Visualcasting: Scalable Real-time Image Distribution in Ultra-High Resolution Display Environments

BY

BYUNGIL JEONG
B.S., Seoul National University, 1997
M.S., Seoul National University, 1999

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Chicago, 2009

Chicago, Illinois

ACKNOWLEDGEMENTS

I greatly appreciate the assistance and support of the following individuals:

- Jason Leigh, Andrew Johnson, Luc Renambot for advising my PhD research throughout my tenure at EVL, for serving on the dissertation committee, and for providing the opportunity to work on SAGE and Visualcasting
- Tom DeFanti for making time to serve on my dissertation committee and for providing the network infrastructure on which I developed and tested SAGE and Visualcasting
- Larry Smarr for leading the OptIPuter that has sponsored my PhD research and for supporting the deployment of SAGE and Visualcasting on the international OptIPuter community
- Erik Hofer for taking time and effort to travel to Chicago to serve on my dissertation committee and for providing valuable feedback
- SAGE team members: Ratko Jagodic, Hyejung Hur, and Sungwon Nam for their effort in designing and implementing SAGE UI, SAGE audio, and SAGE sync algorithm
- Alan Verlo, Lance Long and Pat Hallihan for providing great support for SAGE and Visualcasting tests
- Venkatram Vishwanath for providing valuable input for my research and for helping with submitting this dissertation and the required paperwork.
- Allan Spale and Nicholas Schwarz for reviewing and editing this dissertation

Finally, I will always be grateful to my family, my wife Sarah, and my son Joshua for encouraging me with endless love and support and to my God for His amazing grace throughout my PhD study.

BJ

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

API	Application Programming Interface
CPU	Central Processing Unit
EVL	Electronic Visualization Laboratory
GUI	Graphical User Interface
HD	High Definition
IP	Internet Protocol
LCD	Liquid Crystal Display
PC	Personal Computer
RDP	Remote Desktop Protocol
SDL	Simple Directmedia Layer
TCP	Transmission Control Protocol
UCSD	University of California at San Diego
UDP	User Datagram Protocol
UIC	University Illinois at Chicago
VNC	Virtual Network Computing
WAN	Wide Area Network
XDMX	Distributed Multi-head X11

SUMMARY

Visualization has proven its value in scientific advances by helping scientists gain insight from their data and verify scientific computations. The amount of scientific data collected from sensors and simulations can easily be on the order of petabytes of data. Visualization of this large-scale data requires cluster computing, and more often than not distributed computing over high-speed networks, as the size of the data exceeds the capacity of the average computing clusters, and the data may not even reside locally. To view visualizations of large-scale data at or near native resolution, scalable tiled display walls are increasingly being used for scientific visualization.

In this context, the Scalable Adaptive Graphics Environment (SAGE) has been developed to support large-scale data visualization in a distributed visualization environment that includes ultra-high resolution scalable tiled displays. It is a specialized middleware that enables real-time streaming of extremely high-resolution graphics and high-definition video from remotely distributed rendering and storage clusters to scalable display walls over ultra high-speed networks. This dissertation extends SAGE to support distant collaboration between multiple endpoints.

In the SAGE framework, each visualization application streams its rendered pixels to the virtual high-resolution frame buffer of SAGE, allowing users to freely move, resize and overlap the application windows on the display. Every window movement or resize operation requires dynamic and non-trivial reconfigurations of the involved graphics streams. These reconfigurations become even more complex when SAGE is required to support multiple collaboration endpoints with different tiled display configurations and application window layouts.

Visualcasting is a new SAGE network service to address this problem using a high-speed bridging system that receives pixel streams from rendering clusters and that duplicates and sends them to each end-point. This enables distant collaboration among international researchers in scalable display environments. Using the Visualcasting service, collaborators can share their visualizations and interact with each other through high-definition video conferencing in the SAGE Framework.

Intellectual Merit:

Visualcasting addresses the problem of high-performance graphics multicasting for tiled displays. Although a variety of techniques exist for supporting reliable multicasting, reliably multicasting graphics data onto remote tiled displays with heterogeneous display configurations is still a challenging, unsolved problem. Furthermore, Visualcasting addresses the problems of supporting heterogeneous endpoints in both network and display capacity as well as scaling the graphics multicasting with the number of endpoints and applications.

Broader Impact:

Visualcasting enables SAGE to multicast high-definition video and ultra-high resolution visualizations in scalable, real-time manner across globally distributed research centers. This capability will demonstrate a new way in which high-performance networking and visualization can be used in a broad range of research, academic and commercial applications. Furthermore, understanding the requirements, benefits and limitations of Visualcasting and alternative approaches will provide valuable input into the future of Internet system design.

CHAPTER 1 INTRODUCTION

Chapter 1 outlines the research areas investigated and summarizes the problems solved by this dissertation. Section 1.1 provides an overview of the research areas and introduces SAGE and Visualcasting. Section 1.2 describes the problems solved and articulates the approaches used to solve the problems.

1.1 Overview

In a decade's time, high-performance computing has proven its value in the fields of science, medicine, engineering, education, and filmmaking. These data-intensive domains rely on computational grid technology and high-quality visualizations to produce meaningful insights from terabytes of raw data. As research and development becomes increasingly global and multidisciplinary, the need for a computing infrastructure to support collaborative work among distributed users has grown dramatically [Leigh06]. Because the rate of decline of the cost of bandwidth far exceeds that of computing and storage [Stix01], it has recently become more cost-effective for the domain users to connect to ultra-high-speed networks rather than for them to maintain their own large computing, storage and visualization systems. For this reason, it becomes more cost-effective for users to build low-cost, thin network clients than to have to purchase and maintain their own rendering farms, storage repositories, etc.

In this context, the Electronic Visualization Laboratory (EVL) at University of Illinois at Chicago (UIC) and the California Institute for Telecommunications and Information Technology (Calit2) at University of California, San Diego (UCSD) have led the OptIPuter project [Smarr03,

Leigh03], a major NSF-funded initiative to design an advanced cyber-infrastructure for data-intensive science using optical networks. In order to develop the OptIPuter system, OptIPuter partners, who are primarily domain scientists have provided feedback about essential user requirements for the development of a collaborative scientific visualization system. One user requirement is that they want to view and interact with visualizations of multiple heterogeneous datasets simultaneously on ultra-high resolution display walls. Secondly, they want to reduce cost by sharing remote visualization resources and storage servers in this cyber-infrastructure. Lastly, they want a system that supports human global collaboration across multiple sites that each contains ultra-high resolution display environments.

EVL/UIC has developed the hardware and software systems to support these OptIPuter user requirements. LambdaVision (see Figure 1) is the hardware system, which is an 11x5 tiled display with a total resolution of 105 megapixels fed by network bandwidth on the order of tens of gigabits. A scalable high-resolution display like LambdaVision is essential to render complex geometric models without losing their details. Even though a geometric model has millions of triangles, if it is

Figure 1. LambdaVision display driven by SAGE

rendered onto a single desktop display, only a small subset of those triangles could contribute to the final image [Klosowski02]. Also, geoscientists working with aerial and satellite imagery (365Kx365K pixel maps) and neurobiologists imaging the brain with montages consisting of thousands of pictures from high-resolution microscopes (4Kx4K pixel sensor) are good examples of LambdaVision users.

The Scalable Adaptive Graphics Environment (SAGE) with the Visualcasting software system supports collaborative scientific visualization in scalable high-resolution display environments that include hardware such as LambdaVision. SAGE is an "operating system" for tiled-display environments, letting users launch distributed visualization applications on remote clusters of computers and stream the visualizations directly to their tiled displays, where they can be viewed and manipulated (see Figure 2). The uniqueness of SAGE lies in its ability to enable multiple parallel rendered applications that run on separate and distantly located computer clusters to stream the visualizations to any portion of a tiled display as individually managed windows. This ability allows multi-tasking on tiled displays.

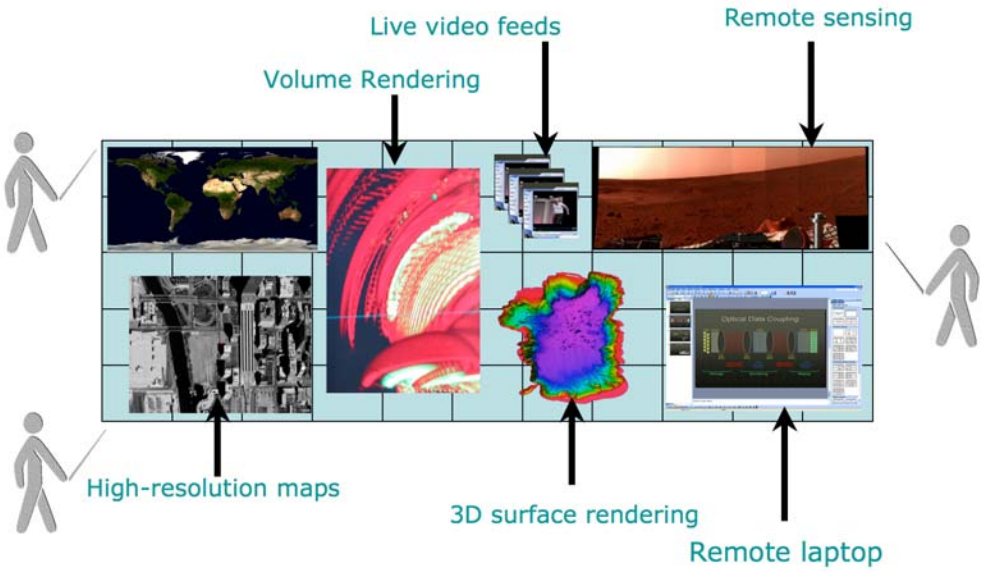


Figure 2. An example of SAGE session

A fundamental requirement of this high-resolution collaborative visualization system is the ability to broadcast or multicast visualizations to all collaborating sites so that all participants can simultaneously see and interact with the data. Multicasting in these high-resolution environments poses a significant challenge because potentially tens of gigabits of network bandwidth are needed to support collaborative visualization. This dissertation investigates this problem in detail and implements a scheme called Visualcasting that is specifically designed to provide the kind of image multicasting service needed for ultra-high definition visualization. Visualcasting enables SAGE-based global collaboration across multiple sites by allowing users to share their visualizations interacting with each other via multi-point high-definition (HD) videoconferencing in scalable tiled display environments.

1. 2 Problem and Approach

The essential problem this dissertation investigates is how to enable the scaling of a system that distributes extremely high-resolution visual content in real-time from multiple rendering clusters

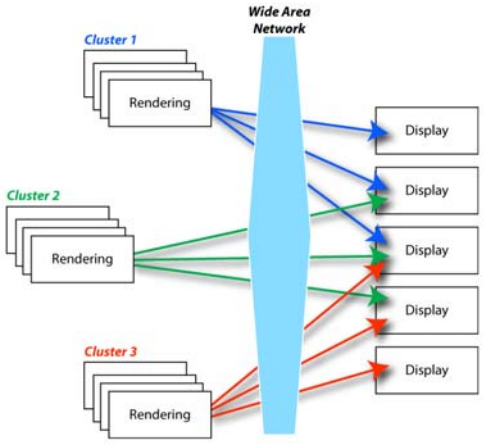


Figure 3. Distributed visualization

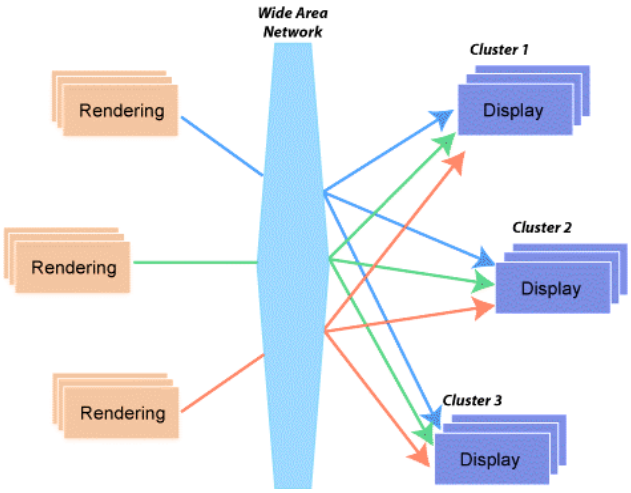


Figure 4. Visualcasting scenario

to multiple tiled displays in order to enable distant collaboration with multiple endpoints using high-resolution tiled displays. Figure 3 shows a distributed visualization scenario supported by SAGE, which involves running visualization applications at multiple remote rendering clusters and streaming their pixels to one big tiled display. Figure 4 shows a Visualcasting scenario including multiple collaboration endpoints and remote rendering clusters.

To support dynamic resizing and repositioning of visualization application windows on the tiled display wall, SAGE needs to dynamically repartition image data and stream each partition to an appropriate tile or display node. This is called dynamic pixel stream reconfiguration and will be discussed in detail in Chapter 3 [Jeong06]. In the first scenario in Figure 3, each sender partitions application images in only one way. However, in the second model in Figure 4, since each endpoint individually manages application windows on its tiled display, each sender manages independent image partitions and streams for all endpoints. As more endpoints join and the required network bandwidth and computation increase, senders will begin to face a scalability problem.

This problem is solved by transferring pixel data through a high-speed bridging system called SAGE Bridge, which is placed at a core network center in the middle of collaboration endpoints (see Figure 5). A SAGE Bridge acts as a pixel stream duplicator and splitter for Visualcasting. It decouples pixel data duplication and partitioning from pixel rendering so as to prevent rendering nodes from being overloaded by the addition of endpoints. This allows each rendering node to stream full image frames to a SAGE Bridge without considering the window layouts and tiled display configurations of multiple endpoints. The SAGE Bridge software is deployed on a high-performance PC cluster equipped with ten-gigabit network interfaces. The experimental results provided in Chapter 5 show that this approach can scale to support an increasing number of endpoints by allocating an increased number of cluster nodes for the SAGE Bridge. The traditional Internet

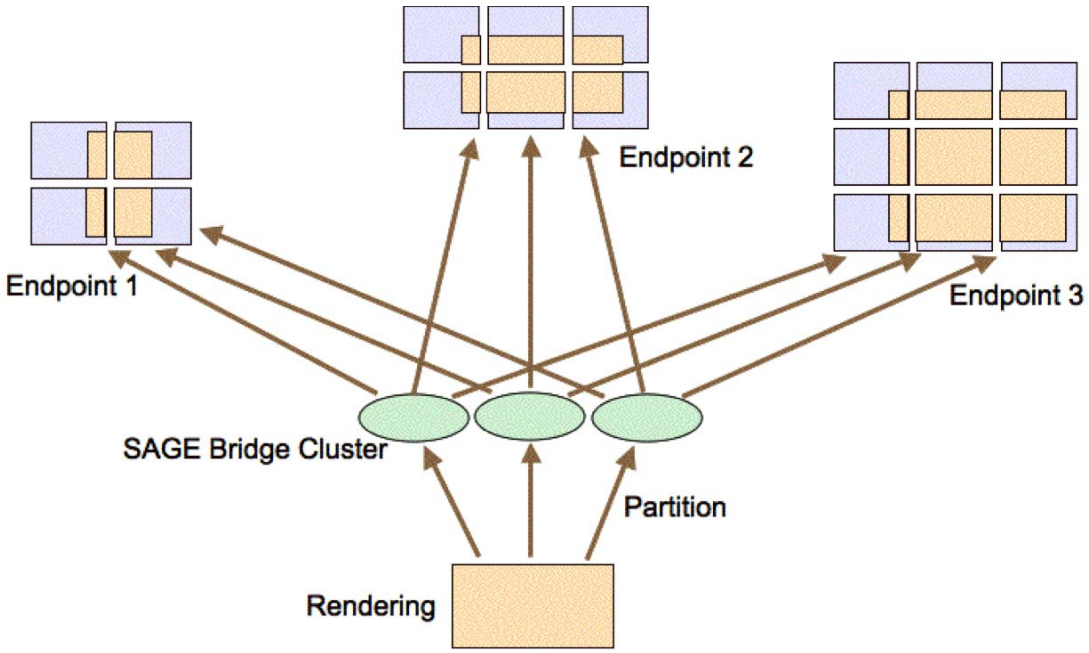


Figure 5. SAGE Bridge approach

Protocol (IP) multicast could be an alternative approach, but it has several limitations that will be discussed in Section 2.2 [Deering91].

An obvious approach to partition the same image data differently for each tiled display (endpoint) involves creating a group of network buffers for each endpoint and copying the appropriate portions of the image to each buffer. However, in this approach, adding a new endpoint incurs a significant system overhead because the required memory capacity for network buffers and the memory bandwidth for copying the image data increas with the number of endpoints. An advanced approach to be used by Visualcasting is called pixel block based streaming. The approach partitions image data into equally sized pixel blocks and calculates their destinations (either tiles or display nodes) for each endpoint. In this approach, senders manage an independent grouping of pixel block pointers for each endpoint rather than a group of network buffers. No additional memory buffers or memory copies are required by the addition of a new endpoint.

Another important problem addressed by Visualcasting is handling the computational resource heterogeneity in network bandwidth and display resolution of endpoints. Due to this heterogeneity, endpoints consume (display) streamed data at different rates. For example, the maximum frame rate of an uncompressed 4Kx2K animation at an endpoint with 6 gigabits per second (Gbps) of network bandwidth is 30 frames per second (fps) while the maximum rate at another endpoint with a 1Gbps of network bandwidth is only 5fps. Visualcasting is able to support these heterogeneous endpoints independently by preventing a slow endpoint from degrading the overall Visualcasting performance. An approach to handle this problem is to drop image frames for slow endpoints at the SAGE Bridge. This technique allows a SAGE Bridge to adaptively stream pixel data at a rate that each endpoint can afford, which permits a different application frame rate at each endpoint.

Visualcasting requires high-performance pixel streaming over wide-area network. User Datagram Protocol (UDP) is chosen as the network protocol for Visualcasting rather than Transmission Control Protocol (TCP) because the former shows much higher and more stable performance over wide-area networks with long round trip time than the latter. However, since UDP is an unreliable network protocol, safe delivery of pixel data is not guaranteed. Pixel data loss may result in significant visual artifacts on the displayed visualization. The SAGE pixel streaming architecture has to be designed to handle the possible data loss and to control pixel data flows so as to minimize data loss. Effective flow control of UDP pixel streams is vital for the heterogeneous endpoint support of Visualcasting because the data transfer rate for each endpoint is determined by the flow control mechanism. The SAGE Bridge decides whether or not to drop image frames based on this rate.

The fundamental research questions that originated from these problems include:

- How to arbitrarily scale simultaneous data distribution to multiple receivers?

- What parameters affect this ‘arbitrarily scaling’ and how do these parameters affect the distribution throughput?
- What happens if the receivers are heterogeneous? Are any special considerations needed?

These questions will be addressed in the following chapters.

CHAPTER 2 RELATED WORKS

This chapter discusses a few of the prior works related to SAGE and Visualcasting. Section 2.1 introduces other parallel and remote rendering systems, and compares them with SAGE. Section 2.2 discusses possible Visualcasting approaches using the traditional IP multicasting technique, and compares them with the SAGE Bridge Visualcasting approach.

2. 1 Parallel and Remote Rendering Systems

There are several existing systems with parallel and remote rendering schemes related to SAGE. The simplest case of remote rendering uses remote desktop systems such as VNC [Richardson98], Remote Desktop Protocol [RDP08] and Xmove [Solomita94]. These systems were designed to transmit a single desktop to remote computers over slow networks and to operate on event-triggered streaming mechanisms. They are not suitable for real-time streaming of high-resolution scientific visualizations or with collaborative applications. Access Grid [Childers00] is a system that supports distributed collaborative interactions over computational grids. Although it enables remote visualization sharing, the major focus of the Access Grid lies in enabling distributed meetings and conferences. Furthermore, the display resolution of remote desktop methods and Access Grid is limited to a single desktop resolution. On the other hand, SAGE can support scalable display walls with a 100-megapixel resolutions and include these systems as SAGE applications by adding the simple SAGE API to them.

Perrine et al. and Klosowski et al. presented the merits of high-resolution display for various visualization applications using the Scalable Graphics Engine (SGE) developed by IBM [Perrine01,

Klosowski02]. SGE is a hardware frame buffer for parallel computers. Disjoint pixel fragments are joined within the SGE frame buffer and displayed as a contiguous image. SGE supports up to sixteen 1GigE inputs and can drive up to eight displays with double-buffering to support display systems of up to 16 megapixels. SAGE and SGE are similar in that they both receive graphics data from multiple rendering nodes and route that data to high-resolution displays.

However, SAGE differs from SGE in that the former is a software approach which is much more flexible and scalable than the latter. Since SAGE does not require any special hardware, new network technologies like 10GigE and other new protocols are easily applied to SAGE. SGE, on the other hand, is bound to 1GigE inputs and the SGE-specific network protocol. There is no theoretical limitation to scaling the performance of SAGE by adding more rendering and display nodes. Conversely, network bandwidth, number of inputs and memory capacity limit the performance of SGE.

There are several parallel rendering systems that can benefit from SAGE or SGE. WireGL [Humphreys00] or parallel scene-graph rendering is a sort-first parallel rendering scheme from a single data source. This approach allows a single serial application to drive a tiled display by streaming graphics primitives that will be rendered in parallel on display nodes. However, it has limited data scalability due to its single data source bottleneck. Flexible scalable graphics systems such as Chromium [Humphreys02] or Aura [Germans01] are designed for distributing visualizations to and from cluster driven tiled-displays. However, since these systems enable only one application at a time with a static layout on a tiled display, they require a graphics streaming architecture such as SAGE or SGE to move, resize and overlap multiple application windows.

XDMX (Distributed Multi-head X11) [DMX04] is another system that can drive a tiled display. It is a front-end proxy X server that controls multiple back-end X servers to make up a

unified large display. XDMX also can support Chromium to display multiple applications on a tiled display. However, XDMX does not support parallel applications. This limits its scalability with respect to large datasets.

No other systems discussed so far were designed to stream graphics data over a high-speed wide-area network. In contrast, SAGE has a UDP-based high-speed pixel streaming architecture for wide-area networks that have multi-ten gigabits of network bandwidth. The architecture is open so that it may use new streaming protocols designed for high-bandwidth and high round-trip time networks that are not considered in the streaming architectures of SGE and Chromium. In addition, SAGE considers the mullions (borders) of each LCD panel of tiled displays when displaying application windows. Hence, the mullions appear to be placed on top of a large continuous image. This feature was considered neither in SGE or Chromium.

TeraVision [Singh04] developed by EVL is a scalable platform-independent solution that is capable of transmitting multiple synchronized high-resolution video streams between single workstations and/or clusters. TeraVision can also stream graphics data over wide-area networks. However, it has a static application layout on a tiled display. It is suitable for streaming a single

Table 1. Comparison between SAGE and other approaches

	SAGE	SGE	XDMX	Chromium	WireGL	TeraVision
Multitasking (Multiple Windows)	Y	Y	Y	-	-	-
Window Reposition and Resizing	Y	Y	Y	-	-	-
Display-Rendering Decoupling	Y	Y	-	-	-	Y
High-Performance WAN Support	Y	-	-	-	-	Y
Scalable Parallel Application Support	Y	Y	-	Y	-	-
Scalable Image Multicasting	Y	-	-	-	-	-

desktop to a high-resolution tiled display but not suitable for supporting parallel applications or multiple instances of applications.

Table 1 compares SAGE with other systems. This table clearly shows that scalable image multicasting (Visualcasting), which is addressed in this thesis is the most unique feature of SAGE. No other approach solves this problem.

2.2 Multicasting Approach

Multicasting is the simultaneous transmission of data to a subset of hosts in the network using efficient strategies to send the data over the network only once. It provides data delivery to groups of hosts with lower network and host overhead than by broadcasting to all hosts or by unicasting to each host in a group [Deering91]. Visualcasting can be defined as a real-time image multicasting service from multiple rendering clusters to multiple display clusters. It is possible to implement Visualcasting exploiting existing multicast techniques, but there are several problems in this approach.

First, generating and managing multicast groups is a complicated problem in Visualcasting because the number and membership of the group are dynamically changed whenever users move or resize visualization windows. A multicast group consists of the end-nodes that receive the same data, but since the visualization image is partitioned differently for each Visualcasting endpoint, the end-nodes receiving the exact same image fragment are rarely found. To generate a multicast group, the system needs to find a group of end-nodes who have a non-empty intersection between the image fragments that they receive, and the image fragment intersections are found by overlapping different image partitions as shown in Figure 6.

Second, since every intersection generated by overlapping the image partitions produces a multicast group, the number of the multicast groups explosively increases with the addition of end-points, the extension of the tiled display dimensions, or dynamic changes in display window layouts. This incurs a scalability problem because of the limited number of multicast IP addresses (i.e. a limited number of multicast groups) are provided for a network.

The last problem with this approach is a long delay for both performing display window operations and the joining of a new endpoint, because dynamic changes of a multicast group membership incur significant latency. For these reasons, a multicast approach is not appropriate for solving Visualcasting problem.

In addition, multicast-based Visualcasting requires very expensive specialized routers or switches in order to support multicast service on the order of multi-ten gigabits/s. Conversely, a

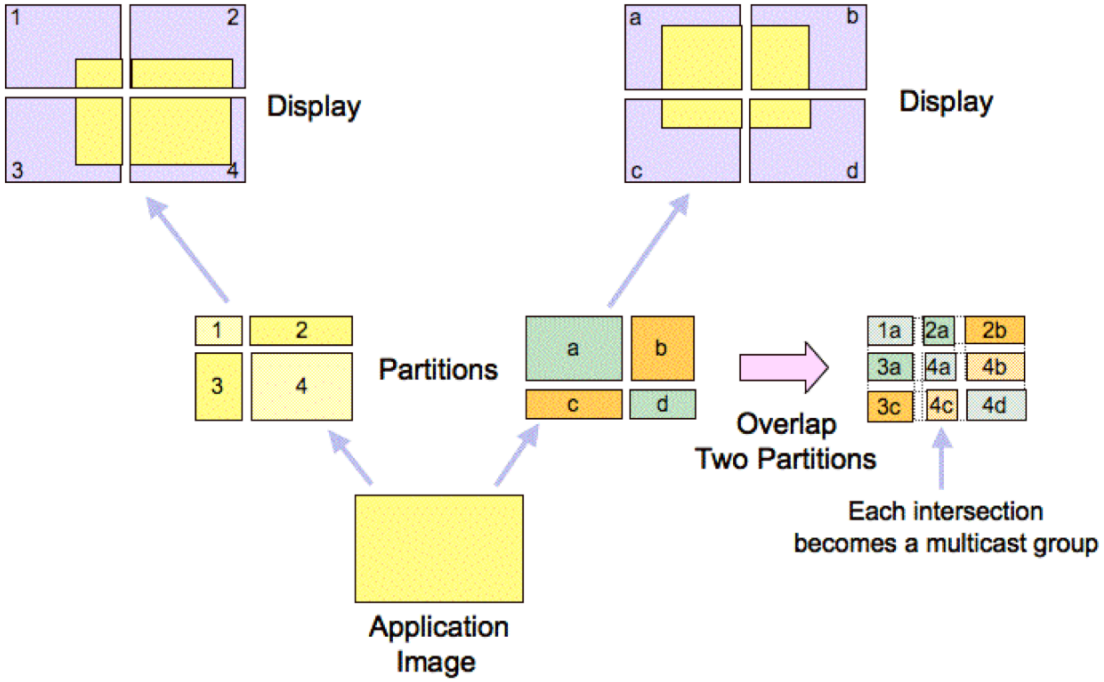


Figure 6. Image partition for multicasting

SAGE Bridge consists of commodity PCs running the SAGE Bridge software. Although a variety of reliable multicast techniques exist, low-latency reliable multicast on the order of tens of gigabits per second is an unsolved problem and an active area of research within the Grid community [Burger05].

Layered multicast [McCanne96] is a possible approach to resolve heterogeneous endpoint issue discussed in Section 1.2. This idea consists of processing source images to generate multiple image versions with differing levels of image quality and streaming each version as a separate multicast layer. Then, each endpoint selectively chooses the layers of the stream that are appropriate to the endpoint considering its network bandwidth and display resolution. However, this approach may place an excessive computational load on senders, which may result in non-trivial system performance reduction.

The SAGE Bridge Visualcasting approach is similar to application-layer multicast in its basic idea and advantages over IP multicast [Jannotti00, Banerjee02]. Since both Visualcasting and application-layer multicast duplicate data on the computing nodes instead of on multicast-enabled routers or switches, they can be easily deployed on conventional networks, while large parts of the Internet are still incapable of IP multicast. Application-layer multicast approaches, however, are typically designed for low-bandwidth data streaming applications with large receiver sets [Banerjee02]. On the other hand, SAGE Bridge Visualcasting software is designed for high-bandwidth, large-scale data distribution for multiple tiled display clusters.

CHAPTER 3 SAGE ARCHITECTURE

SAGE is the infrastructure for Visualcasting, which provides essential capabilities including high-performance real-time streaming of image data between visualization clusters, dynamic reconfiguration of the image streams, synchronization among visualization cluster nodes, centralized control of local and remote cluster nodes, and audio data streaming. This chapter describes the SAGE architecture to implement these essential capabilities for Visualcasting. Specifically, Section 3.1 shows an overview of the SAGE architecture and introduces the hardware and software components of SAGE. Section 3.2 and 3.3 highlights the functions of SAGE components following the SAGE starting procedure and the pixel pipeline stages. Section 3.4 discusses the problem and approach of dynamic pixel stream reconfiguration. Section 3.5 discusses the user interaction with SAGE. Section 3.6 describes the audio data streaming architecture in SAGE.

3. 1 Overview

The hardware environment where SAGE runs comprises a scalable tiled display cluster, distributed rendering clusters and high-speed networks that fully connects the cluster nodes. Figure 7 illustrates a SAGE session running on this environment. Multiple visualization applications run on distributed rendering clusters. SAGE captures their output images and independently streams and displays them on a scalable tiled display. The Free Space Manager (FSManager) is the window manager of SAGE which is akin to a traditional desktop manager in a windowing system, except that it can scale from a single tablet PC screen to a desktop spanning a large tiled display. It controls image streams between rendering clusters and the tiled display cluster in response to various user commands such as a window move, a window resizing, or z-order change. The Application Launcher

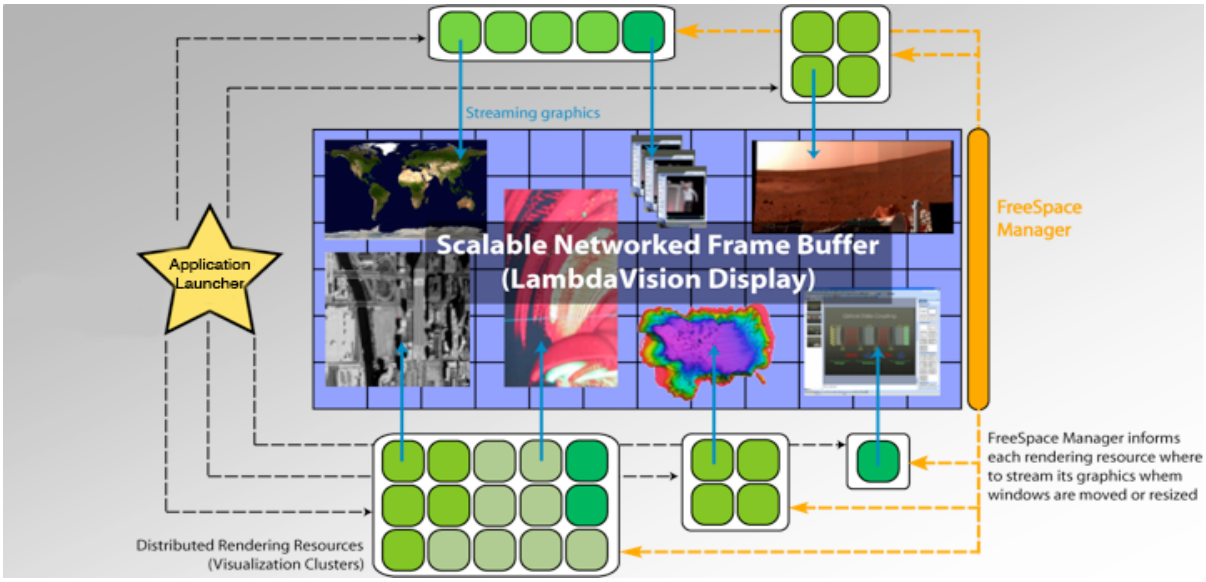


Figure 7. An example of SAGE session

(AppLauncher) allocates rendering cluster nodes to visualization applications and then launches them. The FSManager typically runs on the master (control) node of a tiled display cluster, and the AppLauncher runs on the master node of a rendering cluster. However, it is possible to run them on any machine that has network connectivity to all cluster nodes under their control.

Figure 8 shows the software components of SAGE running on the hardware components. In addition to the FSManager and the AppLauncher introduced above, visualization applications use SAGE Application Interface Library (SAIL) to send their output pixel data to a tiled display. Any application with uncompressed pixel output can be easily ported to SAGE by adding ten to twenty lines of SAIL API code. In order to support parallel visualization applications where each rendering node will generate a portion of the whole picture, the SAIL API allows application programmers to describe output image buffers (width, height, pixel format and so on) and the position of the buffers in the whole application image.

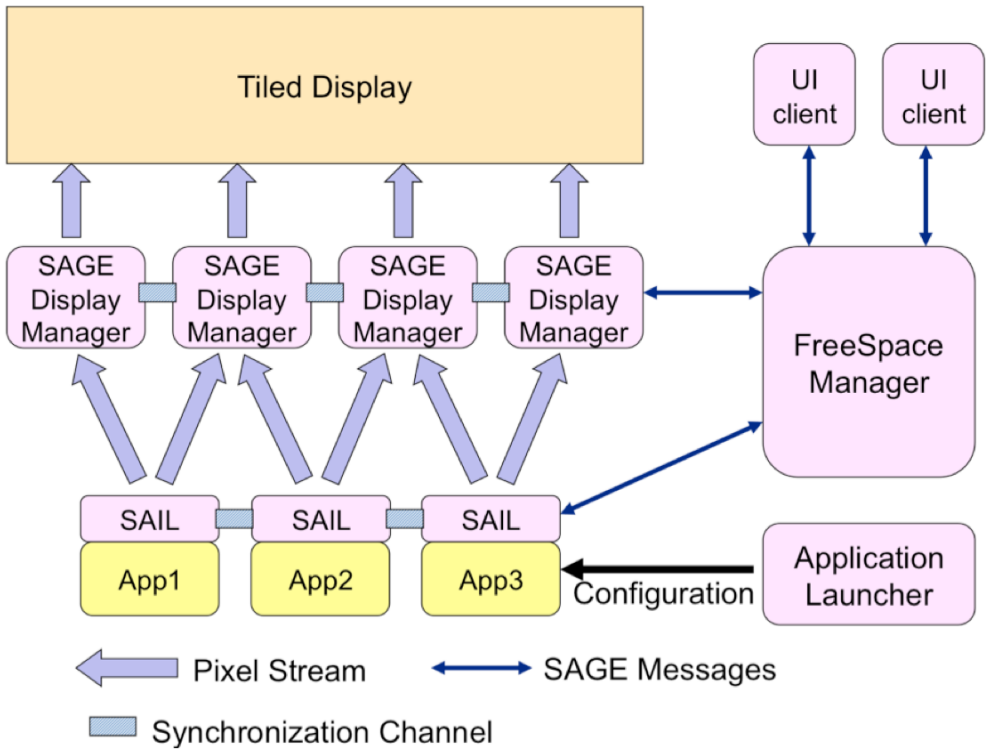


Figure 8. Software components of SAGE

SAGE Display Manager is a pixel stream receiver running on each display node. It supports simultaneous display of multiple SAGE applications on a tiled display by receiving multiple independent pixel streams from each application. Whenever users move or resize an application window, the FSManager updates the new window information to SAIL and SAGE Display Managers so that the application imagery can be displayed on any part of the tiled display as users want. SAGE UI allows users to launch, move and resize SAGE applications by sending user interaction commands to other SAGE components and informs users of various application information including application name, window attributes, and performance data. The SAGE audio manager plays audio data streamed from SAIL that is synchronized with visual data.

SAGE is developed on the Linux operating system and ported to Mac OS X, Sun Solaris and Microsoft Windows. OpenGL [Woo99] and Simple DirectMedia Layer (SDL) are used in order to display images on a screen [SDL06]. Posix Thread (pthread) is used for multi-threading in numerous

Table 2. SAGE tiled display configuration

< Tiled Display Parameters >
Dimensions : the number of rows and columns of the tiled display
Mullions : the left, right, bottom and top width of screen borders in inches
Resolution : the screen resolution of each tile
PPI : pixels per inch
Machines : the number of the tiled display cluster nodes

parts of SAGE. QUANTA, a cross-platform adaptive networking toolkit [He03], is used to deliver the control messages to other SAGE components.

3. 2 Starting Procedure

During the starting procedure, SAGE components are launched by users or another SAGE component and configured by user-defined parameters. The control channels are created between the FSManager and other components and the data channels are created and configured between the application (SAIL) and the receivers (SAGE Display Manager) so that SAGE becomes ready to stream high-resolution image frames.

The first step of starting SAGE is launching the FSManager. It reads various configuration parameters from files, launches a SAGE Display Manager on each tiled display cluster node according to the parameters, establishes control channels to SAGE Display Managers, and distributes necessary configuration information to them.

The configuration read by the FSManager includes IP addresses and port numbers used for control and data connections, buffer sizes for image streaming, and tiled display parameters listed in Table 2. Figure 9 shows an example of a tiled display configuration and the virtual desktop generated from it. The FSManager retains the size and position of each screen in pixel coordinates with its origin at the lower-left corner of the whole tiled display. The mullion (screen border) width in inches is converted to the number of pixels. In this example, the mullion width in pixel numbers is 0.6 inch x 90 ppi = 54 pixels. This number is used for calculating the coordinates of each screen in Figure 9. Based on the virtual desktop information, the FSManager launches SAGE Display Managers and sends them initialization messages. Once the initialization of all SAGE Display Managers is complete, the FSManager and SAGE Display Managers wait for connections from SAGE applications.

The second step is starting the AppLauncher on a rendering cluster unless it is already running on it. The AppLauncher reads a user-defined SAGE application configuration file that includes the list of SAGE applications available on the cluster and various application parameters for SAGE in Table 3. An application can have many different configurations and the available application list is forwarded to the SAGE UI.

```
TileDisplay
  Dimensions  2  2
  Mullions  0.6  0.6  0.6  0.6
  Resolution  1600  1200
  PPI  90
  Machines  2

DisplayNode
   IP   192.168.2.1
   Monitors  2  (0,0)  (1,0)

DisplayNode
   IP   192.168.2.2
   Monitors  2  (0,1)  (1,1)
```

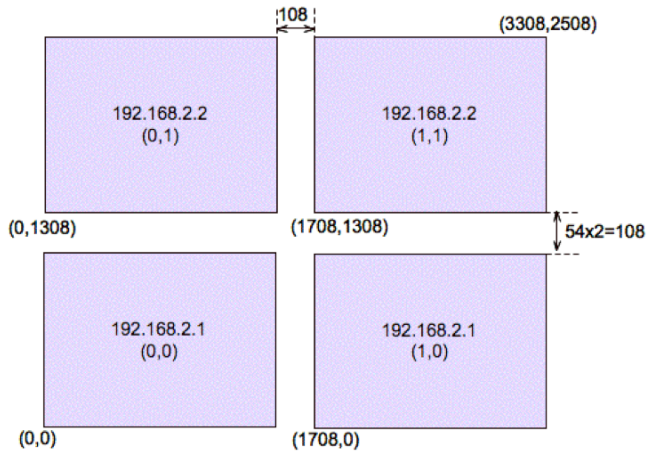


Figure 9. An example of SAGE tiled display configuration

Table 3. Application parameters for SAGE

BridgeOn : enable/disable Visualcasting
BridgeIP/Port : IP address and port number of SAGE Bridge
PixelBlockSize : width and height of a pixel block
GroupSize : the size of a pixel block group
NwProtocol : the network protocol to be used for the streaming of this application (TCP/UDP)
FrameRate : the desired frame rate of this application that is used for the network flow control
StreamType : the type of synchronization and the selection of sending and receiving methods

The third step is starting SAGE UI and connecting it to the FSManager and the AppLauncher. SAGE UI fetches the tiled display information from the FSManager and draws the outline of the display. According to the application list from the AppLauncher, the application icons are drawn below the display outline. When a user selects one of these icons, the list of available configurations for the application is displayed. By selecting a configuration, the application is launched by the AppLauncher.

The fourth step is the starting and connecting of the application to the FSManager. In the initialization phase of the application, a SAIL object is created inside the application. The object is initialized by the application configuration selected by the users. The IP address and the port number of the FSManager are included in the configuration and used for creating a connection to the FSManager. It creates a data object for managing the application information and sends the receiver (SAGE Display Manager) information to the SAIL object.

The fifth step is establishing pixel data channels and configuring pixel streams on them. The SAIL object connects to the receivers running on the tiled display using the information (IP addresses and port numbers) of the receivers from the FSManager. The new connections themselves work as

pixel data channels when the TCP protocol is used, or the connections are used for creating pixel data channels when the UDP protocol is used. The FSManager generates stream information containing the image partitions and the destinations of the application. The stream information is delivered to SAIL and used to configure new streams on the pixel data channels. The details of the configuration are discussed in the following sections.

After these five steps the pixel data streams are started from the application to the receivers, and the application imagery is shown on the tiled display. The next section discusses the pipeline stages for the pixel data delivery.

3. 3 SAGE Pixel Pipeline

Figure 10 shows the five pipeline stages of SAGE. In the data fetch stage, an application writes pixel data on a SAGE frame buffer in the SAIL object. Pixel blocks are generated from the SAGE frame buffer, and the blocks are grouped according to their destination (tiled display node) in the block generation stage. Grouped pixel blocks are streamed to a display node in the block transfer stage. The SAGE Display Manager running on the node receives the pixel blocks and inserts them into the pixel block buffer in the block read stage. The SAGE Display Manager also fetches and downloads the pixel blocks from the buffer into graphics hardware. Downloaded pixel data is updated

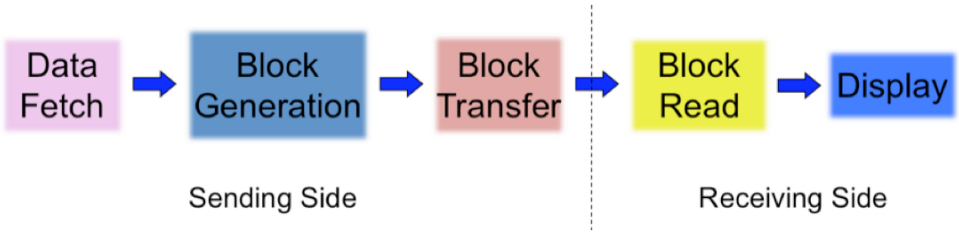


Figure 10. SAGE pipeline

on a screen synchronized with the adjacent screens in the display stage. Each stage has its own thread and is connected to the next stage by various buffers: double frame buffers, pixel block buffers and network buffers.

3.3.1 Data Fetch Stage

The SAGE Application Interface Library (SAIL) has double frame buffers to store application output images and to connect the data fetch stage and the block generation stage. One buffer is used at data fetch stage for storing new pixel data, and the other is used at the block generation stage for generating pixel blocks from it. Two buffers are swapped when both stages are ready to proceed to the next frame. This occurs when writing an image frame on one buffer is done at the data fetch stage, and pixel block generation from the other buffer is done at the block generation stage.

There are two ways to fill the frame buffers. One is passing the address of an application output buffer to SAIL and copying data from the application buffer to a frame buffer. The other is providing the application with the address of a frame buffer so that the application directly writes image data on it. The recent version of SAGE uses the latter. It has an advantage of avoiding a memory copy inside SAIL and using less memory but requires more changes in application code.

Let us call the two frame buffers as buffer A and buffer B. Once an application writes a full image frame on buffer A, it calls SAIL swapBuffer() which passes buffer A to the block generation stage and makes buffer B available to the application. But if buffer B is still in use at the block generation stage, the caller (application thread) is blocked inside the function call until the block generation thread releases buffer B. If the function was a non-blocking call (just return to the caller if

buffer B is still in use), the application would overwrite new frames on buffer A until buffer B is released. One advantage of this method is it does not interfere the execution of application and always sends the newest frame. But it arbitrarily skips application image frames. In the case of a parallel application, this arbitrary skipping ruins synchronization among application nodes. This is the reason why SAIL swapBuffer() call is designed as a blocking call. But it limits the performance of application to the network streaming performance of SAGE.

3.3.2 Block Generation Stage

An obvious approach to distribute image data over multiple tiles (screens) is to partition an image according to the application layout on each tile and stream partitioned image fragments to appropriate tiles. But, rather than sending exact image fragments, the recent version of SAGE generates regularly-sized pixel blocks from an application image and selectively streams them to each tile. The main reason SAGE uses pixel block-based streaming is to support Visualcasting. This is discussed in detail in Section 4.2.

Once a frame buffer filled with new image is swapped into the Block Generation stage, SAIL reconfigures existing pixel streams if new application window layout information is received from the FSManager. A synchronization point is located here for parallel applications so that all application nodes have consistent pixel stream configuration. This synchronization is essential for seamless application window repositioning and resizing. Without this synchronization, parallel application streams may be reconfigured at a different image frame at each rendering node. This would result in broken imagery on the display during the window operation. To guarantee that the streams reconfigure in the same image frame, SAIL delivers a stream with a reconfiguration message attached

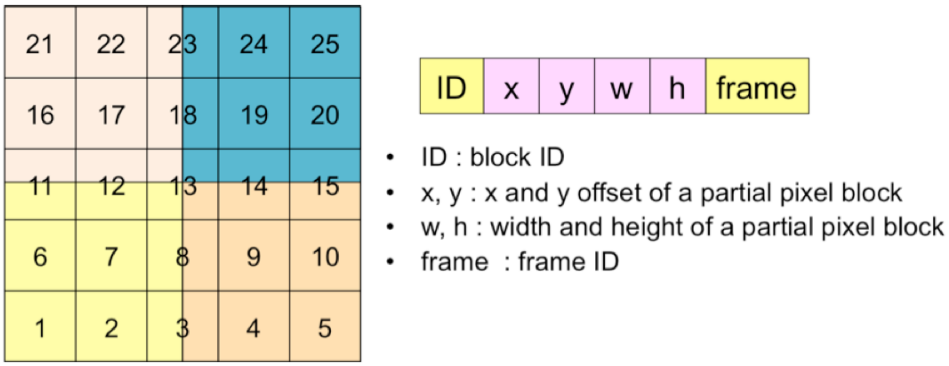


Figure 11. SAGE pixel block generation

to a synchronization signal to each rendering node. Then pixel blocks are generated from the frame buffer, grouped according to the pixel stream configuration and passed to the block transfer stage.

Figure 11 shows an example of pixel block generation and the data structure of a SAGE pixel block. A parallel application runs on four rendering nodes, and its output image stored on four frame buffers is partitioned into an MxN (5x5 in Figure 11) array of pixel blocks with uniform width and height, with the exception of blocks on the edges of the frame buffers. This pixel block array is consistent for every application frame. Each block has a unique ID called blockID that is determined by the location of the block in the application output image. It is calculated by the following equation.

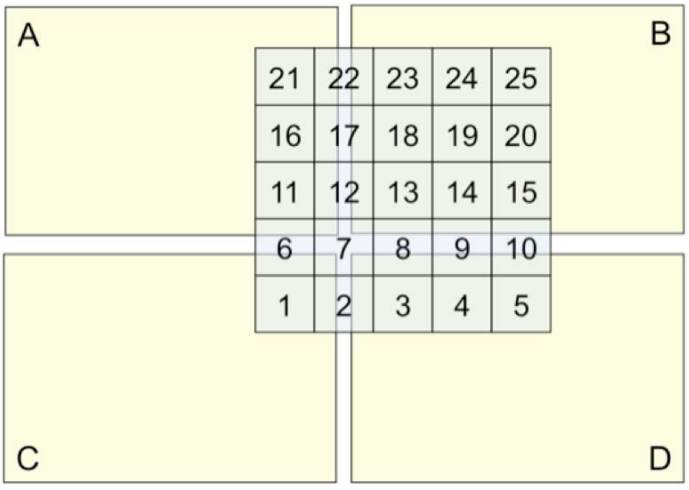
$$blockID = x\_index + y\_index \times M$$

Here x_index and y_index are the x and y indices of the MxN pixel block array. X_index increases from left to right, and y_index increases from bottom to top. On the other hand, x_index and y_index can be calculated from blockID. A display node can calculate the position (x, y) of a pixel block in the application image from its blockID assuming it knows the value of M, the width and height of the pixel blocks, and the row order of the application image which is bottom to top.

$$x\_index = blockID \bmod M$$
$$y\_index = \frac{blockID}{M}$$
$$x = x\_index \times block\_width$$
$$y = y\_index \times block\_height$$

The pixel block position is pre-calculated and stored in a table together with blockID. When a display node locates incoming pixel blocks on its frame buffer to be displayed, the block position in the table is referenced by blockID. The blockID to block position mapping is valid for every image frame of the application. In this way, the display node gets a pixel block position from the blockID without requiring additional runtime calculation.

Also, the geometric calculation results for a pixel block are stored in a table with its blockID and reused later. A good example is the grouping of pixel blocks in order to distribute the pixel data over a tiled display. A lot of geometric calculation (e.g. coordinate comparison) is required to generate the pixel block groups. However, the grouping result is stored with a blockID and reused for many frames since the pixel block groups remain the same until the application window is repositioned or resized. Figure 12 shows an example of the pixel block grouping.



1	C	11	A
2	C,D	12	A,B
3	D	13	B
4	D	:	
5	D	:	
6	A,C	21	A
7	A,B,C,D	22	A,B
8	B,D	23	B
9	B,D	24	B
10	B,D	25	B

Figure 12. Application image partition and pixel block group map

Reconfiguring a pixel stream means the regeneration of pixel block groups according to a new application window layout on a tiled display as shown in Figure 12. Once SAIL receives a window layout message from the FSManager, it regenerates pixel block groups and stores the result in a pixel block group map as shown on the right-hand side of Figure 12. Whenever a new block is generated, it is inserted into a block group according to the pixel block group map. The pixel blocks that are displayed on multiple tiles (e.g. block 7 in Figure 12) belong to multiple groups. This process is repeated for the following image frames until the pixel stream reconfiguration happens again. Each block group has a different destination (a tiled display node) and is sent over a separate stream in the block transfer stage.

In the case of a parallel application, a special case happens in the pixel block generation. A pixel block can be located over multiple application nodes. For example, pixel block 13 in Figure 11 is partially generated by every node. Each node sends a partially filled block to a display node. It receives four partial blocks covering a different portion of pixel block 13. In order to locate the partial blocks correctly, each block needs to carry additional information. The fields x_offset, y_offset, width and height (see Figure 11) in the pixel block header provide the necessary information. The frame ID in the header is used for synchronizing the display of pixel blocks over multiple tiles.

3.3.3 Block Transfer Stage

Pixel block based streaming is suitable for dynamically distributing pixel data over a tiled display. However, sending one big data buffer at a time results in better network streaming performance than partitioning the data into smaller blocks and sending them one by one. In the block transfer stage, SAIL collects the pixel blocks to be sent over a network stream and creates pixel block

clusters. By using iovec[1] data structure, a pixel block cluster is sent by a system call. This achieves higher network performance than streaming block by block. The size of a pixel block cluster is a user parameter that has a significant impact on SAGE network streaming performance. Since every pixel block in a cluster should have the same frame ID and the number of pixel blocks for an image frame is not always a multiple of the size of a pixel block cluster, some clusters can have lower than normal cluster size. Once a pixel block cluster is created, it is inserted into the network stream buffer.

SAGE uses both TCP and UDP as its network streaming protocol. While TCP has its own flow control mechanism, application level flow control is required for UDP to prevent data loss. SAIL has a network streaming thread for UDP flow control. This thread has a network transfer loop that fetches a pixel block cluster from a network steam buffer and sends it over a network. It monitors the average data transfer rate of every network stream. Each stream has a target data transfer rate derived from the frame rate of the application. The priority of a network stream is determined by the ratio of the target data transfer rate to the current average data transfer rate.

The stream with the highest priority among the streams having at least one pixel block cluster in its network stream buffer is selected to send a pixel block cluster in each path of the network transfer loop. But if the average data transfer rate of the selected stream is larger than the target data transfer rate of the stream, no data is transferred for the path. This pixel data transfer algorithm achieves the fairness among network streams by giving the highest priority to the stream most lagging behind from its target data transfer rate. The algorithm controls the data flow of network streams so as not to exceed their target data transfer rate by temporarily suspending the flows. This reduces the possibility of data loss at the receiving ends and at network components in the middle.

[1] A Unix data structure to access scattered buffers in a system call. It has starting addresses and sizes of the scattered buffers.

At the end of the network transfer loop, the total data transfer rate from SAIL is checked at each global flow checking interval. If the rate exceeds the maximum network bandwidth of the application node, the loop is suspended until the rate goes down below the maximum network bandwidth in order to prevent data loss at the node. Table 3 shows pseudocode to implement this algorithm.

Table 4. Pixel data transfer algorithm

```
TDTR : Target Data Transfer Rate
ADTR : Average Data Transfer Rate
GFCI : Global Flow Checking Interval
GDTR : Global Data Transfer Rate

Network Transfer Loop {
  MaxPriority = 1
  For each network stream i  {
    Calculate ADTR for i
    Priority = TDTR[i] / ADTR
    If (MaxPriority < Priority) {    // find the stream with maximum priority
      SelectedStream = i
      MaxPriority = Priority
    }
  }

  if (MaxPriority > 1) {
    Send a pixel block cluster over SelectedStream
  }
  else
    Release CPU     // Suspend network transfer to reduce ADTRs

  if GFCI is reached {    // Controlling total data transfer rate
    Calculate GDTR
    If (GDTR > Max. Bandwidth)
      Release CPU
  }
}
```

3.3.4 Block Read Stage

A SAGE Receiver in a SAGE Display Manager receives multiple independent pixel streams from each application at the block read stage. A SAGE Receiver is created for each application and it has its own thread. It is blocked until new data arrives in one of the incoming streams and also synchronizes those streams from a parallel application. It reads a pixel block cluster from an incoming stream and inserts it into the SAGE block buffer. The pixel block cluster is split into individual blocks in the next stage (the display stage). Another important roll of a SAGE Receiver is checking the last pixel block of an image frame and inserting a special control block into the SAGE block buffer. This helps the pixel downloader in the next stage easily detect the end of an image frame.

3.3.5 Display Stage

The main problems in designing the display stage are how to efficiently display multiple application image streams at different rates on a screen and how to synchronize these streams with the associated streams displayed on neighboring screens. To address the first problem, a SAGE Display Manager creates a couple of OpenGL textures for each application, fetches pixel data from a SAGE block buffer, downloads it onto them and then draws a rectangle mapped with the textures for each application. While one of the textures (the front texture) is used for drawing the rectangle, the other texture (the back texture) becomes the target of newly downloaded pixels. Once a new image frame is fully downloaded on the back texture, two textures are swapped and the newly downloaded image is drawn on the screen.

When a new application image is drawn on the screen, the back frame buffer of the graphics card is cleared completely, all application images on the buffer are redrawn and the frame buffer is

swapped onto the screen. Since the texture swapping happens at a different rate for each application, an application image often has to be redrawn not because the image itself is updated but because another application image on the same screen is updated. Though the overhead for redrawing the image that is already downloaded on a texture is minimal, the SAGE screen refresh rate can increase far exceeding that of a physical monitor.

For example, if two applications are running at the frame rate of 60fps on a screen, the SAGE Display Manager may try to refresh the screen at the rate of 120Hz in the worst case though the physical screen refresh rate of the monitor is just 60Hz. If the swap buffering of the graphics card is synchronized with the screen refresh rate of the monitor, i.e. the swap buffering call is blocked until the screen is actually refreshed, a big performance overhead is incurred in the case of this example. To periodically check if any application image is updated and to refresh the screen if necessary is the more performance-efficient approach than to refresh a screen whenever an application image is updated.

However, the texture swapping and screen refresh on a display node should be synchronous with those operations on its neighbor. So the periodical check for image updates should be performed globally at the synchronization server rather than locally on each node. The synchronization server periodically sends every display node synchronization signals together with a message that indicates whether each application is ready to swap the textures, i.e. whether the application image is updated. Each display node swaps the textures of the applications displayed on it or waits longer according to the message in the synchronization signal.

A typical method to synchronize the texture swapping (image buffer update) and the screen refresh is to place one synchronization point before both the texture swapping and another synchronization point before the screen refresh. When the display thread reaches the texture swapping

synchronization point, it sends the first synchronization update to the synchronization server and waits for a synchronization signal from the server. It broadcasts the synchronization signals once it receives an update from the every display node. The same synchronization procedure is repeated at the screen refresh synchronization point. For the best synchronization result, the display thread has to be blocked while waiting for a synchronization signal.

This method works fine for other tiled display applications that show a single visualization content at a time. In contrast, this method gives a big performance penalty to SAGE since frequently blocking the display thread prevents other application image streams from being downloaded to the graphics card. This frequent blocking is due to the texture swapping of an application stream. To minimize this performance penalty, SAGE does not block the display thread on the texture swapping synchronization point. While waiting for the synchronization signal for an application, the display thread performs pixel data downloading for other applications. The SAGE Display Manager has a separate synchronization signal checking thread that sends a synchronization event to the display thread. Once it receives the synchronization event, it swaps the textures of the applications that are marked as ready for texture swapping in the message of the synchronization signal and proceeds to the screen refresh synchronization point.

Here the display thread is blocked after sending the second synchronization update to the synchronization server and waits for the screen refresh synchronization signal from the synchronization server in order to achieve the best synchronization result. Since the display nodes are already synchronized at the texture swapping synchronization points, the expected blocking time at the screen refresh synchronization point is much shorter than the expected blocking time without the texture swapping synchronization. The minimization of this blocking time is essential for reducing a performance penalty incurred by this synchronization method.

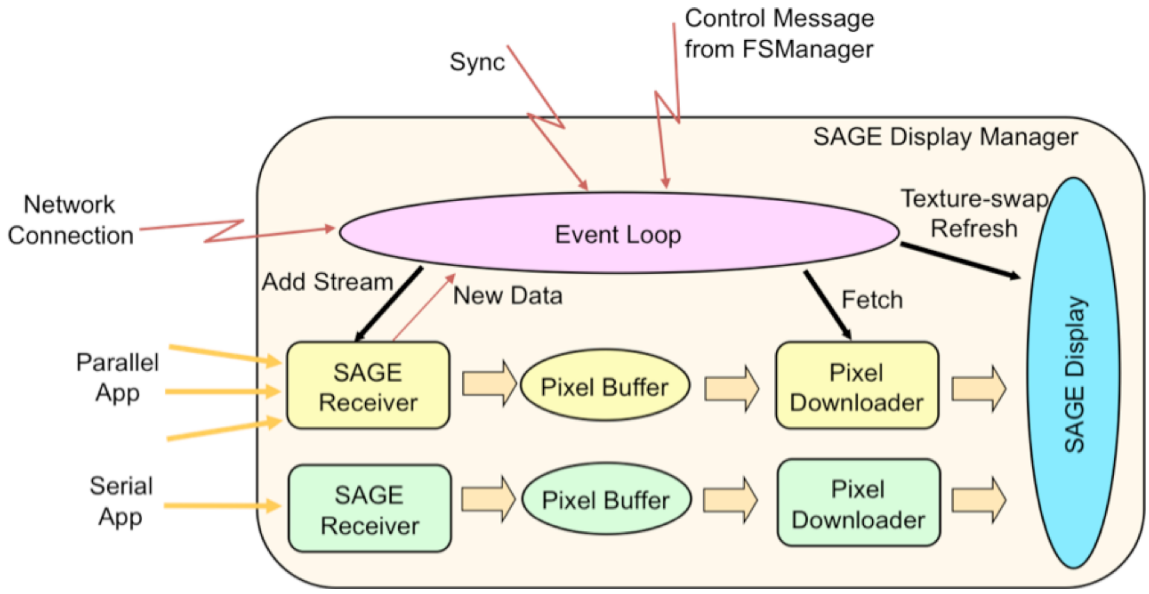


Figure 13. The internal architecture of SAGE Display Manager

Some synchronization jitter is expected at the texture swapping synchronization point since the response for a texture swapping synchronization signal can be delayed in the event queue of the display thread. However, this jitter is invisible on screens and is acceptable for the purpose of minimizing the blocking time at the screen refresh synchronization point. The synchronization jitter at the screen refresh point determines the result of this display synchronization algorithm. Table 4 presents the pseudocode for the synchronization algorithm. Figure 13 shows the architecture of the SAGE Display Manager. In order to reduce synchronization signal latency and to increase its priority, the TCP out-of-band data channel[2] is used for transferring synchronization signals among display nodes.

[2] This channel sends data without waiting for the socket buffer to be filled

Table 5. SAGE synchronization algorithm

```
Pre-condition:
  For all applications, APP.WAIT_FOR_SYNC = false

Display Thread {
  For each application: APP {
     If (not APP.WAIT_FOR_SYNC) {
        Do {
          Fetch and download pixel blocks from the block buffer to the back texture
        } until (a frame is ready on the back texture or the block buffer is empty)

        if (a frame is ready on the back texture) {
          Send FIRST_UPDATE to the sync server
          APP.WAIT_FOR_SYNC = true;    // suspend pixel downloading for this app
        }
     }
  }

  If (a TEXTURE_SYNC arrive) {  // non-blocking sync point
    TEXTURE_SWAP = false
    // an app is active when a part of its image is displayed on this node
    For active APPs that are indicated as Ready in TEXTURE_SYNC {
       Swap back and front texture
       TEXTURE_SWAP = true
       APP.WAIT_FOR_SYNC = false // resume pixel downloading for this app
    }
    If (TEXTURE_SWAP) {
       Redraw application images on the back frame buffer
       Send SECOND_UDPATE
       Wait for SCREEN_SYNC
       Swap the frame buffers
    }
    Else {
       Send SECOND_UDPATE
       Wait for SCREEN_SYNC
    }
  }
}

// The sync server broadcasts sync signals at every sync interval to every display node
// An app is Ready when all nodes where the app is active are ready to swap texture
Sync Server Thread {
  Do {
    Check incoming FIRST_UPDATEs to determine which application is ready
  } Until (next sync interval)
  Write the list of Ready apps on TEXTURE_SYNC
  Broadcasts TEXTURE_SYNC to every node
 Wait for SECOND_UPDATEs from every node
 Send SCREEN_SYNC to every node
}
```

3. 4 Dynamic Pixel Stream Reconfiguration

Since SAGE pixel blocks are regularly sized, the display nodes on which they are displayed and their positions in the screens of the nodes are easily calculated once the application window layout that they belong to is given. This information is stored in a table on each SAGE Display Manager, and incoming pixel blocks are downloaded onto the textures in the SAGE Display Manager referencing this block position table. It needs to be updated whenever the application layout is changed, but the update should be consistent with the update on the pixel block group in the block generation stage. That is, the update should happen exactly when the pixel blocks that were regrouped by the new application window layout arrive at the SAGE Display Manager and are ready to be downloaded to the textures.

In order to make sure that the update happens at the right time, the FSManager sends an application window layout together with a configuration ID to SAIL and SAGE Display Managers, and the pixel blocks streamed from SAIL carry the configuration ID of the application window layout by which they were grouped. SAGE Display Managers compare the configuration ID of the application window layout that it received and that of incoming pixel blocks. If the IDs are matched, it updates the block position table immediately; otherwise, it waits for following pixel blocks (in the case when the pixel block configuration ID is less than the configuration ID of the new application window layout) or newer application window layouts (in the case when the pixel block configuration ID is greater than the configuration ID of the new application window layout).

3. 5 User Interface

SAGE UI Clients can be a Graphical User Interface, text-based console or tracked devices [Krumbholz05], which launch applications using the Application Launcher, send user commands to the Free Space Manager and show the status of SAGE to the users. Any UI client can execute, shutdown, move, and resize SAGE applications in a manner very similar to a typical contemporary windowing system. Furthermore, UI clients can reside on any machine (laptop, tablet, desktop, etc.) connected to the Free Space Manager over any network. Since SAGE is well suited for use in collaborative environments, several tools have been incorporated into the SAGE GUI to facilitate collaborative work. Users could, for example, have discussions and meetings in front of a tiled display where each user is running an instance of the SAGE GUI connected to the same or even different displays. The SAGE GUI captures each user's laptop screen using a VNC server, which pushes the screen over the display using a SAGE-enabled VNC viewer. Using this capability, users can present their problems or achievements to others and share useful information on the display during the discussion.

For basic communication, a chat capability and a list of users currently connected to the display are available from the SAGE UI server managing user connections to every SAGE display. Every user could also be connected to multiple displays at the same time and control applications on any of them. This could prove especially useful when multiple sites are working together. At the end of a meeting, users could save the session and the state of the tiled display so that they can quickly resume their work at a later time.

3. 6 Audio Streaming

Multipoint HD video conferencing is an essential part of Visualcasting. To enable HD video conferencing in the SAGE framework, audio streaming capability is added to the SAGE architecture. SAGE audio capturing capability was implemented as a part of SAIL. The audio enabled SAIL can read audio data from a sound card (microphone), an audio file, or a SAGE application using SAGE audio API functions. The same network streaming modules used for SAGE audio streams are the same as those for SAGE pixel streams. In order to capture and play audio data, an portable cross-platform audio API called Port Audio [PortAudio07] was used. The display of SAGE pixel streams is synchronized with the playing of SAGE audio streams.

CHAPTER 4 APPROACH AND IMPLEMENTATION

This chapter describes the approaches used to implement Visualcasting as well as to solve the problems with Visualcasting. In order to distribute high-resolution images to multiple tiled displays (Visualcasting endpoints) in real-time, a high-performance bridging system called SAGE Bridge was designed and implemented. In order to scale the service to an increasing number of endpoints, a SAGE Bridge requires the SAGE pixel block streaming and dynamic bridging resource allocation scheme. Since display size and network bandwidth can be heterogeneous across Visualcasting endpoints, a SAGE Bridge controls the pixel flows to each endpoint so that it can properly handle the incoming pixel data. For this purpose, a SAGE Bridge may drop image frames at lower-capacity endpoints if necessary. An analytical model for Visualcasting was built to predict the performance of these approaches and was validated by experiments.

4. 1 A New SAGE Architecture with SAGE Bridge

The SAGE Bridge is a new software component of SAGE running on a high-performance PC cluster. With the SAGE Bridge, the SAGE architecture is changed as shown in Figure 14. Multiple SAGE sessions exist in this architecture and each is controlled by a FSManager. The SAGE Bridge is introduced between SAIL and SAGE Display Managers. It intercepts pixel streams from SAIL and duplicates and distributes them for each SAGE session.

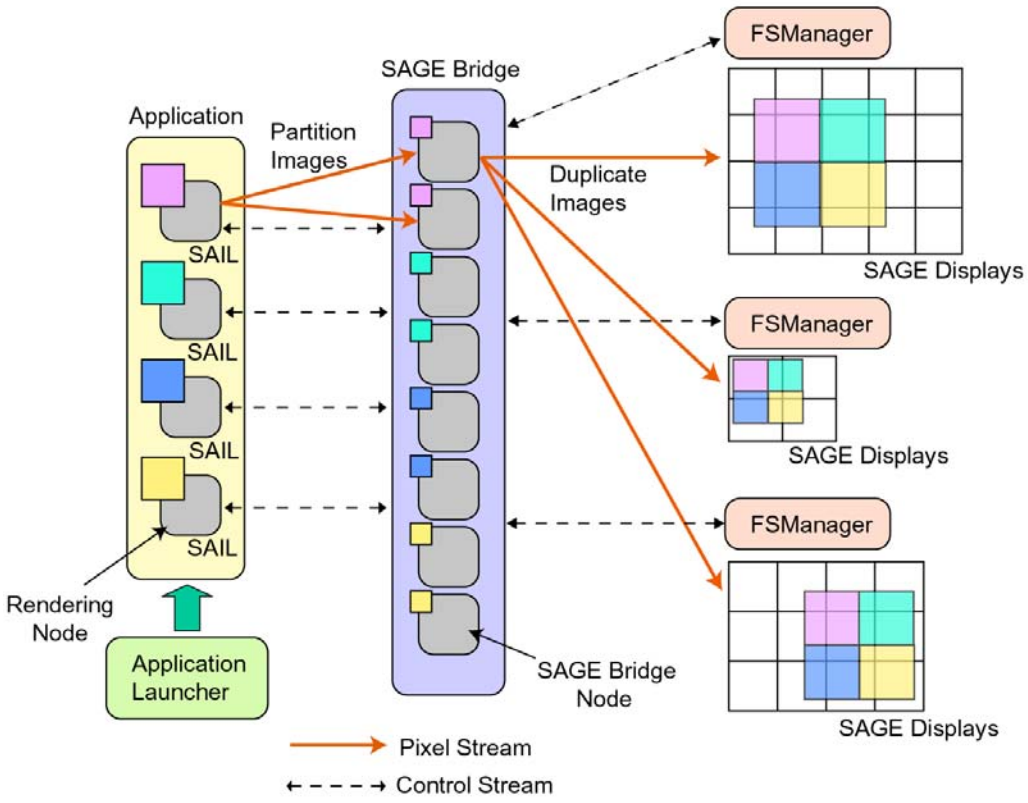


Figure 14. The SAGE architecture with a SAGE Bridge

The introduction of the SAGE Bridge in the SAGE architecture requires a new procedure to execute applications. Figure 15 compares the old and new application launch procedures. The new procedure consists of following nine steps:

(1) A SAGE UI sends commands with application parameters and information about the SAGE Bridge and the first FSManager to the Application Launcher.
(2) The Application Launcher executes an application on the appropriate rendering nodes using information from the SAGE UI.
(3) SAIL creates a control channel with the SAGE Bridge when the application is launched. The SAGE Bridge allocates SAGE Bridge nodes for the application and configures streams between SAIL and the SAGE Bridge.
(4) The SAGE Bridge connects to the first FSManager in order to configure the streams between the SAGE Bridge and the SAGE Displays.

(5) SAIL starts streaming pixels once all configurations are completed.

(6) Application images are displayed in the first SAGE session.

(7) In order to make the second SAGE session join the Visualcasting session, a SAGE UI sends a message that has information about the second FSManager to the first FSManager.

(8) The first FSManager directs the SAGE Bridge to connect to the second FSManager.

(9) The pixel streams between the SAGE Bridge and the second SAGE session are configured and started.

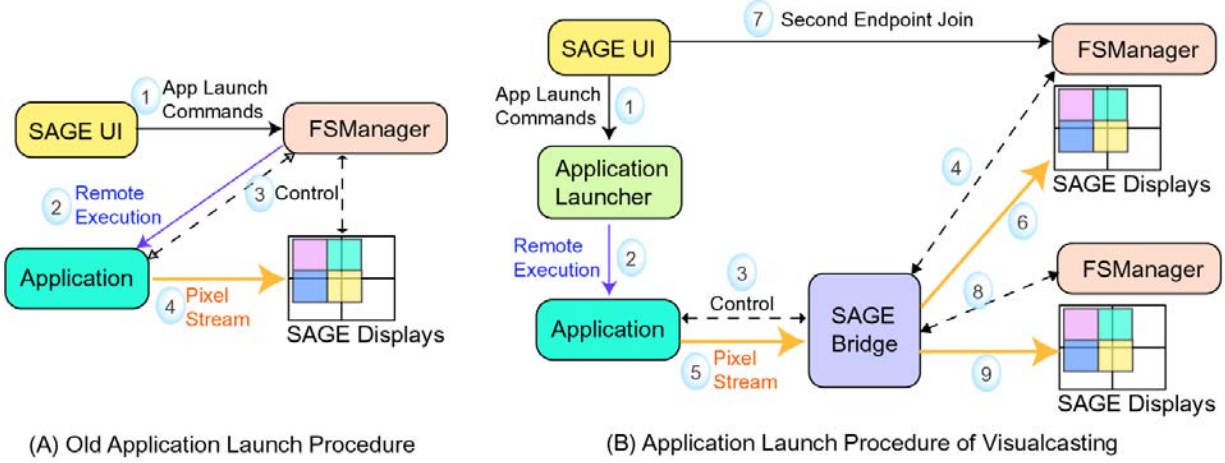


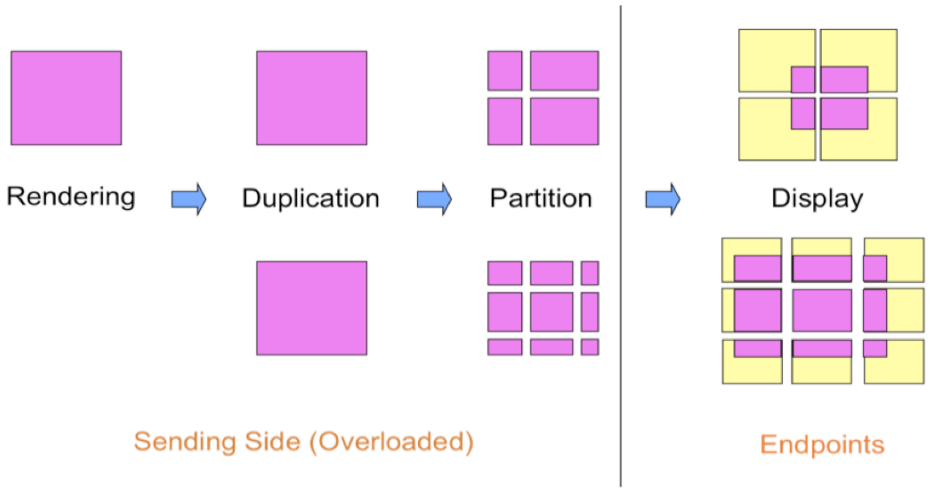
Figure 15. Old and new application launch procedures

In addition to distributing pixel streams, the SAGE Bridge duplicates and sends SAGE audio streams to multiple endpoints. Each application and each endpoint are configured as to whether they are audio-enabled or not. Based on the configuration, the SAGE Bridge may or may not receive and send the audio stream of an application to each endpoint.

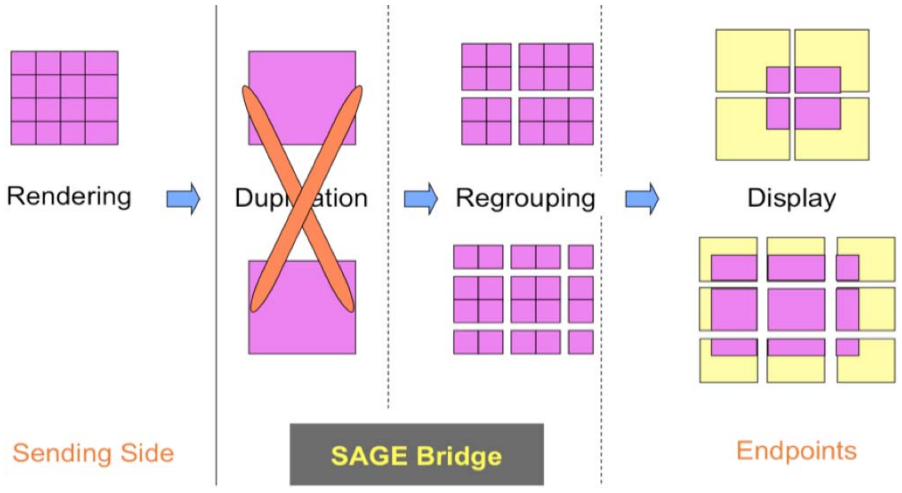
4. 2 Pixel Block Streaming

The initial prototype of SAGE streams application images frame by frame. Each image frame is split into sub-images according to the application layout on the tiled display. Each sub-image is

copied to a network buffer and streamed to a tile. Since the generation of the sub-images depends on the application window layout on a tiled display, Visualcasting requires a different image partition for each endpoint as shown in Figure 16a. However, the required memory space for network buffers and the memory bandwidth for an image data copy continue increasing as the number of endpoints increase. This incurs a significant system overhead in Visualcasting. The approach to resolve this problem is pixel block streaming that was described in Section 3.3.2



(a) Image Duplication and Partition for Multiple Endpoints



(b) Pixel Block Regrouping for Multiple Endpoints

Figure 16. Pixel data distribution for multiple endpoints

With pixel block streaming, the duplication and partition of an image frame at the SAGE Bridge is changed to the grouping of pixel blocks as shown in Figure 16b, which does not require additional network buffers or memory copies when a new endpoint is added to a Visualcasting session. By eliminating these overhead constraints, a SAGE Bridge can scale pixel streaming to increasing number of endpoints until the total data bandwidth of the streams reaches the network or memory bandwidth limit of the bridge node.

4. 3 Bridge Node Allocation

The number of SAGE Bridge cluster nodes is statically configured when the SAGE Bridge starts running. In order to balance the load on each bridge node, the pixel blocks of an image frame have to be equally or nearly equally distributed to each node. The grouping of the pixel blocks for each bridge node does not need to depend on their geometrical location in the image frame because every SAGE Bridge node can stream pixel blocks to an arbitrary location on any tiled display of all endpoints. The recent version of SAGE binds pixel blocks with continuous block IDs as a group.

Though this bridge node allocation strategy is the best for load balancing, an application whose image frame rate is dynamically changing incurs jitters on Visualcasting streams of other applications on the same SAGE Bridge. Another SAGE Bridge node allocation strategy to avoid this problem is allocating a separate bridge node to each application. However, the number of applications to be supported by this strategy is limited to the number of SAGE Bridge nodes. In order to remove this limitation, the master node of the SAGE Bridge monitors the usage of all SAGE Bridge nodes and allocates a new application on the node that has the least usage. In the current implementation, users configure the SAGE Bridge node allocation strategy before they start a Visualcasting session.

4. 4 Supporting Heterogeneous Endpoints

Figure 17 shows an abstracted internal architecture of the SAGE Bridge. A SAGE Receiver receives single or multiple pixel block streams of an application and synchronizes them before pushing pixel blocks into a pixel block buffer. A bridge streamer for each endpoint is created. It reads pixel blocks from the buffer, generates block groups, and streams them to the tiled display at an associated endpoint. The method to be used for generating block groups and streaming them is exactly the same as the method used in the block generation stage described in Section 3.3.2. Once every bridge streamer reads and sends a pixel block, the block is returned to the pixel block buffer for reuse.

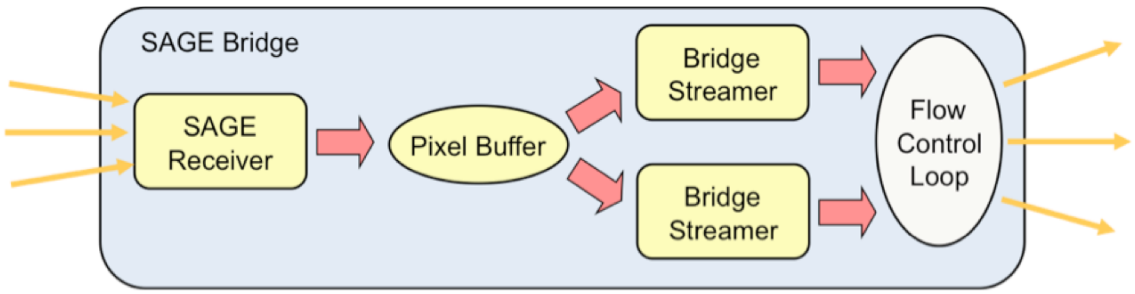


Figure 17. The SAGE Bridge architecture

In an ideal case, every endpoint has enough data bandwidth to afford the incoming pixel block streams. Then the SAGE Bridge's pixel block sending rate to every endpoint is the same as its pixel block receiving rate. However, Visualcasting endpoints do not always have an ideal condition. Each bridge streamer may have a different pixel block transfer rate due to the heterogeneity in the data bandwidth of its endpoint. Since Visualcasting uses the UDP protocol, data loss happens at the endpoint that cannot afford incoming pixel block streams, and then SAGE Display Managers at the

endpoint send the SAGE Bridge a negative feedback. Then the network flow control loop in the SAGE Bridge reduces the pixel block transfer rate to the endpoint. The reduced pixel block transfer rate relieves the congestion on the network streams to the endpoint so as to reduce or stop the data loss.

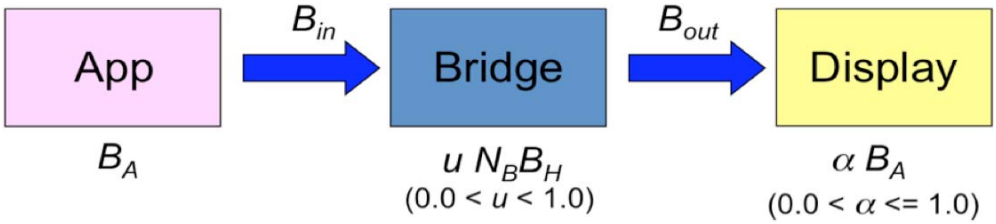
Each bridge streamer has its own read pointer in the pixel block buffer in order to handle temporary difference in the block transfer rate across the bridge streamers. However, the difference is bounded by the size of the pixel block buffer. If the pixel block transfer rate of a bridge streamer is reduced, all bridge streamers are eventually slowed down to match the rate of the slowest one. By dropping image frames, the slowest bridge streamer can catch up with other bridge streamers and stream pixel blocks at its own rate without interfering with other bridge streamers. The number of frames to be dropped is determined by the difference in the image frame transfer rate between the slowest one and the fastest one. The slowest bridge streamer moves its read pointer in the pixel block buffer as if it reads and sends the pixel blocks that belong to the image frames to be dropped. The heterogeneity of endpoints in data bandwidth is handled by this way.

4. 5 Analytic Model of Visualcasting

This section presents an analytic model of Visualcasting performance. The model represents Visualcasting performance in terms of available SAGE Bridge resources, the number of endpoints and applications, available network bandwidth at each stage of Visualcasting pipeline and so on. This model can be used not only for predicting the Visualcasting performance within given system parameters but also for guiding users who want to build a Visualcasting system. For example, the model can answer the following questions which are: (1) how many endpoints can be supported given

a number of SAGE Bridge nodes and the anticipated bandwidth usage of an application; (2) how many SAGE Bridge nodes are required in order to support a given number of endpoints?

Figure 18 shows the abstracted pipeline stages of Visualcasting. The overall Visualcasting performance is determined by the pipeline component that has the minimum data bandwidth. B_A is the bandwidth that an application uses when it runs at a desired frame rate. B_{in} and B_{out} are available network bandwidth to and from a SAGE Bridge cluster. The data bandwidth at the SAGE Bridge cluster is represented as uN_BB_H. For example, if a SAGE Bridge cluster has 5 nodes and each node has a 10Gbps limitation in its network bandwidth and a 70% maximum network bandwidth utilization, the data bandwidth of the SAGE Bridge cluster is 0.7x5x10Gbps = 35Gbps. The data bandwidth at the display stage is determined by the application window layout. For example, if an application window is placed over 4 tiles and each node that drives one of these tiles has 1Gbps of network bandwidth, the data bandwidth is 4Gbps in maximum. However, if the window is shrunk to a single tile, the bandwidth is reduced to 1Gbps. If another application window overlaps the window, the bandwidth decreases even more. In some cases, the bandwidth can be smaller than B_A, and then the Visualcasting performance to the endpoint is limited by the bandwidth that is affected by the application window layout and represented as αB_A.



B_A: expected application bandwidth
B_{in}: network bandwidth application to bridge
B_{out}: network bandwidth bridge to display
N_B: number of bridge nodes
B_H: hardware bandwidth of each bridge node
u : bandwidth utilization
α : application window layout factor

Figure 18. Visualcasting pipeline

Let us assume the network bandwidth to and from the SAGE Bridge cluster and the data bandwidth of every endpoint are enough to afford an application stream at the rate of B_A i.e.

$B_{in} > B_A$, $B_{out} > nB_A$, n is the number of endpoints, $\alpha = 1$ for all endpoints.

If a SAGE Bridge cluster scales the Visualcasting of an application up to n endpoints, the total Visualcasting traffic from the cluster is calculated by the following equation:

$$T = nB_A$$

As the number of endpoints increases, the total Visualcasting traffic reaches the maximum data bandwidth of the SAGE Bridge cluster, and the Visualcasting performance is saturated.

$$T = uN_B B_H$$

The Visualcasting traffic to each endpoint is:

$$T_e = \frac{uN_B B_H}{n}$$

Given the parameters of the SAGE Bridge cluster, the maximum number of endpoints that can be scaled by the SAGE Bridge cluster is:

$$n \leq \frac{uN_B B_H}{B_A}$$

Given the number of endpoints, the minimum number of SAGE Bridge nodes in order to scale these endpoints is:

$$N_B \geq \frac{nB_A}{uB_H}$$

Pre-conditions $B_{in} > B_A$ and $B_{out} > nB_A$ provide users with additional guidance for the application performance and for the number of endpoints of a Visualcasting session given the available network bandwidth to and from the SAGE Bridge cluster.

For example, let us say that one wants to visualcast a compressed 4K animation at 33fps to 10 endpoints. A single stream of the animation uses 1Gbps network bandwidth i.e. B_A= 1Gbps. If a SAGE Bridge cluster consists of machines having a ten-gigabit network interface with 50% of utilization, the minimum number of SAGE Bridge nodes needed in order to support the Visualcasting session is:

$$N_B \geq \frac{nB_A}{uB_H} = \frac{10\text{nodes} \times 1Gbps}{0.5 \times 10Gbps} = 2\text{nodes}$$

The available network bandwidth from the application node to the SAGE Bridge cluster should be more than 1Gbps ($B_{in} > B_A$). The available network bandwidth from the SAGE Bridge cluster to all endpoints should be more than 10Gbps ($B_{out} > nB_A$). The analytical model that has been described in this section was verified by the Visualcasting experiments that will be discussed in the next chapter.

CHAPTER 5 EVALUATION

This chapter describes the Visualcasting tests used to evaluate the performance and to verify the analytic model of Visualcasting. The experimental results showed that the Visualcasting implementation could sustain high-performance pixel data streaming at the rate of multi-ten gigabits per second and scale the streams with an increasing number of endpoints as the analytic model predicted.

5. 1 Visualcasting Testbed

For several years, the Electronic Visualization Laboratory (EVL) has spearheaded a cooperative effort to build the Global Lambda Visualization Facility (GLVF) [Leigh06], a persistent distributed facility aimed at enabling the synergistic research and development of next-generation end-user tools for scientific visualization and collaboration in ultra-high-resolution display environments. GLVF consists of globally distributed high-resolution tiled displays and rendering resources interconnected by a global LambdaGrid (a grid of deterministic high-speed networks). These internationally distributed display-rich environments provided the best testbed for Visualcasting. Figure 19 is the network diagram of the Visualcasting testbed that consists of the following institutions:

- The Electronic Visualization Laboratory at University of Illinois at Chicago;
- SARA Computing and Networking Services in Amsterdam, the Netherlands;
- The School of Information at the University of Michigan;
- Korean Institute of Science and Technology Information (KISTI) in Daejeon, South Korea;

- Gwang-ju Institute of Science and Technology (GIST) in Gwang-ju, South Korea;
- The California Institute for Telecommunications and Information Technology (CALIT2).

Most network links connecting these sites have a 10Gbps of network bandwidth except for two links in South Korea. Four SAGE Bridge nodes are located at a high-performance research network infrastructure called StarLight at downtown Chicago. Each node has a 10gigabit network interface and two dual-core AMD Opteron processors running at 2.2GHz with 4GB of main memory. The total network bandwidth to and from the SAGE Bridge cluster is 40Gbps. Each endpoint has a heterogeneous tiled display dimension and number of display nodes as noted in the network diagram. This provides a good environment to test the heterogeneous endpoint support of the Visualcasting service.

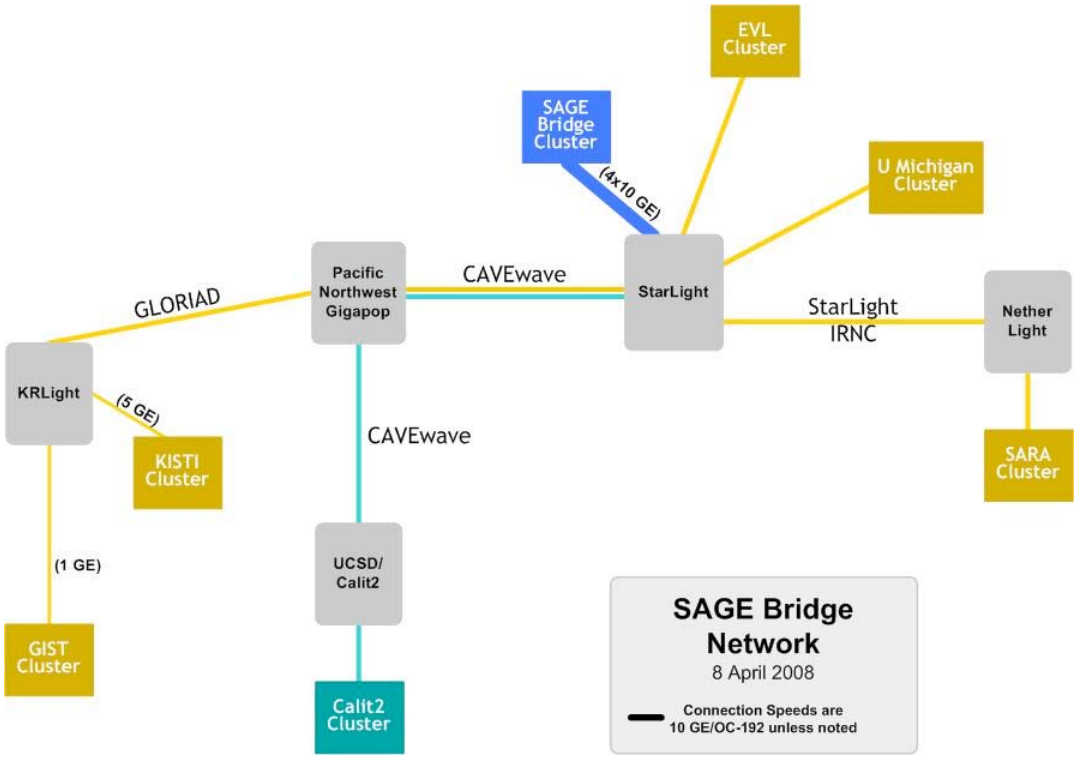
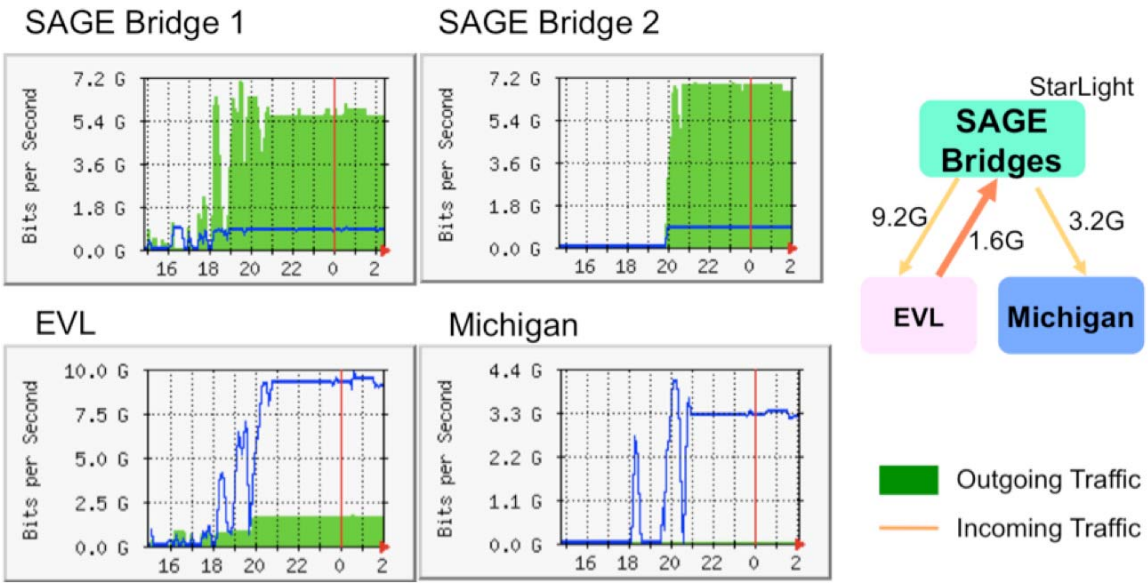


Figure 19. Visualcasting testbed

5. 2 Performance Evaluation

Figure 20 shows a sustained performance test result of Visualcasting for several hours. Two SAGE Bridge nodes placed at StarLight were used for this test. Tiled displays at EVL and the University of Michigan were divided into multiple virtual endpoints to receive Visualcasting streams. Two instances of SAGE benchmark application called “Checker” that streams white pixel buffers ran on two rendering nodes at EVL. Each “Checker” stream used 0.8Gbps network bandwidth. Each SAGE Bridge node received a “Checker” stream and distributed to multiple virtual endpoints at EVL and University of Michigan. The total incoming traffic to the SAGE Bridge cluster was 1.6Gbps and the total outgoing traffic from the SAGE Bridge cluster was 12.4Gbps: 9.2Gbps to EVL and 3.2Gbps to University of Michigan. These results showed that the implemented Visualcasting service could support high-resolution image multicasting at the rate of multi-ten gigabits per second sustaining the rate over five hours.



(The unit of horizontal axis of these graphs is hours)

Figure 20. Sustained Visualcasting performance

Figure 21 shows the result of a Visualcasting scalability test. Almost the same experimental setup was used as that of the sustained performance test. In this case, the tiled displays at EVL and University of Michigan were configured as 16 virtual endpoints. Full-HD (1920x1080) resolution images were streamed to the SAGE Bridge cluster at the rate of 16fps. A SAGE Bridge node was able to scale the Visualcasting streams up to 8 endpoints but was saturated for additional endpoints. By adding another SAGE Bridge node, the Visualcasting streams successfully scaled up to 16 endpoints. These results are consistent with the performance predicted by the analytic model discussed in Section 4.5.

$$T = nB_A = n \times 0.8Gbps$$
$$T_1 = uN_B B_H = 0.7 \times 1 \times 10Gbps = 7Gbps$$
$$T_2 = uN_B B_H = 0.7 \times 2 \times 10Gbps = 14Gbps$$

The total outgoing traffic from the SAGE Bridge cluster T is scaled with the number of endpoints until it reaches T_1 (one bridge node) or T_2 (two bridge nodes). These results show that the

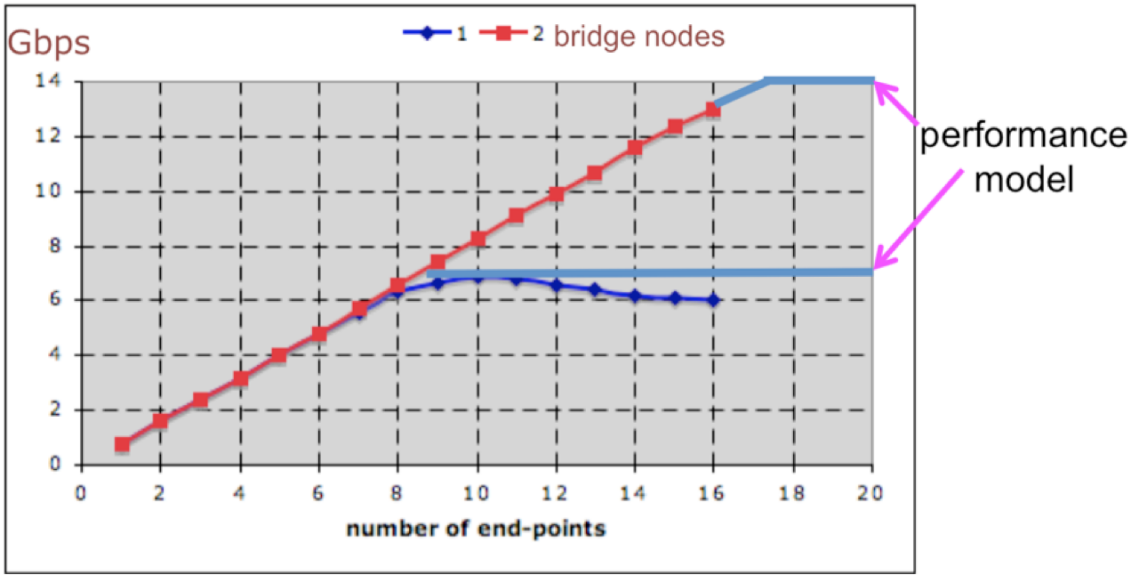


Figure 21. Visualcasting throughput

implemented Visualcasting service can scale the distribution of visualization streams with the number of endpoints as predicted by the analytic model of Visualcasting.

5. 3 Visualcasting Demonstration

The multi-point full-HD video conferencing capability of Visualcasting was demonstrated over the Visualcasting testbed on April 18, 2008 (see Figure 22). The participants of this demonstration included EVL/UIC, SARA, University of Michigan, KISTI and GIST. This showed that Visualcasting enables casual conversation among every participant by its short-latency uncompressed HD video and audio distribution. Each endpoint sent a full-HD camera live-feed and an audio stream to two SAGE Bridge nodes located at StarLight. Whenever a new HD video stream is started, it is allocated to either bridge node according to the current load of each bridge node.

Each HD video stream had a 1920x1080 image resolution and a frame rate of 17~18fps, and 0.7Gbps network bandwidth utilization except for one video stream from SARA that had 6~7fps and

Figure 22. Multi-point HD video conferencing using Visualcasting

having 0.2Gbps network bandwidth utilization. Each endpoint received multiple HD video streams according to its capacity. EVL, University of Michigan, and KISTI received four HD video streams from other endpoints (three 0.7Gbps streams and one 0.2Gbps stream). GIST received the streams from EVL and SARA (one 0.7Gbps stream and one 0.2Gbps stream). SARA received two 0.7Gbps HD video streams. Thus, the total Visualcasting throughput during this demonstration was calculated as follows:

$$(0.7x3 + 0.2)x3 + (0.7+0.2) + (0.7x2) = 9.2Gbps$$

During this demonstration GIST and SARA showed huge image artifacts if they received more HD video streams than the number of streams indicated in the equation above. Though the frame dropping for small-scale endpoints was enabled, the UDP flow control mechanism of Visualcasting was not able to remove data loss at these endpoints. This problem will be investigated more deeply in future research.

At the Supercomputing 2008 (SC08) conference in Austin, Texas, EVL demonstrated the full capabilities of Visualcasting. During this demo, EVL, the University of Michigan, and Masaryk University (Czech Republic) were linked to Visualcasting servers in Chicago so that they were all able to share a 4K (4096x2048) pixel visualization stream as well as communicate over HD video conferencing (see Figure 23).

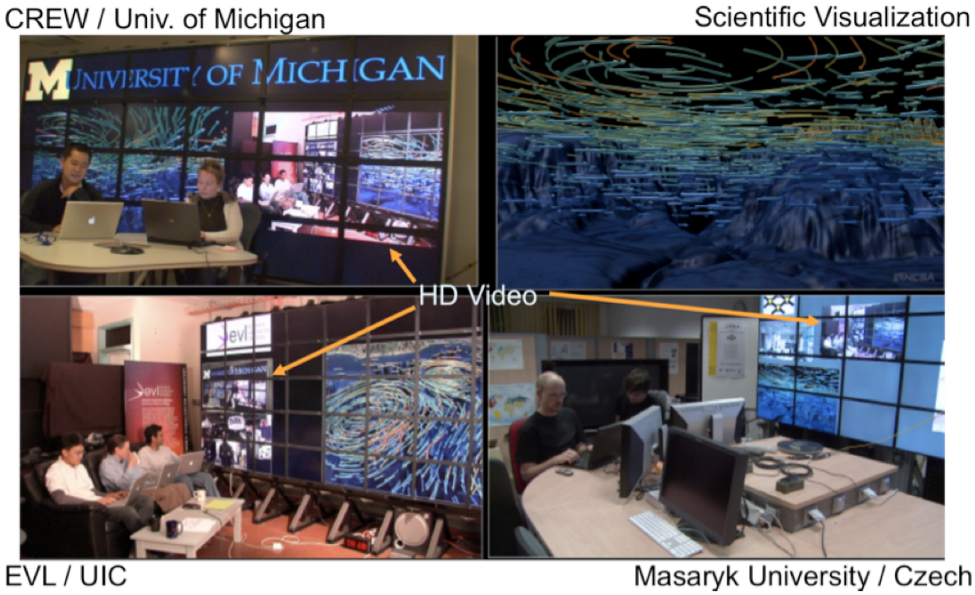


Figure 23. SC08 Visualcasting demonstration

CHAPTER 6 CONCLUSION AND FUTURE WORK

Domain scientists need a collaborative visualization environment that enables them to see and interact with huge amounts of data together with remote collaborators. SAGE supports both wide-area distributed visualization and point-to-point collaboration, and Visualcasting extends SAGE to support global collaboration with multiple endpoints in scalable display environments by distributing high-definition video and audio as well as scientific visualization.

An analytic model of Visualcasting was built and verified by experiments using a Visualcasting implementation. The experimental results showed that the implementation could scale the Visualcasting service with an increasing number of endpoints, and various system parameters affect the Visualcasting performance as predicted by the analytic model.

Frame dropping for a small-scale endpoint was implemented as an approach to address heterogeneity in data bandwidth across Visualcasting endpoints. However, it turned out that the approach required a more elaborate UDP network flow control mechanism or a new reliable network streaming protocol over high-speed wide area network. Another candidate approach to address heterogeneity of Visualcasting endpoints is multi-layered Visualcasting. The idea is to visualcast a different image quality or resolution of image data through each layer. Then each endpoint subscribes to a layer appropriate to its data bandwidth and display resolution. These problems and approaches will be investigated as the future works of this research.

CITED LITERATURE

[Banerjee02] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. “Scalable application layer multicast,” Technical report, UMIACS TR-2002. 2002

[Burger05] Mathijs den Burger, Thilo Kielmann, Henri E. Bal, “Balanced Multicasting: High-throughput Communication for Grid Applications”, SC'05, Seattle, WA, 12-18 Nov. 2005.

[Childers00] Childers, L, Disz, T., Olson, R., Papka, M. E., Stevens, R., and Udeshi, T., “Access Grid: Immersive Group-to-group Collaborative Visualization,” Proceedings of Fourth International Immersive Projection Technology Workshop, 2000.

[Deering91] Deering, S. E., “Multicast Routing in a Datagram Internetwork,” PhD thesis, Stanford University, December 1991.

[DMX04] “Distributed multi-head X project,” http://www.x.org/archive/X11R6.8.2/doc/dmx.html.

[Germans01] Germans, D., Spoelder, H.J.W., Renambot, L., and Bal, H. E., “VIRPI: a High-level Toolkit for Interactive Scientific Visualization in Virtual Reality,” Proceedings of Immersive Projection Technology/Eurographics Virtual Environments Workshop, 2001.

[He03] He, E., et al, “Quanta: a Toolkit for High Performance Data Delivery over Photonic Networks,” Journal of Future Generation Computer Systems, Volume 19, Issue 6, August 2003, pp. 919-933.

[Humphreys00] Humphreys, G., Buck, I., Eldridge, M., and Hanrahan, P., “Distributed Rendering for Scalable Displays,” Proceedings of ACM/IEEE Conference on Supercomputing, 2000.

[Humphreys02] Humphreys, G., et al, “Chromium: a Stream-processing Framework for Interactive Rendering on Clusters,” Proceedings of SIGGRAPH, 2002.

[Jannotti00] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole Jr. Overcast: Reliable multicasting with an overlay network. In Proceedings of the Fourth Symposium on Operating System Design and Implementation (OSDI), October 2000.

[Jeong06] Jeong, B., Renambot, L., Jagodic, R., Singh, R., Aguilera, J., Johnson, A., and Leigh, J., "High-Performance Dynamic Graphics Streaming for Scalable Adaptive Graphics Environment," accepted by ACM/IEEE Supercomputing 2006.

[Klosowski02] Klosowski, J. T., Kirchner, P., Valuyeva, J., Abram, G., Morris, C., Wolfe, R., and Jackman, T., "Deep View: High-resolution Reality," IEEE Computer Graphics and Applications, Volume 22, Issue 3, May/June 2002, pp. 12–15.

[Krumbholz05] Krumbholz, C., Leigh, J., Johnson, A., Renambot, L., and Kooima, R., "Lambda table: High Resolution Tiled Display Table for Interacting with Large Visualizations," Proceedings of Fifth Workshop on Advanced Collaborative Environments, 2005.

[Leigh03] Leigh, J., Renambot, L., DeFanti, T. A., et al, "An Experimental OptIPuter Architecture for Data-Intensive Collaborative Visualization", Third Workshop on Advanced Collaborative Environments, Seattle, WA, June 2003.

[Leigh06] Leigh, J., Renambot, L., Johnson, A., Jeong, B., et al, "The Global Lambda Visualization Facility: An International Ultra-High-Definition Wide-Area Visualization Collaboratory," Journal of Future Generation Computer Systems, Volume 22, Issue 8, October 2006, pp. 964-971.

[McCanne96] McCanne, S., Jacobson, V., and Vetterli, M, "Receiver-driven Layered Multicast," ACM SIGCOMM, 1996.

[Perrine01] Perrine, K. A., Jones, D. R., and Wiley, W. R., "Parallel Graphics and Interactivity with the Scaleable Graphics Engine," Proceedings of ACM/IEEE Conference on Supercomputing, 2001.

[PortAudio07] "PortAudio – portable cross-platform Audio API," http://www.portaudio.com

[RDP08] “Remote Desktop Protocol,” http://msdn2.microsoft.com/en-us/library/aa383015.aspx

[Renambot04] Renambot, L., Rao, A., Singh, R., Jeong, B., et al, “SAGE: the Scalable Adaptive Graphics Environment,” Proceedings of WACE 2004, Nice, France, 09/23/2004 - 09/24/2004.

[Richardson98] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood & Andy Hopper, "Virtual Network Computing", IEEE Internet Computing, Vol.2 No.1, Jan/Feb 1998 pp33-38.

[SDL06] “Simple Directmedia Layer,” http://www.libsdl.org

[Singh04] Singh, R., Jeong, B., Renambot, L., Johnson, A., and Leigh, J., “TeraVision: a Distributed, Scalable, High resolution Graphics Streaming System,” Proceedings of IEEE Cluster, 2004.

[Smarr03] Smarr, L., Chien, A. A., DeFanti, T., Leigh, J., and Papadopoulos, P. M., “The OptIPuter” Communications of the ACM, Volume 46, Issue 11, November 2003, pp. 58-67.

[Solomita94] Solomita, E., Kempf, J., and Duchamp, D. 1994. XMOVE: a pseudoserver for X window movement. *X Resource* issue 11 (Jul. 1994), p143-170.

[Stix01] Stix, G., “The Triumph of the Light,” Scientific American, January 2001.

[Vishwanath06] Vishwanath, V., Leigh, J., He, E., Brown, M. D., Long, L., Renambot, L., Verlo, A., Wang, X., DeFanti, T. A., “Wide-Area Experiments with LambdaStream over Dedicated High-bandwidth Networks,” IEEE INFOCOM, April 2006.

[Woo99] Woo, M., Neider, J., Davis, T., and Shreiner, D., “OpenGL Programming Guide,” Reading, MA, Addison Wesley Longman, 1999.

[Xiong05] Xiong, C., Leigh, J., He, E., Vishwanath, V., Murata, T., Renambot, L., and DeFanti, T., “LambdaStream – a Data Transport Protocol for Streaming Network-intensive Applications over Photonic Networks,” Proceedings of The Third International Workshop on Protocols for Fast Long-Distance Networks, Lyon, France, Feb. 2005.

VITA

NAME Byungil Jeong

EDUCATION

2003 – 2009 Ph.D., Computer Science, University of Illinois at Chicago, Chicago, Illinois

1997 – 1999 M.S., Electrical Engineering, Seoul National University, Seoul, Korea

1993 – 1997 B.S., Electrical Engineering, Seoul National University, Seoul, Korea

PUBLICATIONS

JOURNAL PAPERS

[1] Renambot, L., Jeong, B., Hur, H., Johnson, A., Leigh, J., " Enabling High Resolution Collaborative Visualization in Display Rich Virtual Organizations," Future Generation Computer Systems, Volume 25, Feb. 2009.

[2] DeFanti, T., Leigh, J., Renambot, L., Jeong, B., et al., "The OptIPortal, a scalable visualization, storage, and computing interface device for the OptiPuter ," Future Generation Computer Systems 25, Feb. 2009.

[3] Leigh, J., Renambot, L., Johnson, A., Jeong, B., et al., "The Global Lambda Visualization Facility: An International Ultra-High-Definition Wide-Area Visualization Collaboratory," Future Generation Computer Systems, Volume 22, Oct. 2006.

[4] Singh, R., Schwarz, N., Taesombut, N., Lee, D., Jeong, B., et al, "Real-time Multi-scale Brain Data Acquisition, Assembly, and Analysis using an End to End OptIPuter," Future Generation Computer Systems, Volume 22, Oct. 2006.

[5] Hirano, A., Renambot, L., Jeong, B., Leigh, J., Verlo, A., et al, "The First Functional Demonstration of Optical Virtual Concatenation as a Technique for Achieving Terabit Networking," Future Generation Computer Systems, Volume 22, Oct. 2006.

CONFERENCE PAPERS

[1] Tsukishima, Y., Hirano, A., Nagatsu, N., Imajuku, W., Jinno, M., Hibino, Y., Takigawa, Y., Hagimoto, K., Wang, X., Renambot, L., Jeong, B., Leigh, J., DeFanti, T., Verlo, A., "Lambda Sharing Demonstration via Traffic-Driven Lambda-on-Demand," Proceedings of the 33rd European Conference and Exhibition on Optical Communication (ECOC 2007), Sep. 2007.

[2] Leigh, J., Johnson, A., Renambot, L., DeFanti, T., Brown, M., Jeong, B., Jagodic, R., Krumbholz, C., Svistula, D., Hur, H., Kooima, R., Peterka, T., Ge, J., Falk, C, “Emerging from the CAVE: Collaboration in Ultra High Resolution Environments,” Proceedings of the First International Symposium on Universal Communication, Jun. 2007.

[3] Venkataraman, S., Benger, W., Long, A., Jeong, B., Renambot, L., “Visualizing Hurricane Katrina: large data management, rendering and display challenges,” Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia 2006, Nov. 2006

[4] Jeong, B., Renambot, L., Jagodic, R., Singh, R., Aguilera, J., Johnson, A., and Leigh, J., “High-Performance Dynamic Graphics Streaming for Scalable Adaptive Graphics Environment,” ACM/IEEE Supercomputing, Nov. 2006.

[5] Wang, X., Vishwanath, V., Jeong, B., Jagodic, R., He, E., Renambot, L., Johnson, A., and Leigh J., “LambdaBridge: A Scalable Architecture for Future Generation Terabit Applications,” Broadnets 2006 - Third International Conference on Broadband Communications, Networks, and Systems, Oct. 2006.

[6] Renambot, L., Jeong, B., Jagodic, R., Johnson, A., Aguilera, J., and Leigh, J., “Collaborative Visualization using High-Resolution Tiled Displays,” CHI 06 Workshop on Information Visualization and Interaction Techniques for Collaboration Across Multiple Displays, Apr. 2006.

[7] Tsukinama, Y., Hirano, A., Nagatsu, N., Ohara, T., Imajuku, W., Jinno, M., Takigawa, Y., Hagimoto, K., Renambot, L., Jeong, B., Leigh, J., DeFanti, T., Verlo, A., Winkler, L., “The First Application-Driven Lambda-on-Demand Field Trial over a US Nationwide Network,” Proceedings of OFC/NFOEC 2006 (Optical Fiber Communication/ National Fiber Optic Enginners Conference), Mar. 2006.

[8] Jeong, B., Jagodic, R., Renambot, L., Singh, R., Johnson, A., and Leigh, J., “Scalable Graphics Architecture for High-Resolution Displays,” IEEE Information Visualization Workshop on Using Large, High-Resolution Displays for Information Visualization, Oct. 2005.

[9] Renambot, L., Rao, A., Singh, R., Jeong, B., et al., “SAGE: the Scalable Adaptive Graphics Environment,” WACE 2004, Sep. 2004.

[10] Singh, R., Jeong, B., Renambot, L., Johnson, A., and Leigh, J., “TeraVision: a Distributed, Scalable, High resolution Graphics Streaming System,” IEEE Cluster, Sep. 2004.

[11] Jeong, B., Yoo, S., Lee, S., and Choi, K, “Hardware-Software Co-synthesis for Run-time Incrementally Reconfigurable FPGAs", in Proceedings of the Asia South Pacific Design Automation Conference (ASPDAC), pp.169-174, Jan. 2000.

[12] Jeong, B., Yoo, S., and Choi, K, “Exploiting early partial reconfiguration of run-time reconfigurable FPGAs in embedded systems design”, in Proceedings of the 1999 ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays (FPGA ’99), pp.247-250, Feb. 1999.