architecture. It then presents a detailed description of CSA, a novel framework for supporting highly scalable non-linear media streaming. The chapter concludes with a presentation of several experimental results that highlight CSA's most important performance characteristics over both broadcast and multicast network models. Throughout the results, a CSA performance model is developed to highlight the important factors that contribute to overall system performance.

7.1 Achieving Scalability

A scalable solution for non-linear media streaming requires a carefully balanced design that can manage the trade-off between (1) the requirement of delivering custom flows to each client and (2) the need to remove per-client work from the server for scalable performance. This section presents a number of design considerations that address this trade-off. It first revisits the simple server design philosophy outlined in Chapter 6. It then describes the spectrum of possible delivery solutions.

7.1.1 Simple Server Design Philosophy

Achieving scalable delivery of non-linear media can be accomplished by adhering to the simple server design philosophy first described in Section 6.1.3. This design philosophy encourages the migration of all per-client tasks from the server toward individual clients. This client-driven approach is motivated by two factors.

First, the philosophy strives for a constant server load model. If the server itself is responsible for any per-client tasks, the goal of a constant server load is impossible to reach because additional work will be required for every additional client that accesses the sever.

Second, adaptation is performed independently on each client and must reflect local system and application conditions. Therefore, the logical location for per-client adaptive decisions is on the individual clients themselves.

These two factors led to the adoption of the simple server design philosophy where the centralized server is tasked with constant level work loads that are equally useful for all participants and independent from the needs of any individual clients. The simple server design leads directly to a bounded server load that is independent of the number of participating clients.

7.1.2 Spectrum of Delivery Solutions

There is a wide spectrum of possible solutions for delivering non-linear media to large audiences. This section first presents a sample application to give our discussion a concrete context. It then outlines the two extremes of the solution spectrum. Finally, this section describes the compromise approach adopt in the CSA solution.

Example Non-Linear Application

As an illustrative example of a many-user non-linear streaming application, this chapter uses the same digital museum application as in Chapter 6. The sample application is described briefly below.

Consider a digital museum that aims to digitize and share a famous space (e.g., the Palace of Versailles) with a group of virtual visitors from around the globe. This could be accomplished by capturing a large set of digital pictures from the scene, storing them in an IBR dataset, and making them available on-line.

IBR is a computer graphics technique that uses real world pictures from a scene as input, and renders novel photo-realistic views of the scene in response to a user moving a virtual viewpoint. The novel views are generated by interpolating between the captured samples. Users can navigate through the virtual space interactively, exploring the scene with the same freedom that video game players have while exploring a game's virtual setting.

IBR datasets are typically very large in size. A digital museum would therefore want to stream the dataset to each user to avoid long download delays. In addition, because users will be navigating the scene independently, they will each require a unique flow of image data. Such a system would therefore need to support non-linear streaming that scales to support a large group of museum visitors.

The Adaptive Extreme

There are a variety of possible solutions for supporting the digital museum streaming application. At one extreme, the most adaptive architecture for streaming IBR data is the unicast client-server model proposed in Chapter 6. Under this model, each client first obtains a list of all available images and their semantic (e.g., position in space) and syntactic (e.g., encoding dependencies) relationships. For applications using the RG data model, this can be accomplished by sharing the representation index (see Section 3.1.4).

Armed with the representation index, a client would iteratively determine which images are most important using a benefit function and request those images from the server, for example using the GAL library and the UCR adaptation metric. As shown in the experiment of Section 6.3.5, the unicast model performs best by placing every edge into its own cluster enabling data requests for specific images.

Allowing the client to make individual image requests provides the highest degree of adaptive behavior to each client. They can custom compose the incoming stream of images by specifically requesting each photograph.

However, this approach does not take advantage of any similarity in interests across users. The server must respond individually to each client's requests and the server's outbound bandwidth requirement grows linearly with respect to the number of clients. This design does not scale well and violates our simple server design philosophy by requiring per-client streams to be sent out by the server. The end result of this design is a dramatic drop in performance past the point of first-mile link saturation as illustrated by the experiment in Section 6.3.6.

The Scalable Extreme

At the other extreme, the most scalable architecture for the digital museum application is to cluster all of the data into one large unit and transmit the data over a single multicast channel. Under this design, the server would repeatedly transmit the information on a carousel. Individual clients would then tune in to the channel, continuing to receive data until the entire dataset has been downloaded.

This architecture is infinitely scalable because the server does not perform any per-client work and the design adheres strictly to the simple server design philosophy. However, this solution is not practical for non-linear applications. Clients have no options for adapting the flow of images and must settle for the predefined linear ordering chosen when grouping the images into a monolithic cluster of data. This solution essentially degrades the fully interactive IBR application model into a non-adaptive flow of images: a static video stream.

This extreme of the spectrum is the reason that multicast works so well for linear media distribution where all users have identical interests. When all users are satisfied with the same flows of information, as in video or audio streaming, multicast transmission of the streams is the ideal scalable solution because it adheres to the simple server design philosophy.

However, in practice, the straightforward multicast solution is not even satisfactory

for many linear media streaming applications which often require mono-dimensional adaptation to control the bit-rate of data transmission over a constrained network. This has been the motivation behind technologies such as Receiver-driven Layered Multicast (RLM) (McCanne et al., 1996).

This extreme falls even farther short in support of non-linear media, where every client requires control over not only the rate of data delivery but the content of data delivery as well. These dual requirements for both per-client rate control and per-client content control make the traditional multicast unacceptable for non-linear media.

A Scalable and Adaptive Solution

The ideal solution for the illustrative application, and for scalable non-linear media streaming in general, would retain both the adaptive nature of the first extreme and the scalable properties of the second extreme. It turns out that these goals can be reached through a middle-ground approach which achieves both scalable and adaptive distribution of non-linear media.

Under the hybrid approach, related data elements would be grouped into larger units, as supported by the cluster component of the RG data model. This would partition a media object into several large blocks, each of which has a semantic meaning. For example, in the sample application, images might grouped so that all the low resolution pictures from one corner of a room are placed within a single cluster of images.

Each cluster could be distributed using multicast to scalably deliver them to all interested clients. For example, if five users were exploring the same corner of a room, they could all subscribe to the multicast stream that contained the associated cluster of images. Users in a different room would choose instead to subscribe to an alternate multicast stream with an image cluster that more closely matched their requirements.

If clients were aware of the available multicast channels and their associated semantic meanings (e.g., which images are in which cluster, and which clusters are on which multicast channel), they could intelligently and independently subscribe to the multicast streams that contain information most relevant to their needs. As those needs change over time, clients could quickly choose to subscribe to whichever multicast streams had become most appropriate.

Clients would be able to compose a custom flow of images based upon the order of their subscriptions. At the same time, the server would perform no per-client work because it would only be responsible for transmitting a fixed number of multicast

streams on a regular schedule. This channel-based approach to providing scalable and adaptive access to non-linear media forms the conceptual foundation for CSA, a novel approach to scalable non-linear media streaming.

By varying the degree to which images are clustered, this hybrid approach is powerful enough to represent delivery solutions at both extremes of the spectrum. By placing every image in its own cluster, we reach the extreme of allowing every user to request individual pictures. By clustering all images into one big cluster, we achieve the scalable extreme using a single multicast channel.

The choice of exactly how to cluster the images into larger groups and how many clusters to employ can have a large impact on the effectiveness of the architecture. The proper choice will depend greatly on both the cluster penalty on performance (as explored in Section 6.3.5) and the overheads associated with a particular multicast protocol.

7.2 Channel Set Adaptation

This section outlines CSA, a framework that enables efficient streaming of non-linear datasets to large user groups. CSA allows individual clients to request custom data flows for interactive applications using standard multicast join and leave operations. CSA scales to support very large user groups while continuing to provide interactive data access to independently operating clients.

The CSA framework is built upon the RG data model presented in Chapter 3. This section begins with a quick overview of the RG components relevant to CSA. It then describes the CSA communication model designed to provide scalable service to large user groups. Finally, it covers the client-driven adaptation algorithms that perform both congestion and content control.

7.2.1 CSA and the RG Data Representation

A critical task in the CSA framework is the expression of relationships between individual data elements. This task requires formal structures for expressing both syntactic relationships (e.g., encoding dependencies) and semantic relationships (e.g., similarity in meaning or utility). The RG data model presented earlier in this dissertation provides the representational power required by CSA and serves as the underlying data structure. This subsection provides a brief overview of the RG data representation. For

more detailed coverage, refer to Chapter 3.

The RG data model is a flexible representation abstraction designed specifically for expressing both syntactic and semantic data relationships in multimedia databases. The RG abstraction also provides mechanisms for evaluating the relative utility of individual elements of information in the database based on dynamic system conditions.

A RG is composed of a graph-based structure embedded within a multidimensional utility space. Individual elements of information are modeled as *nodes*. Syntactic dependencies are expressed via a set of *edges* that connect sets of dependent nodes. Semantic relationships between nodes are expressed by the nodes' positions within the utility space. Furthermore, the RG model defines *clusters* as groups of edges which are accessed atomically. Each cluster is considered a semantically consistent unit of data. The underlying structure of a RG, including the list of nodes and their connectivity, is stored explicitly as the *representation index*.

There are two parts of the RG abstraction that are particularly important within the context of CSA: (1) Clusters and (2) the representation index.

Clusters

A cluster is a block of data, corresponding to one or more edges in the RG model, which is semantically consistent and accessed atomically. For example, clusters in Section 7.1.2 represented groups of images captured from similar locations. When modeled using a RG, a dataset is essentially partitioned into a set of clusters, $C = \{c_1, \cdots, c_n\}$.

Each cluster, c_i, has two key properties. First, c_i has a set of assigned edges, $\mathbf{Edges}\{c_i\}$, which correspond to the data associated with the given cluster. Second, c_i has a cost function, $\mathbf{Cost}\{c_i\}$, which defines the cost of accessing the block of information.

The Representation Index and The CSA Index

The representation index is a specification of the underlying structure of the RG. The index is a concise enumeration of the nodes, edges, and clusters that make up the RG, as well as the definition of the utility space in which the graph.

The CSA architecture defines a new index description called the *CSA index*. This structure uses an XML-based index format to represent both the RG's standard representation index as well as additional metadata required by CSA. This meta data includes the cluster-to-channel mapping M that will be described in Section 7.2.2.

```
<gal>
  <utilityspace dimensionality=5>
    <subspace id=0 dimensionality=3>
      <dimension type=navigable name=x> </dimension>
      ...
    </subspace>
      ...
  </utilityspace>
  <nodelist>
    <node id=1 subspace=0 pos=12.332,23.32,1.1> </node>
    ...
  </nodelist>
  <clusterlist>
    <cluster id=1 resource=30012 cost=308>
    ...
  </clusterlist>
  <edgelist>
    <edge id=1 src=12 dest=14 cluster=1> </edge>
    ...
  </edgelist>
</gal>
```

Figure 7.1: A brief sample the XML index format used within CSA to specify both the RG representation index as well as metadata required by CSA. The CSA index includes a description the inherent structure of the RG data representation (i.e. nodes, edges, and clusters), while omitting the actual encoded data.

A sample of the CSA index format is shown in Figure 7.1. The CSA index does not include any actual media data and is therefore very small in size in comparison to the overall dataset.

7.2.2 Media Communication Model

The CSA media communication model is designed to meet two goals. First, the model must allow individual users to access the non-linear media interactively and independently. In other words, it must provide custom flows of information to each user in response to their own unique and locally determined requirements. This requirement argues for a unicast client-server approach, capable of delivering a unique data flow to each user.

Second, the model must scale to support large groups of independent users. Unfortunately, the unicast solution for meeting the first goal places a load on the server that

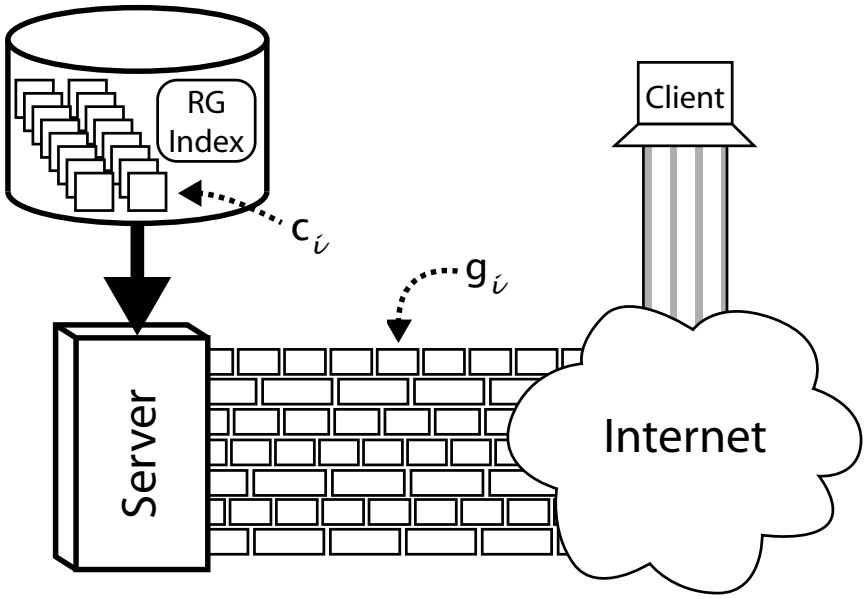

Figure 7.2: CSA enables efficient streaming of non-linear media to large groups of independently operating users. The CSA communication model partitions a media object into semantically meaning clusters, labeled c_i. These clusters are then mapped to a large set of broadcast or multicast channels, labeled g_i. Clients compose custom data flows that match their local application requirements without ever contacting the central server by managing their Active Channel Set through scalable subscription operations.

scales linearly with the number of participants. As the final experiment from Chapter 6 shows, this makes it extremely difficult to support large groups of users.

This section presents a new media communication model that is not based on unicast transmission which provides a solution that meets the competing goals for both interactivity and scalability. The approach delivers custom data flows to each user while maintaining a constant and bounded server load that is independent of the number of users.

There are three primary components of our design: (1) channel-based transmission, (2) session initiation, and (3) client behavior. These are each covered in the subsections below. This discussion concludes with an analysis of the implications of the communication model on the two design goals.

Channel-Based Transmission

CSA requires the central server to maintain a large set of communication channels, noted as $G = \{g_1, \cdots, g_n\}$. In this context, a channel is an individual data flow to which users can subscribe and unsubscribe. Upon subscription, users have no control over the data contained in an individual channel. They must either accept the data flow assigned to the active channel, or unsubscribe to terminate the flow of information.

The subscription model of the channel-based transmission scheme can be easily supported in both broadcast and multicast networks. The subscription model is also aligned with the subscription-based communication layer API specified by GAL (see Section 5.1).

The number of channels in G is equal to the number of clusters in the RG model used to represent a media object. A one-to-one mapping $M : C \mapsto G$ maps each cluster $c_i \in C$ to a corresponding channel $g_i \in G$. Because clusters are semantically consistent blocks of data (e.g., sets of images from the same location), the mapping M assigns a semantic meaning to each channel g_i. The mapping information in M is appended to the CSA index. A discussion on how individual clients make use of this mapping to support non-linear data access is presented in Section 7.2.3. This section concentrates on the server's responsibilities.

At runtime, the server simultaneously transmits all channels in G. Each channel g_i is transmitted at a constant bit rate. It is important to note that the data assigned to each cluster is typically finite in size. In this case, the server transmits the data on a carousel transmission schedule, repeatedly sending out the entire cluster of data with a cyclical schedule. The server is not responsible for any other tasks. The overall architecture is shown in Figure 7.2.

Session Initiation

Clients are responsible for initiating a new sessions. The client's first step in creating a new session is to obtain a copy of the CSA index. The mechanism for this transaction is not specified as part of the CSA framework and must be supported through some out-of-band mechanism. For example, the CSA index could be made available through a well-known HTTP or FTP host.

The XML description of the CSA index contains the cluster-to-channel mapping, M, as well as the traditional representation index contents describing the semantic and syntactic data relationships of the associated non-linear media dataset. The CSA

index is essentially a menu describing which communication channels are available as well as each channel's assigned semantic meaning. For example, in the digital museum application, the CSA index would specify which channels contained images from each part of the Palace of Versailles.

Client Behavior

Following session initiation, a client has all the information it needs to begin receiving the non-linear data stream. Using the a client-driven adaptation algorithm that is described in Section 7.2.3, the client begins to manage its Active Channel Set (ACS).

The ACS is a list of all channels to which the client is currently subscribed. By choosing which channels are in the ACS as well has how many channels are active any any point in time, a client can compose a unique stream that delivers a custom flow of non-linear media data that is individually tailored to meet the needs of the client.

Satisfying Design Goals

The media communication model meets the two primary design goals. First, individual users can access the non-linear media stream interactively and independently through management of the ACS. Second, the model easily supports large groups of independent users because of the channel-based transmission design. Coverage of the algorithms for managing the ACS is deferred until Section 7.2.3. The remainder of this section concentrates on the scalable properties of the communication model.

As outlined in the simple server design philosophy, a key requirement for any scalable solution is the removal of all per-client work from the server. This requirement is achieved in CSA by utilizing a channel-oriented network infrastructure, which can be supported by a broadcast or multicast network. By their nature, these network protocols don't require any per-client work by a server. The utilization of these network models leads to a highly scalable server-side solution whose performance is independent of the number of participating clients.

The independence of server performance from the number of clients is a critical property in CSA. It allows for the determination of a constant upper bound on computation and bandwidth requirements. As a result, servers can be properly provisioned with a finite and static level of resources to support, in the ideal case, an infinite number of simultaneous users.

The upper bound property is critical in supporting scalable delivery. In contrast,

the unicast model described in Chapter 6 places a load on the server that grows linearly with the number of users. This makes it impossible to provision a server with a constant amount of bandwidth, no matter how plentiful, that will be adequate for any user group size.

However, the scalable nature of broadcast and multicast data delivery is well known. The key contribution of CSA is that it provides a framework for using these scalable technologies for non-linear media, where every receiving client requires a unique data flow.

The following subsection describes CSA's novel method for allowing the composition of individualized flows via broadcast and multicast networks typically employed for homogeneous data distribution.

7.2.3 Client-Driven Adaptation

The client-driven adaptation algorithm that forms the foundation of CSA is aligned very closely with the iterative adaptation algorithm presented in Section 4.10. As dictated by the simple server design philosophy, Individual clients are responsible for adapting their own incoming data flows to match their own application preferences and resource requirements. Adaptation is performed independently, without any assistance from the centralized server, as each client manages their own ACS.

ACS management is performed through two fundamental operations. The first operation, $\mathbf{Sub}\{g_i, ACS\}$, is used to subscribe to a new channel. Upon subscription, the new channel is added to the ACS. The second operation, $\mathbf{Unsub}\{g_j, ACS\}$, is used to unsubscribe from an already active channel. This operation removes channel g_j from the ACS assuming it is a member. The two operations are defined below.

$$\mathbf{Sub}\{g_i, ACS\} = ACS \cup \{g_i\} \tag{7.1}$$

$$\mathbf{Unsub}\{g_j, ACS\} = ACS \setminus g_j \tag{7.2}$$

Both the subscribe and unsubscribe operations can be performed in broadcast or multicast networks without any direct contact with the server. Adaptive data flows as well as scalable performance can be enabled by defining adaptation in terms of these two operations.

The client-driven adaptation algorithm must accomplish two tasks. First, it must perform *congestion control* to manage the speed at which data arrives. Second, it must

perform *content control* to achieve the individualized data flows required by non-linear media applications. The following sections define both of these adaptive tasks in terms of the subscribe and unsubscribe operations outlined above.

Congestion Control

A client participating in a non-linear media stream using CSA must manage the speed at which data arrives over the network through a process known as *congestion control*. Within the CSA framework, this is done by managing the size of the ACS, noted as $|ACS|$.

Under the channel-based transmission scheme, the server offers a large set of constant bit-rate channels, G. Clients subscribe to a subset of this offering, so that $ACS \subset G$. Because each channel $g_i \in ACS$ is offered at a constant bit-rate, the overall bit-rate of the arriving ACS is determined by the size of the set, or $|ACS|$. The congestion control problem for CSA is analogous to the problem faced in Receiver-Driven Layered Multicast (McCanne et al., 1996), and we apply a similar solution.

At runtime, the client adjusts the size of the ACS through subscribe and unsubscribe operations At signs of network congestion, such as the detection of lost packets, the client decreases $|ACS|$ through an unsubscribe operation. In order to maintain the most useful data flow after the decrease in subscription level, a client will choose to unsubscribe from the least useful active channel. The UCR metric described in Section 4.7 is utilized for this evaluation. The UCR metric combines application-specific utility and cost functions to determine how best to adapt a multimedia dataset.

In times of exceptionally strong network performance, the client probes for additional bandwidth by increasing $|ACS|$ through a subscribe operation. Once again, the UCR metric is used to determine which channel should be added to the ACS.

A series of timers are used for each level of subscription to improve stability and to allow the system to converge more quickly to an appropriate subscription level. A simplified version of the congestion control algorithm is shown in lines 2-9 of Figure 7.3.

Content Control

Parallel to performing congestion control, each client must also perform content control. This task is unique to the problem of non-linear streaming. In traditional linear media applications, data is delivered in a fixed order and there is no freedom to change the

```
1   repeat forever:
2     if ((experiencingNetworkLoss) and (timerExpired))
3       c_active = GetLeastUsefulActiveCluster(RG, ACS)
4       g_active = M(c_active)
5       Unsub(g_active, ACS)
6     else if ((notExperiencingNetworkLoss) and (timerExpired))
7       c_inactive = GetMostUsefulInactiveCluster(RG, ACS)
8       g_inactive = M(c_inactive)
9       Sub(g_inactive, ACS)
10    else
11      c_inactive = GetMostUsefulInactiveCluster(RG, ACS)
12      c_active = GetLeastUsefulActiveCluster(RG, ACS)
13      if Utility{c_inactive} > Utility{c_active}
14        g_inactive = M(c_inactive)
15        g_active = M(c_active)
16        Unsub(g_active, ACS)
17        Sub(g_inactive, ACS)
18      endif
19    endif
```

Figure 7.3: A simplified version of the CSA client-driven adaptation algorithm. To perform congestion control, CSA iteratively monitors network conditions to determine if it should contract the ACS (lines 2-5), expand the ACS (lines 6-9), or maintain a constant size. When the ACS size does not change, CSA perform content control by ensuring that the most useful clusters always map to the set of active channels (lines 11-18).

order to meet application needs. This obviates the need for content control over the arriving data stream. However, individualized control over the contents of an arriving data stream is a primary requirement for non-linear media streaming applications.

Within the CSA framework, Content control is performed by aggressively changing channels over time, managing the ACS to ensure that the active channels match the current application requirements. Recall that the RG data representation abstraction builds clusters that are semantically consistent. As a result, each channel has an associated semantic meaning. This allows the adaptation algorithm to use channel subscription operations to express an application's needs for specific semantically meaningful units of data.

At runtime, a client iteratively compares the least useful active channel, g_{active}, with the most useful inactive channel, $g_{inactive}$. Whenever it is discovered that the utility of $g_{inactive}$ is greater than that of g_{active}, the two swap positions and $g_{inactive}$ becomes a

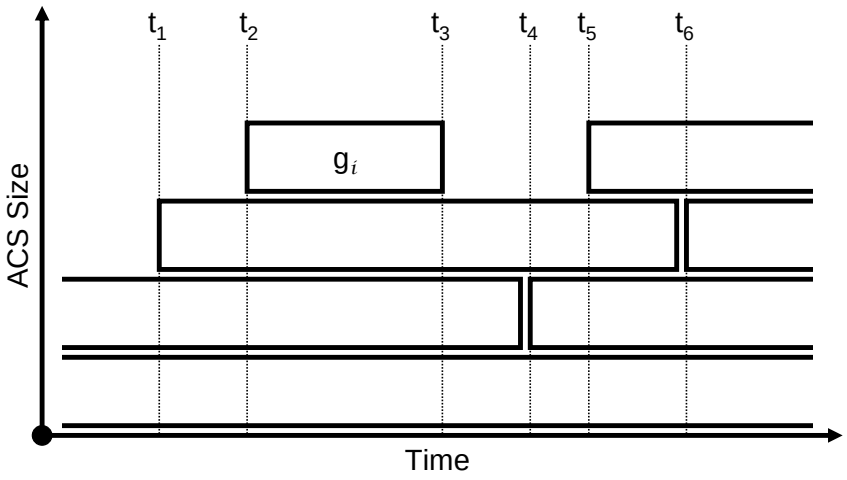

Figure 7.4: This figure illustrates the evolution of the ACS over time. The client-driven adaptation algorithm composes a custom flow of data to match application needs by a series of subscribe and unsubscribe operations. The composite of these channel operations yields a flow of data that is unique to the individual client. This figure illustrates a channel plot that shows a series adaptive operations over a window of time. Times t_1, t_2, and t_5 mark subscriptions to expand the ACS. Time t_3 marks an unsubscribe operation to contract the ACS. Times t_4 and t_6 mark channel changes performed solely for content control.

member of the ACS. Lines 11-18 of Figure 7.3 show a simplified version of the algorithm.

The channel subscription pattern is driven by the evaluation of utility that is performed on each iteration of the adaptation algorithm. The content control logic uses the same UCR metric as the congestion control algorithm for determining the relative utility of each channel.

The UCR metric is a spatial measure of utility defined on the RG structure included in the CSA index. Most importantly, it evaluates utility with respect to the current application conditions and preferences as expressed through both the prediction and alpha vectors. As a result, the sequence of subscription operations performed in the adaptation process is determined uniquely on each client in response to user interactions and locally changing system conditions.

Each client will exhibit their own pattern of subscription requests based upon their own local needs. For example, Figure 7.4 illustrates a possible sequence of subscription operations over a small window of time. In the figure, the ACS starts at size two and grows to size four with the subscriptions at times t_1, and t_2. At time t_3, the congestion control algorithm determines that the ACS is too large and the ACS is contracted back

down to size three. The content control algorithm initiates a channel swap at time t_4. This is followed by another subscription to enlarge in the ACS (at t_5) and another channel swap (at t_6).

The concatenation of data flows, following a series of subscribe and unsubscribe operations, produces a unique flow of data that is delivered to each individual client. The *number of channels* within the ACS is controlled to perform congestion control. Given an ACS of size n, the *selection of channels* in the ACS is controlled to perform content control.

When CSA is employed together with a broadcast or multicast network model, the unique flow represented by the concatenation of the subscribed channels is composed without any direct communication between clients and the server. As a result, CSA can deliver unique, customized data flows to individual clients in fully scalable manner.

Implications of Using Non-Uniform Bit-Rate Allocation

In the content and congestion control algorithms, the fundamental operations for expressing adaptive behavior is adding or removing individual channels from the ACS. The algorithm is relatively straightforward when all channels are transmitted at identical bit-rates and, in practice, such a configuration is often the best approach.

It is possible, however, that the data rate for each channel can be set individually. In this case, the adaptation algorithms will need to be altered slightly to take this into account. Under this scenario, the adaptation problem becomes an instance of the classic Knapsack Problem which is known to be NP-complete (Garey and Johnson, 1979). As a result, the adaptation algorithms would need to be altered to use a heuristic-based or approximation technique to choose which channels to add or remove from the CSA.

7.3 Experimental Testbed and Methodology

The effectiveness of the CSA architecture in supporting scalable delivery of non-linear media streams has been evaluated via a series of experiments. This section describes the experimental testbed and methodology employed throughout the evaluation.

7.3.1 Prototype Application

The experiments presented in this chapter utilize the same prototype application (see Section 6.1 as the TCP-based evaluation covered in Section 6.3. Briefly, the prototype is

an network streaming implementation of the SOI algorithm for reconstructing digitized spaces (Aliaga et al., 2002).

When distributed to a large audience, such an application could allow digital museums to share digitized versions of famous places with vast audiences. Each member of such an audience would be able to freely navigate through the recreated space along an independent path and at their own rate. As the results to be presented later in this chapter will highlight, this vision can be enabled using the CSA architecture.

7.3.2 Experimental Testbed

As in the TCP-based evaluation of GAL described in Section 6.3, the CSA experiments covered here were performed on the Emulab network emulation testbed (White et al., 2002). Once again, the Emulab testbed provided a middle ground between the realism of the actual Internet and the laboratory control over network simulations.

Using an emulation tool such as Emulab allows for evaluation under fairly realistic network conditions because it employs actual network hardware and client computers sending real data packets. At the same time, experiments are performed in a closed environment where variable factors such as competing network traffic can be carefully controlled.

7.3.3 The SUM Performance Metric

For all of the experiments in this chapter, it is critical to measure how well the system performs under specific conditions. As in the TCP-based evaluation of GAL (Section 6.3, the CSA experiments use the SUM metric to quantify performance. The SUM metric is a performance metric applicable to any GAL-based application.

The SUM is defined as a function upon the RG maintained within the application-independent adaptation layer. It can therefore evaluate how well the communication layer supports an application's adaptation needs given specific cost and utility metrics. Relative comparisons of network behavior can be made by changing the underlying communication model's parameters and evaluating the SUM metric.

For a more detailed description of the SUM metric, refer to Section 6.2.4.

7.3.4 Network Models

The CSA architecture exploits a channel-based network model to provide scalable non-linear media streaming. Two classes of channel-based protocols have been developed: (1) broadcast and (2) multicast.

Broadcast networks transmit the same information to all receivers irregardless of individual user preferences. Once data arrives at a receiver, the receiver can choose to ignore it. However, data is sent by the broadcast network to every individual receiver.

Multicast, in contrast, allows receivers to individually opt in to receive a flow of information. Without opting in, a receiver will not receive the flow of data. Only after specifically subscribing to a particular flow will data begin to arrive at the receiver.

In many ways, broadcast is an idealized multicast model. It has no group management duties because all data gets sent to all receivers. Similarly, there are negligent subscription delays as receivers tune in to individual channels because the data is always arriving irregardless of the receivers' interests.

However, broadcast data networks are not practical for wide-area distribution of data because of the finite bandwidth available over network links. If literally every data packet on the Internet were sent to every user, the resulting congestion would cause the Internet to quickly grind to a halt.

Given the contrast between broadcast and multicast, the experiments in this chapter evaluate the performance of CSA under both models. Broadcast-based CSA measures CSA performance under idealized conditions where the overheads associated with channel management are negligible. The multicast-based CSA experiments provide a better look at how CSA performs using today's most practical channel-based protocols. The specific network topologies utilized by each CSA variant will be presented together with their experimental results in the sections that follow.

7.4 Performance Evaluation using Broadcast-based CSA

This section highlights several experiments performed using a broadcast-based network model in support of CSA. Because broadcast networks send all data to all receivers, there is no overhead required for group-management. This allows for individual receivers to subscribe or unsubscribe to individual channels of data without any latency, without any additional network traffic, and without ever contacting the centralized

server.

For these reasons, the performance of the broadcast-based CSA prototype can be considered the ideal level of performance achievable by CSA. Later in this chapter, these results will be compared to more practical solutions based on today's technology to highlight where improvements can be made.

This section begins with a detailed description of the network topology employed for the broadcast-based CSA experiments. It continues with a detailed discussion of the prototype's performance at a variety of scales.

7.4.1 Broadcast Network Topology

The broadcast network topology is designed to test CSA's ideal scaling behavior when the performance overhead for group management is negligible. The network topology used for the broadcast experiments is the simplest used in this dissertation.

For an experiment with n clients, $n + 1$ nodes were connected to a single local area network with all links provisioned at 100Mbps. The one additional node corresponds to the single SWIM server tasked with transmitting the channels of data to the other n receivers. An illustration of a sample topology from the broadcast experiments is shown in Figure 7.5.

7.4.2 Ideal Scalability of CSA

One of the primary motivations for the CSA framework is the ability to support large groups of independent users. This section presents the results from a series of experiments designed to evaluate the performance of CSA at a range of group sizes. The evaluation presented here compares the performance of broadcast-based CSA with the TCP-based GAL performance reported in Section 6.3.6.

In the TCP-based evaluation from Chapter 6, performance was measured for group sizes between one and sixty-five clients accessing a dataset modeled by a 15,568 cluster RG. The TCP-based experiments performed well for small user groups, but performance degraded past 20 users which marks the saturation point of the first-mile link. From that point onward, performance continued to drop as additional clients joined the session.

In this evaluation, a similar series of experiments were run with between one and 65 clients. These experiments were performed using the broadcast network topology and

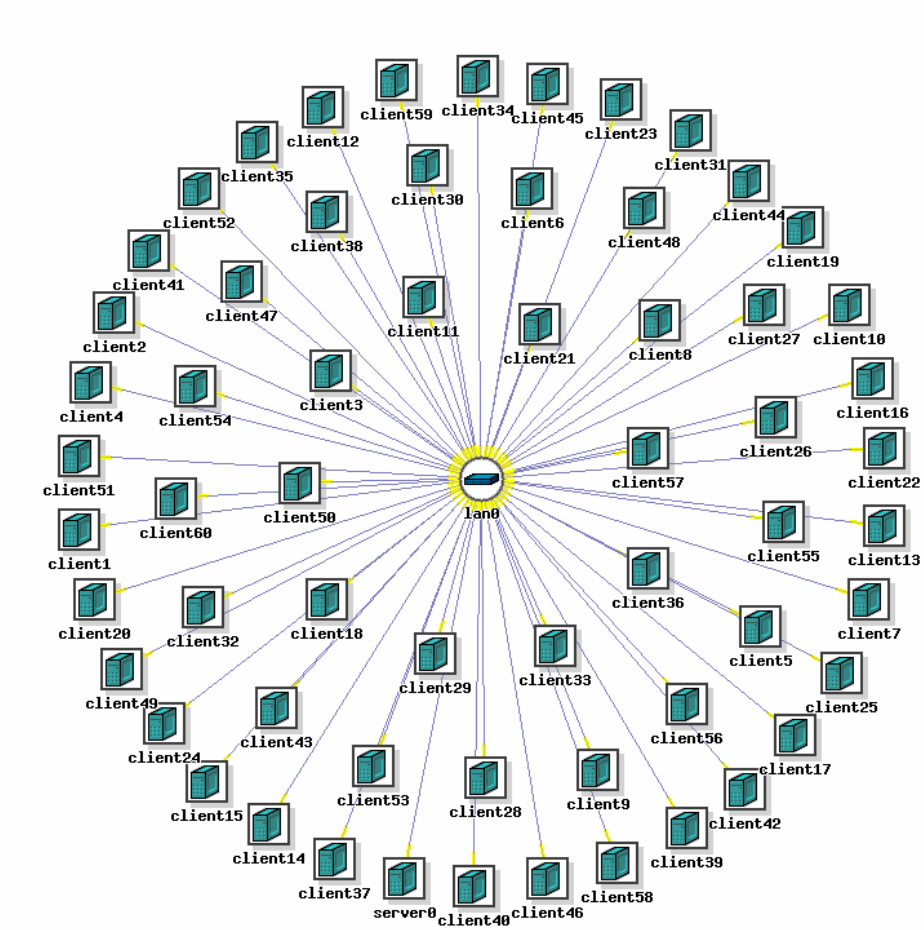

Figure 7.5: A broadcast-based experiment topology with 60 clients is depicted in this image taken from the Emulab testbed's web interface. The topology shows 61 nodes (60 clients and 1 server) arranged within a single local area network.

with a RG containing 160 clusters. Besides the assignment of edges to clusters, the RG used in these experiments was identical to one used in the TCP-base GAL evaluation.

The similarity between the two RG models means that the experiments performed here and those performed previously both deal with the exact same dataset in terms of utility space, nodes, and edges. The only difference is the granularity of data access (i.e. the number of clusters).

The results of both the broadcast-based CSA and the previous TCP GAL evaluation are shown in Figure 7.6. There are several critical elements to observe.

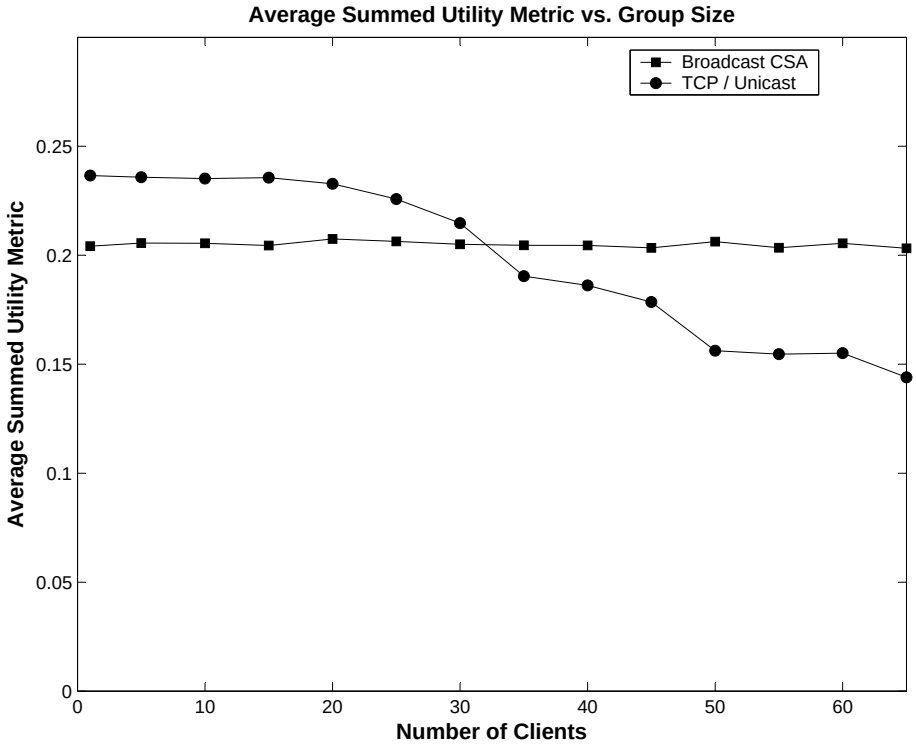

Figure 7.6: The broadcast-based CSA prototype performs at a constant level regardless of the number of users. This compares favorably to the TCP-based GAL approach for non-linear media streaming. Past the crossover point at $n \approx 32$, broadcast-based CSA clearly outperforms the unicast alternative.

The Cluster Penalty

First, notice the superiority in performance of the TCP-based GAL experiment for small user groups. When the number of clients n is below 20, the TCP-based system outperforms broadcast-based CSA by roughly 14%. The margin in performance for the TCP-based experiment is directly attributable to the two orders of magnitude difference in the number of clusters.

In the TCP experiments, which are based on the unicast network model, individual clients are given the greatest degree of freedom in data access because they are able to make individual requests from the server using 15,568 clusters. In the broadcast-based CSA experiments, a channel-based network model is used making the large cluster count prohibitive.

The experiments presented here reduce the number of clusters by two orders of magnitude to 160. As highlighted in Section 6.3.5, the reduction in flexibility due to clustering negatively impacts performance via the cluster penalty. However, the cluster

penalty's impact is relatively small until the clustering becomes extreme.

CSA aggressively employs clustering as part of a strategy to support large groups of users. However, at small group sizes, where the TCP-based approach proved effective, CSA under-performs by roughly 14% due to the cluster penalty.

Immunity to Scale

Broadcast-based CSA essentially transmits the entire dataset all of the time. It is up to the individual clients to filter out the data that they are interested in based on the channel structure outlined in the CSA Index. This simple mechanism allows for an infinitely scalable mechanism for distributing non-linear media. As illustrated in Figure 7.6, client performance remains constant regardless of the number of simultaneous clients. The performance metrics with group sizes of one and sixty-five are essentially identical.

While CSA under-performs for small groups due to the cluster penalty, it dramatically outperforms TCP-based GAL for large groups of users. The performance under scale of the two experiments highlights two key points in the relationship between the two distribution methods.

The first key point is the *unicast saturation point*. As illustrated in the TCP-based GAL experiment, unicast performance remains constant until the first-mile link becomes saturated due to the growing outbound data rate. Up until this point, CSA under-performs by the fixed cluster penalty. This highlights that the fine-grained data access provided by the TCP-based approach is the best solution for small group sizes.

The second key point is the *crossover point*, where unicast performance drops below the scalable CSA approach. In the experiments presented here, the crossover point is at $n \approx 32$. The exact location of the crossover point depends on several parameters of the experiment.

Between the saturation point and the crossover point, the TCP-based GAL approach continues perform better than CSA. However, the margin continuously decreases as congestion further constrains the saturated first-mile link. The margin drops to zero at the crossover point as the drop in performance due to congestion equals the cluster penalty.

Past the saturation point, the TCP-based GAL approach continues to drop in performance. However, the CSA approach is immune to scale and continues to perform at the same constant level. As the group size increases, the CSA approach outperforms by a greater and greater margin. This highlights that the channel-based architecture

of CSA is far superior for large user groups.

7.4.3 A CSA Performance Model

A simple CSA performance model can be developed given these early results. First, assume that the TCP-based GAL prototype with a single client defines a baseline for ideal performance. The exact performance level depends directly on the last-mile bottleneck link bandwidth.

The performance of the unicast prototype can be noted by $UnicastPerf(n, B_{lm})$, where n is the number of clients and B_{lm} is the bandwidth of the last-mile link. Using this notation, the ideal performance can be noted as a function of the last-mile link bandwidth as in Equation 7.3.

$$IdealPerf(B_{lm}) = UnicastPerf(1, B_{lm}) \tag{7.3}$$

As the results presented in this section show, the broadcast-based CSA prototype performs equally well for all group sizes. However, the performance suffers from the cluster penalty. Therefore, the performance for broadcast-based CSA is equal to the ideal performance benchmark minus the cluster penalty, noted as CP. Notice that Equation 7.4 is independent of the number of clients, n.

$$BroadcastCSAPerf(n, B_{lm}) = IdealPerf(B_{lm}) - CP \tag{7.4}$$

Under this model, the crossover point is defined as the number of clients n where $UnicastPerf$ is equal to $BroadcastCSAPerf$ as defined in Equation 7.5.

$$CrossoverPt(B_{lm}) = n|(UnicastPerf(n, B_{lm}) = BroadcastCSAPerf(n, B_{lm})) \tag{7.5}$$

An important feature of Equation 7.5 is that the crossover point is a function of the last-mile link bandwidth. This is because the saturation point at which the unicast performance begins to decline is directly related to the average client-side bandwidth requirements for the group of receivers.

7.5 Performance Evaluation using Multicast-based CSA

The broadcast-based CSA prototype performs very well in supporting non-linear media streaming to large user groups. However, broadcast delivery is not a realistic solution except for dedicated distribution networks such as cable television networks or over-the-air transmission. A more practical solution for streaming over the Internet requires a multicast solution for CSA.

This section explores the performance of multicast-based CSA in comparison to both the unicast GAL and broadcast-based CSA approaches covered so far. In addition, several experiments are performed that explore the performance impacts of various overheads associated with multicast as well as several engineering parameters.

7.5.1 Multicast Network Topology

The network topology specified for the multicast experiments presented in this section is identical to the topology used for the TCP-based GAL evaluation in Chapter 6. This subsection provides a brief overview of the topology. Refer to Section 6.2.3 for a more detailed description.

The topology defines a single image server provisioned with a 100Mbps network first-mile network connection. This corresponds to the maximum network speed supported by the Emulab testbed. Last-mile bottleneck links are provisioned independently for each client with a bit-rate of 5Mbps. Interior links are provisioned with the same 100Mbps as the server, ensuring that bottlenecks occur only at the topology's last-mile links.

The topology links the server to a large set of clients via interior routers which are connected via a tertiary tree. This topology is defined to maximize the possible group size in the experiments by using the Emulab testbed's resources as efficiently as possible. An illustration of the tertiary tree topology with 60 clients can be found in Figure 6.5.

The highly regular tertiary branching pattern and balanced nature of the topology, which is required to enable evaluation using large group sizes, is a limitation of the Emulab testbed environment. However, its impact on the overall evaluation should be negligible. For example, while highly unbalanced trees may result in greater variation in join and leave latencies, I have been careful to explore a wide range of such overheads

and discuss their impact on overall performance.

7.5.2 Practical Scalability of CSA

The multicast-based CSA prototype is designed to provide a practical realization of the scalable behavior exhibited by the broadcast-based solution evaluated in Section 7.4.2. This section compares the results from a set of experiments evaluating multicast-based CSA with both the broadcast and unicast results presented previously.

For the multicast-based CSA experiments, a RG with 160 clusters was used. Each cluster was transmitted over a multicast channel at a rate of 480Kbps. The number of clients ranged from one to 65. These parameters match up exactly with the broadcast-based CSA experiments. Therefore the results can be compared directly.

The multicast-based CSA experiments were built on top of a lab-based infrastructure multicast algorithm that imitates many properties of IP Multicast with the added feature of allowing explicit variation of the overhead parameters such as join and leave latencies. This feature is used to perform several of the evaluations presented later in this chapter. For the scaling experiments presented here, the overheads were set to values similar to IP Multicast.

The results from the multicast-based CSA experiments, together with the broadcast-based CSA and TCP GAL results, are presented in Figure 7.7. There are several observations to be made about the results.

First, the multicast-based CSA prototype performs equally well for all group sizes. Just as with the broadcast-based CSA prototype, performance was flat all the way through to the maximum group size of 65. This compares favorably to the unicast approach of the TCP-based GAL prototype of Chapter 6. As with the broadcast-based prototype, there is a crossover point after which the performance of CSA outperforms the TCP-based prototype.

However, the multicast-based CSA prototype performs worse than the broadcast-based prototype across the entire range of group sizes. The drop in performance is a direct result of the group management overhead assocaited with the multicast protocol.

Recall that a broadcast network has virtually no overhead in managing groups of receivers. Multicast, however, must maintain state at each of the internal routers to properly route and replicate the individual data flows. This group management task is decentralized to scale in support of large user groups. Unfortunately, the group management process delays responses to **Sub** and **Unub** operations. It is this delay

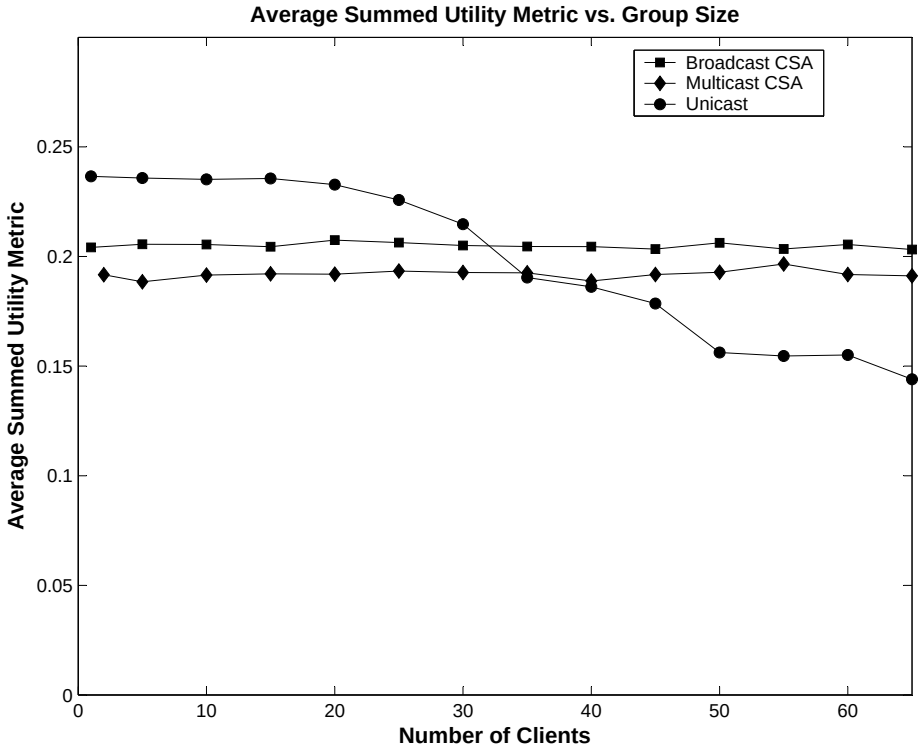

Figure 7.7: The multicast-based CSA prototype performs at a constant level regardless of the number of users. This behavior is similar to the broadcast-based CSA prototype and compares favorably to the TCP-based GAL approach. The multicast solution performs slightly worse than broadcast-based CSA due to the overhead associated with group management. However, the multicast solution is more practical to deploy.

that is responsible for the drop in performance between multicast and broadcast CSA.

This result can be used to extend the performance model of Section 7.4.3. Given a group management overhead of MO, multicast-based CSA performs as governed by Equation 7.6. The MO value corresponds directly to the constant level of difference in performance indicated in Figure 7.7.

$$MulticastCSAPerf(n, B_{lm}) = IdealPerf(B_{lm}) - CP - MO \tag{7.6}$$

The exact value of MO varies by multicast protocol. IP Multicast, for example, has leave latencies that average around three seconds. Various application-layer multicast algorithms exhibit a range of overheads. In general, the faster the response to join and leave operations, the better a multicast protocol will support CSA. The impact of these factors is discussed in more detail later in this chapter

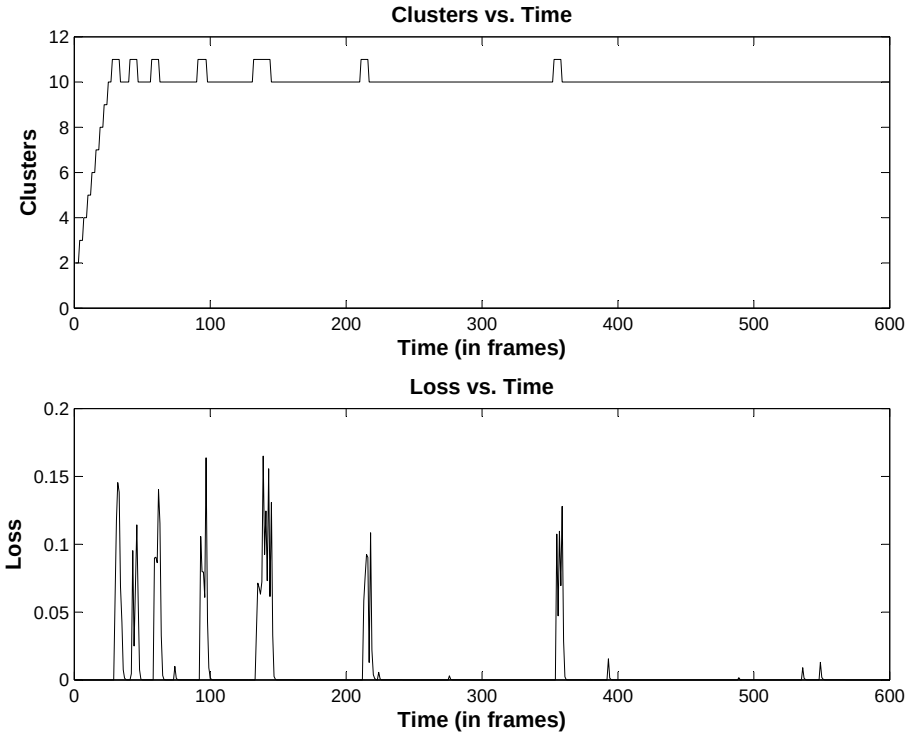

Figure 7.8: CSA achieves congestion control by adjusting the size of the ACS. The top plot shows the number of active clusters over the course of a session. The lower plot shows the estimated packet loss rate for the same session. CSA probes for additional bandwidth by increasing ACS size. When loss is detected as a result, the ACS size is decreased. Notice the seven spikes in loss and the corresponding drop in the number of active clusters.

7.5.3 CSA Congestion Control

The CSA framework performs congestion control by adjusting the number of actively subscribed channels. Under CSA, a server transmits a set of channels all at fixed bit-rates. Therefore, a client can reduce the bit-rate at which data arrives by either adding or removing channels from their ACS. The algorithm is described in more detail in Section 7.2.3.

In this experiment, a single client was evaluated while running a ten minute session over a multicast network. Throughout the session, a log was kept of both the size of the ACS and the estimated packet loss rate. The results are illustrated in Figure 7.8.

The top plot of Figure 7.8 shows the number of actively subscribed clusters. This is equivalent to the size of the ACS. At the very start of the session, the size of the ACS is quickly expanded as the client attempts to use as much of the available bandwidth as possible. The expansion stage continues until the estimated loss rate rises above a

threshold.

The lower plot of Figure 7.8 shows the estimated loss rate over the course of the same ten minute session. The first spike in the plot corresponds to the end of the initial ACS expansion period. In response to the first spike in the loss rate, the congestion control algorithm decreases the size of the ACS. The actual trigger is a sustained loss rate greater than a threshold.

The triggers for both increases and decreases in the size of the ACS are governed by timers which guide the congestion control toward stability. This is evidenced by the decreased rate of bandwidth probing over the life of the session. In Figure 7.8, the effect of the timers can be seen by the increased spacing between the intermittent probes to 11 clusters.

7.5.4 Congestion Control with Cross Traffic

Congestion control must adjust the data rate in response to transient competing data flows as well as long term changes in available bandwidth. The previous experiment highlighted the congestion control's behavior during self-interference as it expanded to utilize the entire last-mile link. In this section, the congestion control algorithm is observed during its response to a variable load of cross traffic.

The experiment in this section evaluates the performance of the congestion control algorithm in the face competing TCP traffic. The experiment follows the size of the ACS and the estimated loss rate through a ten minute session. The session begins with no competing traffic over its bottleneck link. After two and a half minutes ($t = 150$), a 180 second load of simulated HTTP traffic is introduced over the congested link. At $t = 330$, the HTTP traffic ceases. The results are shown in Figure 7.9.

In the first 30 seconds, the size of the ACS quickly increases as the client performs its initial probe for available bandwidth. At $t \approx 30$, the ACS grows to size 11 and congests the bottleneck link. The increase in ACS size is matched by a spike in the loss rate estimate. As a result, the congestion control algorithm backs the ACS down to size 10. From $t = 30$ to $t = 150$, the client continues to probe for additional bandwidth, but at growing intervals as the timer duration increases.

At $t = 150$, the competing HTTP traffic begins flowing over the bottleneck link and the measured loss rate begins to climb. Typically, the client would back down extremely fast in response to the increased loss rate. However, in this case, as shown in Figure 7.9, the client initially hesitates to back down from $|ACS| = 10$. The delayed response

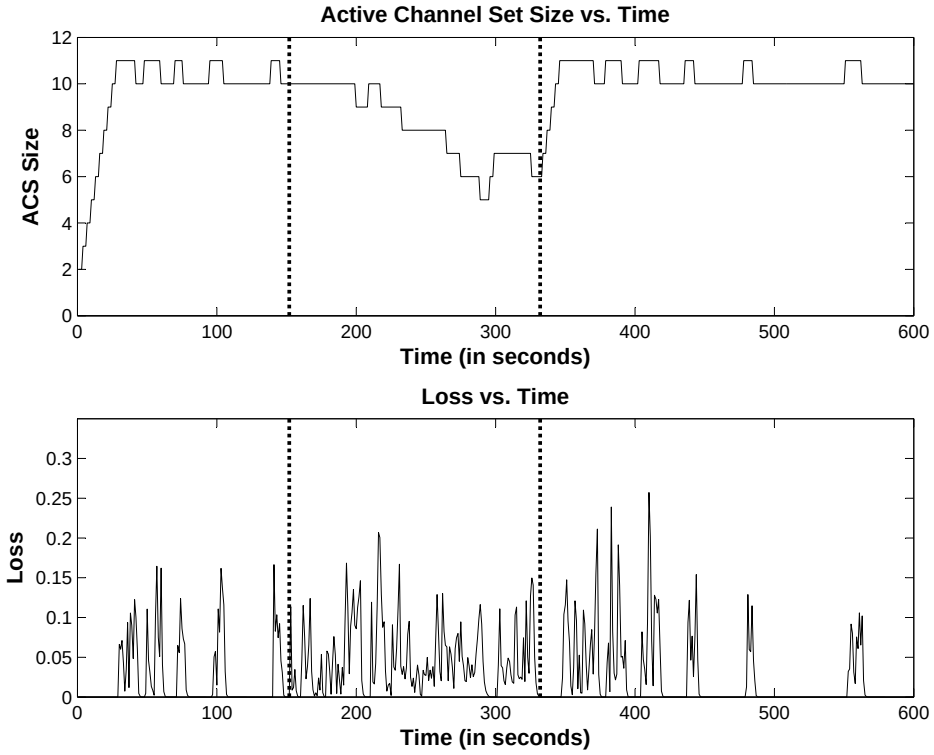

Figure 7.9: At the start of a session, the client increases the ACS size until size ten, when it detects high packet loss rates. The ACS size remains stable until a blast of HTTP cross traffic causes the system to back off (the region between the dotted lines). When the cross traffic fades, the client once again increases its subscription level to the steady state of ten.

is due to the fact that the onset of competing traffic occurred nearly simultaneously with a decrease in ACS size.

After detecting that the loss rates remained steady, the client continued to back down, with the ACS size falling to as low as five. At $t = 330$, the HTTP traffic was removed from the bottleneck link and the client detected an improvement in network conditions. Very rapidly, the ACS size was increased to 10 following the same probing pattern as seen at the start of the session.

The timers used to govern the rate of increase and decrease in ACS size are tunable and can be configured to yield faster back-off times at the expense of lower stability. The specific settings for the timer parameters should be chosen to best match a particular application. For example, stability is less critical for the IBR prototype application than it is for typical video streaming applications. The prototype therefore has its timer parameters set to adapt more quickly to changes in network congestion.

7.5.5 Impact of Channel Cycle Size on Performance

Among the many CSA engineering parameters that must be configured by an application developer is the channel cycle size. When partitioning the edges of a RG into clusters, not only must the number of clusters be determined, but so must the size of each cluster. For example, all clusters could be configured to contain the same amount of data. Conversely, the clusters could be specified so that there is a great degree of variability in the cluster size.

The amount of data in a cluster has a great impact on CSA performance. CSA transmits the data for an individual cluster according to a round-robin schedule, where every byte is sent out over the channel before starting once again from the beginning.

The time required to send out all bytes over the channel is the *cycle time*. For clusters with little data assigned to them, the cycle time can be very short. For larger clusters the cycle time can be very long. The cycle time is critical because it determines the expected time required to resolve a node in the RG. At runtime, a client that subscribes to a channel can begin receiving the data for the associated cluster at any point in the cycle. The expected time to resolve a specific node is one half the cycle time. Therefore the expected access time to data in large clusters is much longer than the expected access time to data in small clusters.

In practice, the proper configuration of edges to clusters depends highly on the application. Certain applications may require perform best with relatively even access times across clusters. Other applications may require very fast access to certain data elements and would therefore perform better using variably sized clusters.

In this section, the multicast-based CSA prototype is evaluated using two different clustering configurations. Both configurations utilize a similar RG. Both have the same utility space, nodes, and edges. However, in one data set the clusters are set to be equal size. In the other data set the clusters are of variable size. The results are shown in Figure 7.10.

The two plots in Figure 7.10 show the cumulative SUM value over the life of each of the two sessions. The rational for using the cumulative SUM metric as well as its formal definition is described in more detail in Section 6.3.3.

The plots indicate that for the prototype application, equally sized clusters have superior performance when compared to variably sized clusters. Recall that when variably sized clusters, some data elements have faster access times while others require additional time. The results in this experiment indicate that the benefit of faster access

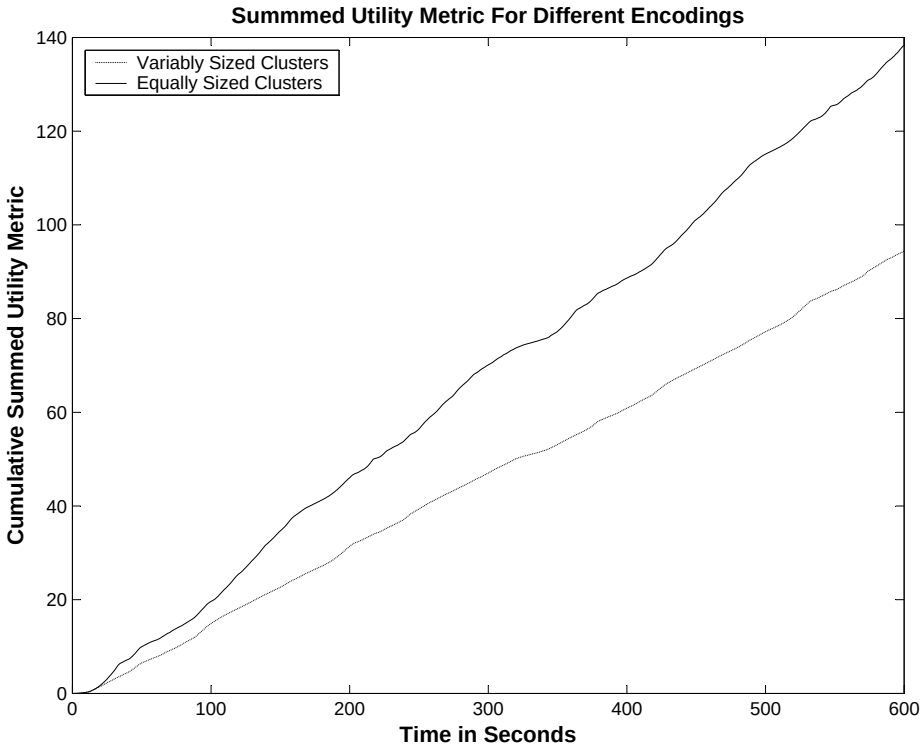

Figure 7.10: The amount of data assigned to a cluster has a direct impact on system performance. Variably sized clusters can provide faster access to certain data at the cost of longer access times for other data. Equally sized clusters performed best for the prototype application, though other applications may behave differently.

to certain data is not worth the penalty of longer access times to the remaining data.

In general, the impact of cluster size on performance is highly application dependent and careful experimentation should be performed to determine the optimal configuration. However, this result highlights that the allocation of edges to clusters does indeed have a sharp impact on overall performance.

7.5.6 Impact of Leave Latency on Performance

The CSA adaptation algorithm uses subscription operations to perform both content and congestion control. Any significant latency between the issuance of a subscription operation and the actual effect on transmission can have dramatic impact on overall performance. This section presents the results from two sets of experiments exploring the impact of leave latency on CSA performance.

In this section, I concentrate on leave latency in particular in this section because IP Multicast, the most widely accepted multicast protocol, exhibits far greater latencies

for leave operations than for join operations. The impact of symmetric join and leave latencies is discussed in Section 7.5.7.

Performance Impact of Various Leave Latency Values

Various multicast implementations (e.g., IP Multicast, ALM protocols, etc.) exhibit a wide range in *leave latency*: the time it takes between an unsubscribe request and the actual termination of the data flow. For example, in the course of performing the experiment in this dissertation, IP Multicast typically showed an average leave latency of about three seconds. Depending on their design, ALM protocols can be significantly better or worse.

The experiments in this section are designed to evaluate the impact of leave latency on CSA performance by introducing artificial leave latencies from 0 to 5000 milliseconds. The results are shown in Figure 7.11. The experiment shows that longer latencies have a negative impact on performance. In particular, the three second leave latency measured in our IP Multicast experiments is far from the ideal range for supporting CSA.

The results show a steep drop in performance at between two and three seconds of leave latency. The overall trend in performance is important. However, the exact slope of the drop is highly dependent on fraction of time spent on overhead and depends on the specific parameters of the experiment.

The system parameters along with user behavior indirectly set the expected *listen time*: the average time for which a client receives any single channel before unsubscribing. Together with the expected listen time, the average join and leave latencies of the underlying multicast algorithm define the CSA *efficiency*.

$$Efficiency = 1 - \frac{SubOpLatency}{\langle ListenTime \rangle} \tag{7.7}$$

Efficiency is a measure of the fraction of time for which a channel subscription is actively receiving data. For latency-free multicast protocols, efficiency becomes one. As latency grows, efficiency drops. As the efficiency factor becomes higher, the overhead factor MO of the performance becomes lower. This maps directly to the ideal case of broadcast. Broadcast has an efficiency factor of 1, leading to zero overhead.

When the average duration for a single subscription is long, the inefficiency introduced by subscription operation latency is relatively small and the impact on performance will be lower. Conversely, if the average subscription duration is short, the efficiency is large and can dramatically impact performance.

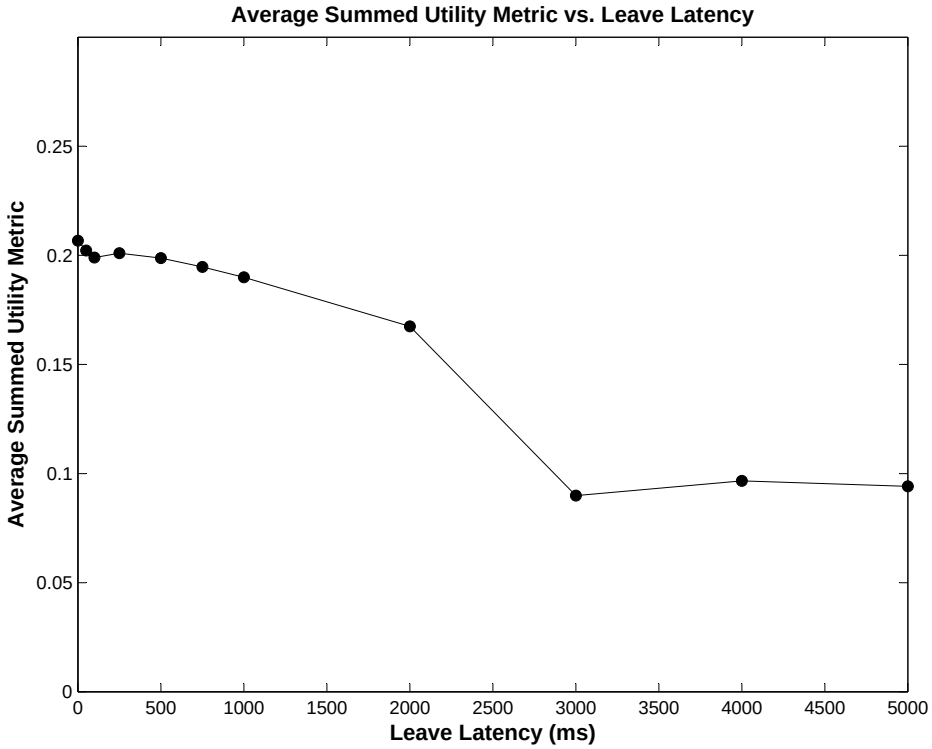

Figure 7.11: Increased leave latency results in lower CSA performance. Leave latency is a major contributor to the overhead cost of managing the ACS under multicast. As overhead increases, the efficiency of data delivery decreases, dragging down overall performance.

In other research, I have been participating in the development of StrandCast (Begnoche et al., 2005), a novel ALM algorithm that attempts to minimize leave latency, and therefore maximize efficiency, while supporting a high rate of subscription operations. In the future, StrandCast may serve as an ideal multicast protocol to support CSA.

Leave Latency Impact on Congestion Control

Long leave latencies impact more than just CSA efficiency. It can cause serious problems with the congestion control algorithm as well. This is particularly true for multicast algorithms that don't provide explicit confirmation of **Unsub** operations. IP Multicast, for example, does not provide any feedback that an unsubscription request has been honored. The only method for determining that an unsubscribe operation has been handled is the eventual cessation of the data flow.

Figure 7.12 shows the results of an evaluation of the congestion control algorithm

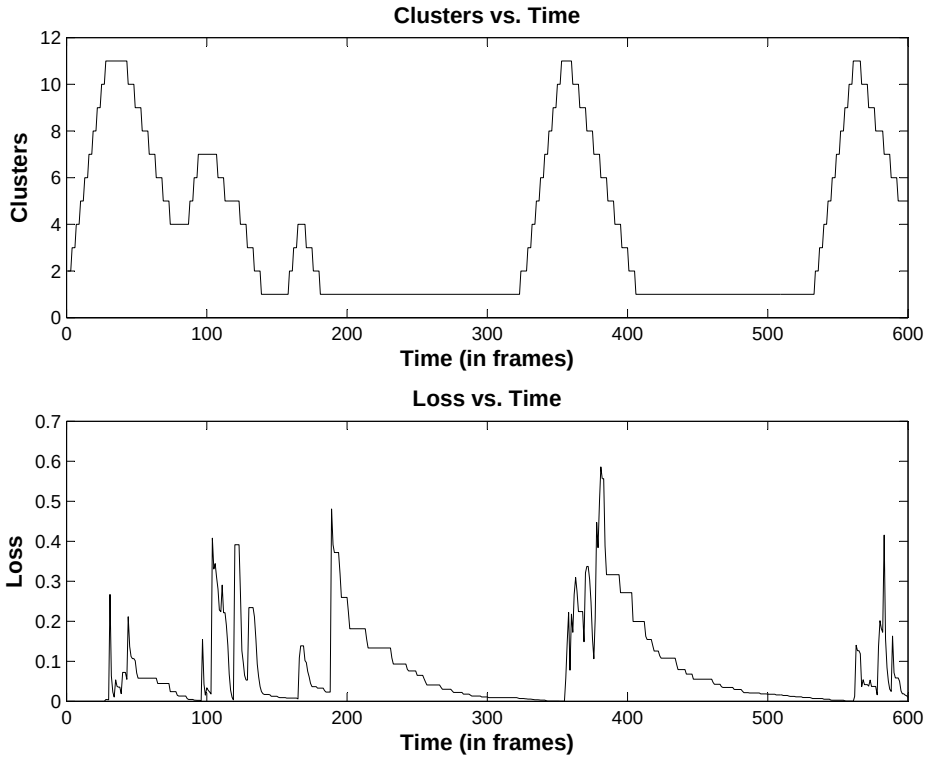

Figure 7.12: Excessive leave latency values can play havoc with the congestion control algorithm. Because **Unsub** operations do not have a timely effect on the loss rate, the congestion control algorithm backs off too aggressively. This leads to a dramatic fluctuation of the ACS size.

using a multicast algorithm with an average leave latency of four seconds. Similar to the properly behaving congestion control results of Figure 7.8, the session begins with a series of **Sub** operations to expand the ACS.

The trouble occurs at the first sign of loss. After the first **Unsub** operation, the congestion control algorithm waits for the loss rate to drop below the threshold. However, because the leave latency is so long, the loss rate remains high for several seconds. In the mean time, the algorithm has no way of ensuring that the **Unsub** operation has been honored. As a result, it issues a second **Unsub** request. The overly aggressive behavior continues throughout the life of the session resulting in the dramatic swings in ACS size.

The unstable behavior in ACS management is caused directly by the long leave latencies and unconfirmed **Unsub** operations. Better tuning of the timers for adjusting the ACS size can reduce the instability, but regardless of the setting long leave latencies negatively impact CSA performance.

7.5.7 Impact of Symmetric Join and Leave Latencies

CSA efficiency is determined in part by the performance of the underlying multicast network. In the previous section, results from leave latency experiments showed the deleterious impact of high latency values on performance. The results from that section focused on asymmetric subscription overheads where the leave latency was significantly different than the join latency. The results are particularly important because IP Multicast exhibits these asymmetric properties.

However, alternative multicast algorithms, such as some of the many ALM protocols, may show more symmetric overheads and provide explicit confirmation of join and leave events. In this section, the experiments aim to measure the impact of symmetric subscription overhead values on CSA performance. As the results in this section show, even these well behaved multicast algorithms introduce significant inefficiencies and contribute to a substantial value for the MO term in the performance model.

The experiments in this section were performed with a single client using multicast-based CSA. For each session, the average SUM was computed over a ten minute session to produce a quantitative measure of performance. Sessions were configured with various levels of subscription operation delay over the last-mile link. The delay values ranged from 0ms to 5000ms. The results are illustrated in Figure 7.13.

The plot shows a linearly decreasing trend in performance directly inversely related to the length of delay in responding to subscription operations. With a subscription delay of five seconds, performance was roughly 75% of the peak levels shown with no delay at all. Unlike the asymmetric case, where high latencies led to instability and significant drops in performance, CSA behaves much better with symmetric delays.

This interesting result highlights an important design factor that varies across different multicast algorithms. In IP Multicast, for example, only join latencies can be measured directly. This can be done by timing the delay between a **Sub** operation and the arrival of the first packet. However, no such direct method exists for the **Unsub** operation. Using IP Multicast, packets cease being forwarded to the application layer as soon as an **Unsub** operation is executed even though the flow of data packets continues down the wire for as much as several seconds. This results in only rough estimates of the leave latency based on estimated loss rates, leading to the unstable behavior highlighted in Section 7.5.6.

For multicast protocols with symmetric join and leave latencies, or for protocols that explicitly notify the application of when the flow of data has stopped, much improved

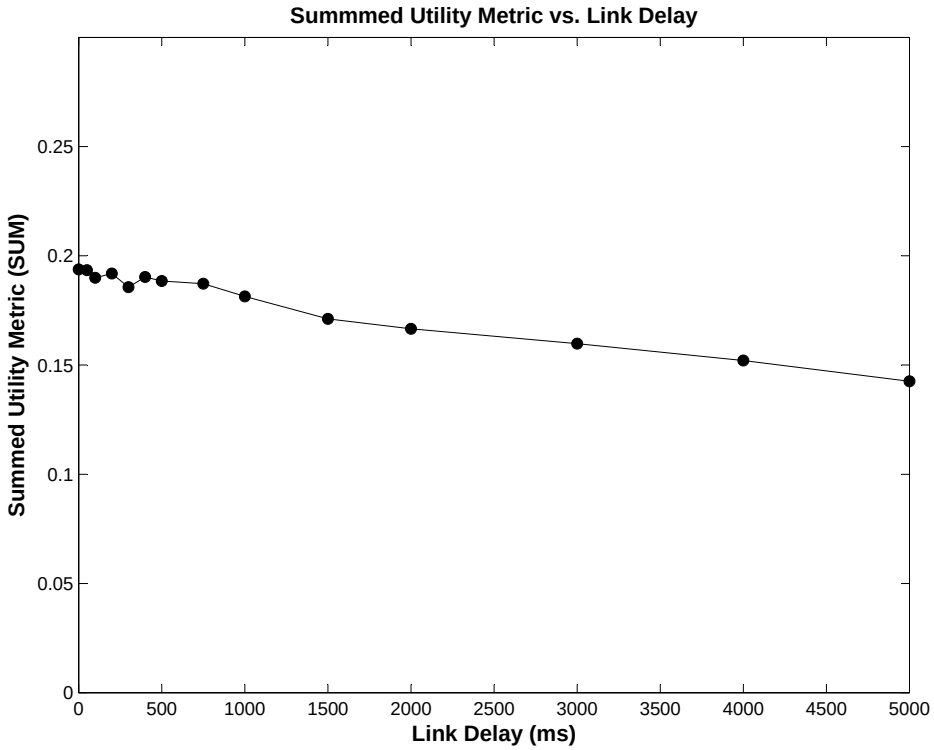

Figure 7.13: While IP Multicast exhibits asymmetric latencies for subscription operations, alternative multicast protocols such as ALM may have more symmetric delays. This figure shows the impact on CSA performance of symmetric delays for subscription operations. Long delays have a negative effect on performance, though to a lesser degree than asymmetric latencies.

results are possible. As shown by the experiments in this section, the actual drop in performance due to the delay in satisfying subscription operations is relatively small compared to drop in performance resulting from poor leave latency estimates in the congestion control algorithm.

7.5.8 Interaction Between Cluster Count and Subscription Latency

An important engineering parameter in designing a CSA application is the number of channels in the set G. Because G is mapped to the set of clusters C, the number of channels defines the number of clusters and therefore determines the granularity of access to the overall dataset.

A small size for G provides relatively few choices for adapting the ACS, reducing the ability of individual clients to customize their incoming data flow. Conversely, a

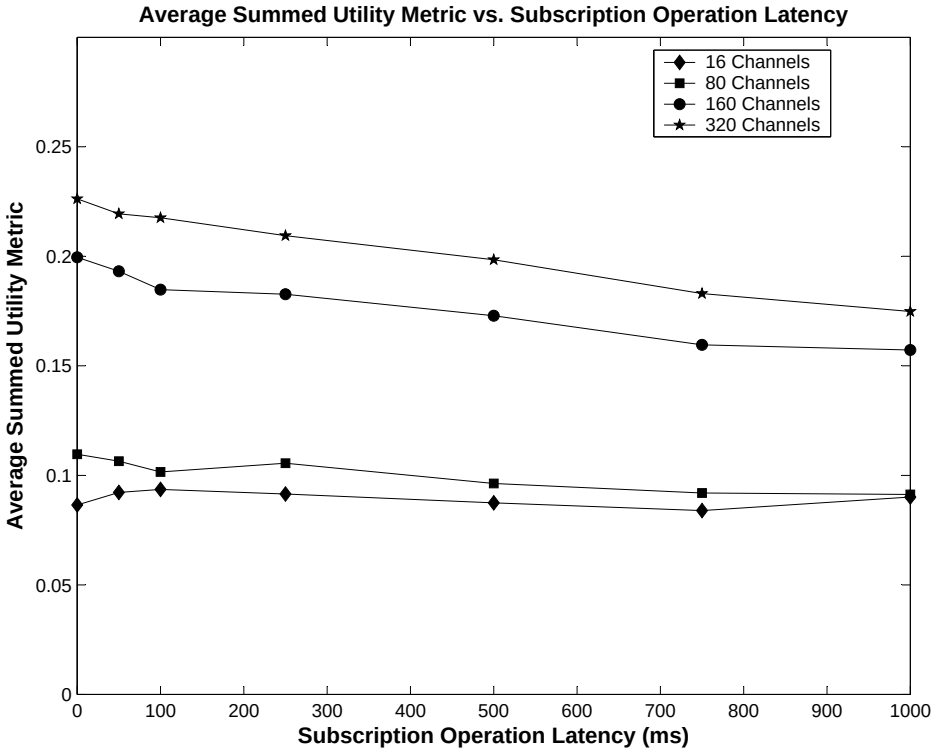

Figure 7.14: The number of clusters, which maps directly to the number of channels, impacts the degree to which subscription latency affects performance. Performance improves as additional channels are made available due to increased flexibility in data access. However, the impact of overhead costs is more pronounced as a result of the higher frequency of subscription operations. This is evidenced by the steeper slope in high-channel configurations.

large size for G provides far greater flexibility in ACS management and enables highly customized data flows.

Consider the size of the overall channel set, noted as $|G|$. In the extreme, a dataset with $|G| = 1$ corresponds to a single channel and is equivalent to a common monolithic file that all clients must download. When $|G|$ is maximized so that every byte of data is available through a unique channel, clients are given random access to the database.

As a result, additional channels generally result in higher performance. Figure 7.14 shows a series of experiments performed using four different channel configurations. As expected, the SUM value is highest when the number of channels is greatest.

The benefit of additional channels is greatest when the subscription operation latency is negligible. However, the impact of latency on performance is more pronounced in high-channel configurations. This is evidenced in Figure 7.14 by the steeper slopes in the high-channel plots.

The steeper decline in performance is a direct result of the increased rate of channel subscription operations for high-channel configurations. The faster pace of subscriptions is exactly what makes large channel sets beneficial: additional channels aid in composing custom data flows. However, the increase in subscription operations reduces the expected listen time for any given channel. As reflected in the results as well as Equation 7.7, a shorter expected listen time magnifies the impact of changes in the subscription latency by reducing the efficiency.

At first glance, the results seem to hint that it would be desirable to use an enormous number of channels to obtain the best overall performance. In fact, in the absence of any overhead costs, that would be the case. This is why the random-access unicast configuration in the TCP-based GAL prototype outperforms both CSA variants for small user groups.

However, in a CSA-based system, which is needed to support large user groups, it would not be practical to place every edge in its own cluster. First, any multicast infrastructure will introduce some amount of subscription operation latency. Second, the expected listen time in this extreme configuration would be extremely short. Equation 7.7 shows that these two factors combine to produce extremely poor efficiency values and lead to a poor solution.

A practical system design must balance the benefit of a high channel count with the overhead cost of supporting it. The optimal compromise depends very highly on subscription operation latencies associated with the multicast infrastructure. This conclusion motivates additional work in developing more efficient multicast algorithms with low subscription operation overheads, especially in high-churn environments.

7.5.9 Performance Model Implications

Throughout this chapter, the experiments have motivated a simple performance model that describes the impact of several engineering parameters. The ideal performance model of Equation 7.3 highlights that the best possible performance is represented by the GAL prototype built upon the unicast network model where clients have random access privileges to the non-linear media dataset. In practice, however, the experiments of Chapter 6 show that the ideal solution does not scale to support large user groups.

CSA is proposed to address this challenge by mapping clusters of semantically consistent data to a large set of scalable delivery communication channels. Using broadcast as the ideal channel-oriented scalable delivery network model, this chapter developed

the performance model of Equation 7.4. The broadcast performance model shows that the broadcast solution is immune to scale and comes as close as possible to the ideal performance given the cluster penalty, CP.

Finally, the performance model for multicast in Equation 7.6 introduces the MO factor to represent the group management overhead. The amount of overhead is directly impacted by subscription operation latencies and therefore the efficiency term of Equation 7.7.

The results in this chapter show that the performance of the multicast-based CSA prototype depends very heavily on the underlying properties of the multicast infrastructure. Because CSA employs the multicast model in a novel and aggressive fashion, the particular properties most important to CSA performance are not those traditionally optimized in widely deployed multicast algorithms.

For this reason, CSA can serve as a true stress test for emerging multicast solutions. It's aggressive channel hopping strategy requires the optimization of several important properties often overlooked by multicast developers. For example, the typical measures of quality for multicast protocols are the notions of stress and stretch. For CSA applications, however stretch is of little consequence due to the repeated carousel transmission schedule. Similarly, the minimization of stress is less important than the *stability* of stress levels because fluctuations have a negative impact on congestion control.

Finally, and perhaps most importantly, CSA is highly sensitive to join and leave latencies because of their impact on overall efficiency and the MO factor of the performance model. Typical multicast algorithms are designed for long-life flows and therefore expend significant effort in global optimization of the distribution topology at the expense of longer join and leave latencies. Given CSA's shorter expected duration of subscriptions and sensitivity to MO, these design decisions regarding long-life flows should be reconsidered.

While beyond the scope of this dissertation, my colleagues and I have made significant progress in developing a new approach to ALM protocols that we hope will significantly improve on the best of today's current multicast technologies. The new protocol, named StrandCast (Begnoche et al., 2005), aims to support high-churn multicast sessions with very low join and leave latencies and a fixed stress levels for each node in the ALM overlay topology.

7.6 Summary

This chapter presented CSA, a scalable solution for non-linear media streaming. Motivated by the shortcomings of the TCP-based GAL prototype of Chapter 6, CSA takes the simple server design philosophy outlined in the previous chapter to the next level by removing literally every per-client task from the server. This is accomplished by employing a group-based channel oriented network model in a novel way to allow non-linear media streaming to large groups of users.

This chapter began with an overview the CSA architecture. CSA calls for the grouping of semantically similar edges within a dataset's RG into relatively large clusters. The data from each cluster is then mapped to an individual communication channel, producing a set of semantically meaningful channels, G. At runtime, a CSA server transmits all of the channels all of the time, repeatedly transmitting the data assigned to each channel in carousel fashion.

CSA provides individual clients with custom data flows for interactive applications using standard multicast join and leave operations. At any point in time, a client is actively receiving a set of channels. This set is called the ACS. Congestion control is performed by managing the size of the ACS. Content control is performed by choosing which of the channels in G should be added or removed from the ACS.

The chapter concluded with a presentation of several experimental results that highlight CSA's most important performance characteristics over both broadcast and multicast network models. Throughout the results, a CSA performance model is developed to highlight the important factors that contribute to overall system performance.

The performance model indicates the high level of scalability inherent in the CSA architecture. At the same time, the performance model motivates several areas of focus important for the development of the next generation of multicast algorithms capable of achieving even better results for the scalable streaming of non-linear media.

Chapter 8

Summary and Conclusion

The research presented in this dissertation lies at the crossroads of two pronounced trends in the world of computing. First, digital media techniques have grown dramatically in importance across a vast array of domains. Second, the proliferation of broadband communication technologies has led to changes in our way of life that have had a tremendous impact on how we access information, communicate, and value location.

At the intersection of these two trends, where digital media and broadband networking meet, there is the broad application space that combines the power of both technology trends. Included in this application space is digital media streaming: a technique for efficiently distributing large media datasets to interested receivers.

The recipient of a digital media stream receives a continuous flow of data which has been carefully arranged to allow the receiving application to make use of the data as it arrives. Streaming has been widely deployed to support access to multimedia objects because these objects are typically large in size and multimedia applications benefit greatly from the reduced access time afforded by the technique.

Existing media streaming applications have been largely limited to linear media domains. Linear media objects, most notably audio and video, consist of linear arrangements of data ordered statically for all recipients. The prevalence of linear media among network streaming applications is not an isolated phenomenon. Rather, it parallels the long time dominance of linear media within more traditional media channels, including books, film, and television.

Recently, however, the tide has been changing. Advances in computing and interactive technology have led to an increasing adoption of non-linear media experiences. The data contained in non-linear media objects, such as video games, interactive visu-

alizations, and virtual environments, is accessed in unique patterns by each user. The individual access patterns are dictated by independent interaction events occurring within each user's application.

Non-linear objects are therefore best suited for applications which foster individual interaction, such as video games. This is in contrast to linear media objects which are easily applicable to large audiences, as evidenced by the large crowds that turn up at movie theaters.

In the context of media streaming, non-linear media poses new challenges. In particular, the need to deliver a custom data flow to each member of a large group of independent users can not be solved using traditional media streaming techniques. It is this challenge—scalable non-linear media streaming—which has been the primary topic addressed by the research presented in this dissertation.

For fully interactive non-linear media applications, the task of adapting data flows to match individual resource requirements is a complicated task. Performing this task in a scalable fashion is an even bigger challenge. In my research, I have begun to address some of the fundamental obstacles confronting such systems.

My work provides a complete framework that supports the streaming of non-linear media to a large group of independently operating and heterogeneously provisioned clients. As the experimental results of Chapter 7 have shown, my proposed framework scales well for large user groups and allows for each client to independently perform both content control and congestion control.

My research proposes three primary formalisms: (1) an abstract representation model for non-linear media called a *Representation Graph*, (2) a quantitative adaptation algorithm that frames adaptation as a maximization problem, and (3) the *Channel Set Adaptation* method that exploits scalable channel-based network models such as broadcast and multicast to provide highly scalable non-linear media streaming.

8.1 Research Contributions

The research presented in this dissertation has made several contributions. These include conceptual developments in the areas of data representation, adaptation, and distribution, as well practical contributions in both software library implementation and a novel prototype application.

Specifically, the research contributions of my work include:

- **A Graph-based Model for Data Representation**: I presented the Representation Graph in Chapter 3. An RG is an abstract data model that can be used to express the complex multidimensional and multi-resolutional properties common to a large space of non-linear media data sets.

  The RG is a graph-based abstraction which models semantic data relationships and syntactic dependencies through a network of nodes, edges, and clusters. Nodes are used to represent individual elements of information, edges are used syntactic data dependencies between nodes, and clusters express access-level restrictions on how the data is accessed.

  I use the generic RG abstraction as the core data representation throughout my other work. This enables the incorporation of the remaining contributions into any application which can map its dataset to the RG model.

- **Utility-Driven Multidimensional Adaptation Algorithm**: An essential component of any non-linear media streaming application is data adaptation. As information flows from the media server to the consumer, the flow of data must be controlled to satisfy both the limited communication resources as well as the receiver's application-level needs and preferences.

  Previously, this task has been most often performed via ad hoc solutions based on complex rule systems or application-specific heuristics. In Chapter 4, I proposed a general framework for expressing multidimensional adaptation via a spatial utility metric defined as a function over the RG data abstraction.

  The adaptation framework poses adaptation as a maximization problem in which the goal is to maximize the utility of the received data while simultaneously minimizing the access cost. The framework allows for the efficient computation of adaptive decisions via a quantitative and iterative adaptation algorithm. The algorithm provides intuitive spatial mechanisms for expressing application-specific data requirements without the burdensome task of developing complex rule sets.

- **Simple Server Design Philosophy**: In Chapter 6, I proposed a central philosophy for designing scalable distribution systems. This philosophy, which serves as the cornerstone behind the CSA approach to non-linear media streaming, stipulates that all per-client tasks must be pushed away from any centralized resources and placed as close to the individual clients as possible in order to truly achieve scalable performance.

This design philosophy is strongly embraced by the architecture of CSA. The CSA server model does not perform any per-client work resulting in a constant level of work independent of the size of the client pool. Because it adheres so strictly to the simple server design philosophy, the CSA streaming architecture is, theoretically, infinitely scalable.

- **Channel Set Adaptation**: I proposed Channel Set Adaptation, a channel-based communication framework for scalable and adaptive streaming of non-linear media in Chapter 7. CSA uses a novel method of channel subscription management to achieve truly scalable streaming of non-linear media. CSA exploits the scalable aspects of multicast and broadcast networks while developing a novel approach to content control.

  Typically such scalable channel-based network models are used to distribute identical flows of information to large audiences. CSA aggressively utilizes subscription operations to deliver unique data flows to each client in a highly scalable approach.

- **Performance Model**: I developed a performance model in Chapter 7 that expresses the impact on application performance of a number of important system parameters. The performance model has been developed after careful analysis of the experimental results gathered while testing a non-linear media streaming prototype on a network emulation testbed.

  The performance model provides an insightful look at the impact that underlying system properties can have on overall performance. The performance model highlights where current multicast implementations can be most improved in order to support novel application-level technologies such as Channel Set Adaptation.

In addition, my research has led to the development of the following practical contributions:

- **A Library for Generic Adaptation**:

  I have implemented a C++ library, GAL, based on the representation abstraction and multidimensional adaptation algorithms presented in this dissertation. The library has direct support for multi-dimensional multimedia adaptation. Chapter 5 defines the three-layer architecture of the middleware library and presents an overview of the library API.

- **Motivating Application Prototype**: As part of my evaluation, I have implemented a scalable non-linear media streaming infrastructure designed for an image-based rendering application that reconstructs entire spaces for virtual exploration. The prototype is described in detail in Chapter 6. One promising application of this technology is in digital museums, where a centrally stored image-based model of a famous space can be streamed to a large number of virtual visitors.

8.2 Future Work

The experimental results from the evaluation of CSA in Chapter 7 are extremely promising and hint at the extremely scalable performance made possible by the channel-based distribution method and adaptation algorithm proposed in this work.

However, there is still more exploration to be performed. In this section, I provide a brief glimpse into the areas of future work which I feel must be addressed before a practical and deployable solution to scalable streaming of non-linear media can be realized.

8.2.1 Relaxing the Communication Model

One important area of future work is a relaxation of the communication model. The experiments performed throughout the evaluation of GAL and CSA, as outlined in Section 6.2.2, were based on a network model that assumes independent bandwidth bottlenecks located on each client's last-mile link.

This assumption is based on the common idea that a consumer's home network connection, via technologies such as DSL or dial-up modems, is typically far slower than the core network links at the heart of corporate network service providers. In practice, however, this assumption is not always true. The exceptions to the last-mile assumption are becoming especially important as newer and faster last-mile technologies (such as fiber-optics) are rolled out to reach individual customers.

Additional improvements to the CSA architecture must be developed that address bottlenecks that occur further away from the client. In particular, the interference between multiple clients' join experiments as CSA tries to grow the ACS may prove problematic to the current CSA adaptation algorithm. This complication makes it difficult for an individual client to know if congestion has been caused by their own join

experiment or by one belonging to another client.

Similar problems have been observed with more traditional layered multicast techniques aimed at layered media applications, such as Receiver-driven Layered Multicast (McCanne et al., 1996). Certain derivatives of the original Receiver-driven Layer Multicast proposal have explored solutions to the shared bottleneck problem (Legout and Biersack, 2000; Wu et al., 1997) and may serve as models for future improvements to CSA.

8.2.2 Moving Beyond Emulation

Another area for future work is in moving beyond network emulation to evaluation within the real operating conditions of the Internet. The long-term goal for the CSA architecture is the ability to support large scale non-linear media distribution via a wide-scale deployment of the adaptation and distribution technologies developed in my research.

The experiments performed as part of the CSA evaluation presented here were performed on an emulation-based network testbed. Emulation provides superior realism when compared to simulation because it sends real data packets over real network equipment to be processed by applications running on real computers. However, it is a closed environment where the behavior in the face of competing network traffic can only be observed by introducing artificial cross traffic.

Before widespread deployment can be achieved, a thorough evaluation is required that measures the effectiveness of CSA on real Internet topologies and facing actual competing network traffic. This type of real-world experimentation is now possible using PlanetLab (Bavier et al., 2004), a network of hundreds of nodes distributed across the Internet at nearly 300 sites around the world.

8.2.3 Improving Multicast

The CSA performance model developed in Chapter 7 highlights the critical impact of the underlying multicast protocol's properties on overall system performance. Existing multicast algorithms, both infrastructure-based and ALM, are typically designed for long-life subscriptions and the design decisions made during their development reflect this aim.

Even without the unique demands of CSA, standard IP multicast (Deering and Cheriton, 1990) has faced problems with both functionality and deployment. As a

result, a vast research effort has been invested in developing overlay technologies, called ALM, that combine the scalable performance of a multicast network model with a deployable infrastructure built at the application layer (Banerjee et al., 2002; Castro et al., 2002; Chu et al., 2000; Chu et al., 2001; Ratnasamy et al., 2001).

However, CSA places unique demands on the underlying multicast protocol. First, CSA's extremely aggressive use of multicast join and leave operations stresses the group management infrastructure far more than traditional multicast-based applications. Second, CSA reacts differently to changes to both stress and stretch, two common evaluation metrics applied to ALM protocols. CSA is immune to stretch due to its carousel transmission schedule, and it is extremely sensitive to stress because it is a greedy framework that attempts to utilize as much bandwidth as possible.

Using the performance model as a guide, I believe that an application-layer multicast algorithm can be designed that meets the requirements of CSA far better than the algorithms that have so far been proposed. My colleagues and I have already begun working on such a protocol (Begnoche et al., 2005) and significant effort is still required. Developing such a protocol can further drive performance closer to the ideal benchmark set by the broadcast-based CSA performance evaluations.

8.3 Summary

Linear media streaming is ubiquitous in today's digital world. Radio programs, sports events, and news coverage are available on-line and on demand. However an emerging class of non-linear media objects, such as large 3D computer graphics models and visualization databases, remain beyond the scope of existing streaming technology.

In this dissertation, I begin to address the area of scalable and adaptive streaming for non-linear media. Via experimentation and analysis, I have shown that the scalable streaming of non-linear media to a large group of independently adapting clients is enabled through Channel Set Adaptation: a framework that maps a partitioned media representation to a set of relatively thin multicast communication channels to provide scalable congestion and content control.

Channel Set Adaptation provides a solution that is, theoretically, infinitely scalable. It allows individual clients to arbitrarily compose custom data flows that match their local requirements in a fully scalable manner. CSA therefore allows for the efficient delivery of individualized and interactive non-linear experiences on a large scale.

BIBLIOGRAPHY

Acharya, S., Alonso, R., Franklin, M., and Zdonik, S. (1995). Broadcast disks: data management for asymmetric communication environments. In *Proceedings of ACM SIGMOD*, pages 199–210.

Al-Regib, G., Altunbasak, Y., Rossignac, J., and Mersereau, R. (2002). Protocol for streaming compressed 3-d animations over lossy channels. In *Proceedings of IEEE International Conference on Multimedia and Expo*, pages 353–356.

Aliaga, D. G., Funkhouser, T., Yanovsky, D., and Carlbom, I. (2002). Sea of images. In *Proceedings of IEEE Visualization*.

Banerjee, S., Bhattacharjee, B., and Kommareddy, C. (2002). Scalable application layer multicast. In *Proceedings of ACM Sigcomm*.

Bavier, A., Bowman, M., Chun, B., Culler, D., Karlin, S., Muir, S., Peterson, L., Roscoe, T., Spalink, T., and Wawrzoniak, M. (2004). Operating system support for planetary-scale network services. In *First Symposium on Networked Systems Design and Implementation*, pages 253–266.

Begnoche, B., Gotz, D., and Mayer-Patel, K. (2005). The design and implementation of strandcast. Technical Report TR05-004, The University of North Carolina at Chapel Hill Department of Computer Science.

Beynon, M., Ferreira, R., Kurc, T. M., Sussman, A., and Saltz, J. H. (2000). Datacutter: Middleware for filtering very large scientific datasets on archival storage systems. In *IEEE Symposium on Mass Storage Systems*, pages 119–134.

Bischoff, S. and Kobbelt, L. (2002). Towards robust broadcasting of geometry data. *Computers and Graphics*.

Black, J. (2001). This three-way slugfest is no game. *Business Week*. December 13.

Boll, S., Klas, W., and Wandel, J. (1999). A cross-media adaptation strategy for multimedia presentations. In *Proceedings of ACM Multimedia*.

Bowers, S., Delcambre, L., Maier, D., Cowan, C., Wagle, P., McNamee, D., Meur, A.-F. L., and Hinton, H. (2000). Applying adaptation spaces to support quality of service and survivability. In *DARPA Information Survivability Conference and Exposition*.

Castro, M., Druschel, P., Kermarrec, A.-M., and Rowstron, A. (2002). Scribe: A largescale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*.

Chakareski, J., Han, S., and Girod, B. (2003). Layered coding vs. multiple descriptions for video streaming over multiple paths. In *Proceedings of ACM Multimedia*.

Chatterjee, S., Sydir, J., Sabata, B., and Lawrence, T. (1997). Modeling applications for adaptive qos-based resource management. In *Proceedings of IEEE High Assurance Systems Engineering Workshop*.

Chu, Y., Rao, S. G., and Zhang, H. (2000). A case for end system multicast. In *Proceedings of ACM SIGMETRICS*.

Chu, Y.-H., Rao, S. G., Seshan, S., and H.Zhang (2001). Enabling conferencing applications on the internet using on overlay multicast architecture. In *Proceedings of ACM SIGCOMM*.

Clark, D. D. and Tennenhouse, D. L. (1990). Architectural considerations for a new generation of protocols. In *Proceedings of ACM SIGCOMM*.

Cohen-Or, D., Chrysanthou, Y., and Silva, C. (2001). A survey of visibility for walkthrough applications. *SIGGRAPH Course Nodes # 30*.

Cooley, J. W. and Tukey, J. W. (1965). An algorithm for the machine computation of the complex fourier series. *Mathematics of Computation*, 19:297–301.

Deering, S. E. and Cheriton, D. R. (1990). Multicast routing in a datagram internetworks and extended lans. *ACM Transactions on Computer Systems*, 8(2):85–110.

Floriani, L. D. and Magillo, P. (2002). Regular and Irregular Multi-Resolution Terrain Models: A Comparison. In *Proceedings of 10th ACM International Symposium on Advances in Geographic Information Systems (ACM-GIS'02)*, pages 143–148.

Funkhouser, T. and Sequin, C. (1993). Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *Proceedings of ACM SIGGRAPH*.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

Gortler, S., Grzeszczuk, R., Szeliski, R., and Cohen, M. (1996). The lumigraph. In *Proceedings of ACM SIGGRAPH*, pages 43–54.

Gotz, D. (2004). Supporting Adaptive Remote Access to Multiresolutional or Hierarchical Data for Large User Groups. In *Proceedings of ACM Multimedia Doctoral Symposium*, New York, NY, USA. Association for Computing Machinery.

Gotz, D. and Mayer-Patel, K. (2004). A General Framework for Multidimensional Adaptation. In *Proceedings of ACM Multimedia*, New York, NY, USA. Association for Computing Machinery.

Gotz, D. and Mayer-Patel, K. (2005a). A framework for scalable delivery of digitized spaces. *International Journal on Digital Libraries*, 5(3):205–218. Special Issue on Digital Museums.

Gotz, D. and Mayer-Patel, K. (2005b). Gal: A middleware library for multidimensional adaptation. Technical Report TR05-023, The University of North Carolina at Chapel Hill Department of Computer Science.

Gotz, D. and Mayer-Patel, K. (2005c). Scalable and adaptive streaming for non-linear media. Technical Report TR05-022, The University of North Carolina at Chapel Hill Department of Computer Science.

Gotz, D., Mayer-Patel, K., and Manocha, D. (2002). IRW: An incremental representation for image-based walkthroughs. In *Proceedings of ACM Multimedia*.

Herman, G., Lee, K. C., and Weinrib, A. (1987). The datacycle architecture for very high throughput database systems. In *Proceedings of ACM SIGMOD*.

Hoppe, H. (1996). Progressive meshes. In *Proceedings of ACM SIGGRAPH*.

Hua, K. A. and Sheu, S. (1997). Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems. In *Proceedings of SIGCOMM*, pages 89–100.

IBM Corporation and Microsoft Corporation (1991). *Multimedia Programming Interface and Data Specifications 1.0*.

ISO/IEC (1993). *ISO/IEC International Standard 11172*.

ISO/IEC (1995a). *ISO/IEC International Standard 13818*.

ISO/IEC (1995b). *ISO/IEC International Standard 13818-3 - Part 3: Audio*.

ISO/IEC (2000). *ISO/IEC International Standard 15444, Final Committee Draft*.

Juhn, L.-S. and Tseng, L.-M. (1997). Harmonic broadcasting for video-on-demand service. *IEEE Transactions on Broadcasting*, 43(3):268–271.

Kanakia, H., Mishra, P., and Reibman, A. (1993). An adaptive congestion control scheme for real-time packet video transport. In *Proceedings of ACM SIGCOMM*.

Kerlow, I. V. (2003). *The Art of 3-D Computer Animation and Effects*. John Wiley and Sons, third edition.

Kharif, O. (2002). Focusing on picture-perfect diagnoses. *Business Week*. October 15.

Kharif, O. (2005). Turning wi-fi into a must-have. *Business Week*. January 11.

Lamar, E. C., Hamann, B., and Joy, K. I. (1999). Multiresolution techniques for interactive texture-based volume visualization. In *Proceedings of IEEE Visualization*.

Legout, A. and Biersack, E. W. (2000). PLM: Fast convergence for cumulative layered multicast transmission schemes. In *Proceedings of ACM SIGMETRICS*.

Levoy, M. and Hanrahan, P. (1996). Light field rendering. In *Proceedings of ACM SIGGRAPH*, pages 31–42.

Lindstrom, P., Koller, D., Ribarsky, W., Hodges, L., Faust, N., and Turner, G. (1996). Real-time continuous level of detail rendering of height fields. In *Proceedings of ACM SIGGRAPH*, pages 109–118.

Luebke, D. (2001). A developer's survey of polygonal simplification algorithms. *IEEE Computer Graphics and Applications*, pages 24–35.

McCanne, S., Jacobson, V., and Vetterli, M. (1996). Receiver-driven layered multicast. In *Proceedings of ACM SIGCOMM*.

McIlhagga, M., Light, A., and Wakeman, I. (1998). Towards a design methodology for adaptive applications. In *Proceedings of ACM/IEEE International Conf. on Mobile Computing and Networking*.

Meehan, M., Razzaque, S., Whitton, M. C., and Frederick P. Brooks, J. (2003). Effect of latency on presence in stressful virtual environments. In *Proceedings of IEEE Virtual Reality*.

Miller, G., Rubin, S., and Poncelen, D. (1998). Lazy decompression of surface light fields for pre-computer global illumination. *Proceedings of Eurographics Workshop on Rendering*, pages 281–292.

Pennebaker, W. and Mitchell, J. (1993). *JPEG: Still Image Data Compression Standard*. Van Nostrand Reinhold.

Perkins, C., Hodson, O., and Hardman, V. (1998). A survey of packet-loss recovery techniques for streaming audio. *IEEE Network*, 12:40–48.

Policroniades, C., Chakravorty, R., and Vidales, P. (2003). A data repository for fine-grained adaptation in heterogeneous environments. In *International Workshop on Data Engineering for Wireless and Mobile Access*.

Ramanujan, R. S., Newhouse, J. A., Kaddoura, A. A. M. N., Chartier, E. R., and Thurber, K. J. (1997). Adaptive streaming of mpeg video over ip networks. In *Proceedings of the IEEE Conference on Computer Networks*.

Rao, K. R. and Yip, P. (1990). *Discrete Cosine Transfomr: Algorithms, Advantages, Applications*. Academic Press, Inc.

Ratnasamy, S., Handley, M., Karp, R., and Shenker, S. (2001). Application-level multicast uisng content-addressable networks. In *Proceedings of 3rd International Workshop on Networked Group Communication*.

Rejaie, R., Handley, M., and Estrin, D. (1999). Quality adaptation for congestion controlled video playback over the internet. In *Proceedings of ACM SIGCOMM*, pages 189–200.

Rejaie, R., Handley, M., and Estrin, D. (2000). Layered quality adaptation for internet video streaming. *IEEE Journal on Selected Areas of Communications (JSAC)*. Special issue on Internet QoS.

Rendleman, J. (2002). Have dsl and firewalls, will telecommute. *InformationWeek*. March 25.

Rickitt, R. (2000). *Special Effects: The History and Technique*, page 175. Watson-Guptill Publications.

Rizzo, L. (1997). Effective Erasure Codes for Reliable Computer Communication Protocols. *ACM Computer Communication Review*, 27(2):24–36.

Rowe, L. and Smith, B. (1992). A continuous media player. In *Network and Operating System Support for Digital Audio and Video*.

Rusinkiewicz, S. and Levoy, M. (2001). Streaming qsplat: A viewer for networked visualization of large, dense models. In *Proceedings of ACM Interactive 3D Graphics*.

Salkever, A. (2003). Everyone's smiling for digital cameras. *Business Week*. December 9.

Samet, H. (1984). The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260.

Schulzrinne, H., Casner, S., Frederick, R., and Jacobson, V. (1996). RTP: A Transport Protocol for Real-Time Applications. *IETF Netowrk Working Group*. RFC 1889.

Schulzrinne, H., Rao, A., and Lanphier, R. (1998). Real Time Streaming Protocol (RTSP). *IETF Netowrk Working Group*. RFC 2326.

Stice, R. (2005). Kodak's many negatives. *Business Week*. January 7.

Taubman, D. and Zakhor, A. (1994). Multirate 3-d subband coding of video. *IEEE Transactions on Image Processing*, 3(5):572–588.

Teler, E. and Lischinski, D. (2001). Streaming of complex 3d scenes for remote walkthroughs. In *Proceedings of Eurographics*.

Union, I. T. (1993). *ITU-T H.261: Video codec for audiovisual services a p x 64 kbits.*

Varadhan, G. and Manocha, D. (2002). Out-of-core rendering of massive geometric environments. In *Proceedings of IEEE Visualization*.

Vetterli, M. and Kavacevic, J. (1995). *Waveletes and Subband Coding*. Prentice Hall PTR.

Viswanathan, S. and Imielinski, T. (1996). Metropolitan area video-on-demand service using pyramid broadcasting. *Multimedia Systems*, 4:197–208.

Walpole, J., Krasic, C., Liu, L., Maier, D., Pu, C., McNamee, D., and Steere, D. (1999). Quality of service semantics for multimedia database systems. *Database Semantics: Semantic Issues in Multimedia Systems*.

White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., and Joglekar, A. (2002). An integrated experimental environment for distributed systems and networks. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA. USENIX Association.

Wildstrom, S. H. (2005). Google's magic carpet ride. *Business Week*. July 7.

Wu, L., Sharma, R., and Smith, B. (1997). Thin streams: An architecture for multicasting layered video. In *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*.

Zhang, C. and Li, J. (2001). Interactive browsing of 3d environment over the internet. In *Proceedings of Visual Communication and Image Processing*.