Multi-User Centered Resource Scheduling for Large Scale Display Wall Environments

BY

SUNGWON NAM
B.E., Hong-Ik University, Republic of Korea, 2002
M.S., University of Southern California, Los Angeles, 2006

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, 2013

Chicago, Illinois

ACKNOWLEDGEMENTS

During my graduate study at the Electronic Visualization Laboratory, I think I learned more about myself. I had plenty of occasions to diverge and ponder on what really matters to me. In addition to the doctorate degree, academic knowledge, research accomplishments, and professional experiences, I also learned many other skills that would greatly help me in my life as an engineer and a scientist. These would never have been possible if I did not have chances to freely choose what I was going to pursue, explore, try, and fail. My advisor Jason once said that it is the students who have the most power in the laboratory. I think I now understand what it means. I deeply thank Jason for patiently waiting for me and continuously giving me opportunities.

I thank Andy and Jason for their contribution to this work and all the other work that led me here. I have not seen them too tired to discuss ideas and provide advice to me and all the other students in the laboratory. Whenever I had a discussion with them, I could listen to critical questions that I had never thought of, dazzling observations from a different perspective, and invaluable guidance. I also thank Luc for his willingness to share his profound understanding and experience in broad range of disciplines in the Computer Science domain. I learned many things just by watching over his shoulder. Lance and Alan made complex technologies possible and ensured that programs ran all the time with their know-how and knowledge. I kept requesting them resources and diverse configurations of systems and never had an occasion where I did not get what I asked. Maxine was always available and instantaneously responded to me when I struggle with English writing. During all those years, I never worried about various forms and procedures I had to go through as an international student. I would have had many frustrating moments without Dana’s exceptional ability and infinite energy to make everything smooth.

ACKNOWLEDGEMENTS (CONTINUED)

I was privileged to be a part of EVL family with its talented faculties, staff, and students. I am so happy that I have worked with many brilliant students in the laboratory and am grateful to my wonderful friends at EVL. I admire Ratko's positive way of viewing the world. He is the brightest person I've ever met. Ratko and I shared lots of thoughts and honest opinions together. Many times, I could lift off pressures just by having a short conversation with him. He encouraged me with heart whenever I felt frustrated and led me to uncover positive perspectives. James and Khairi shared their academic opinions numerous times. They never hesitated to help when I asked for their time to discuss my work even when they were occupied with their own. I would not have been able to finish my papers and experiments without their support. Byungil and Venkat helped me many times too. From them I learned valuable skills to conduct scientific research early in my doctoral studies.

I extend my sincere appreciation to all of my family members who never stop trusting me and help me to move forward with confidence. My parents made my study in the United States possible and comfortable. I am clueless about how to repay them for their unselfish and endless support. I thank my sisters who have such warm hearts. They are always willing to help me and encourage me to succeed. Undoubtedly, my wife Seunghee and my daughter Christine have been my biggest supporters. They had to endure much when I became such a fastidious man during the tough times. I certainly wouldn't have been able to do this work alone. We have been through this together.

ACKNOWLEDGEMENTS (CONTINUED)

This dissertation is based on work supported in part by the National Science Foundation (NSF), awards OCI-0225642 (OptIPuter) and OCI-0943559 (SAGE), Sharp Laboratories of America, and the King Abdullah University of Science and Technology (KAUST) award number US 2008-107/SA-C0064. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the funding agencies and companies.

SN

TABLE OF CONTENTS

TABLE OF CONTENTS (CONTINUED)

LIST OF FIGURES

LIST OF FIGURES (CONTINUED)

LIST OF FIGURES (CONTINUED)

LIST OF FIGURES (CONTINUED)

LIST OF ABBREVIATIONS

API	Application Program Interface
EDF	Earliest Deadline First
GUI	Graphical User Interface
HD	High-definition
HPC	High Performance Computing
LCD	Liquid Crystal Display
NSF	National Science Foundation
PDF	Portable Document Format
QoS	Quality of Service
SAGE	Scalable Adaptive Graphics Environment
SAGE-Next	Next generation of SAGE
UI	User Interface
VNC	Virtual Network Computing

SUMMARY

Data-intensive e-Science applications are run by global cyberinfrastructure where visualization platforms, scientific instruments, computing and storage resources are distributed over several locations and connected by high-speed networks. The enormous amount of data produced by these applications poses major challenges to researchers who need to look at, analyze, and make sense these massive troves of scientific data. To ensure high productivity, researchers need cost-effective tools that can be integrated in their work environments.

Visualization has proven extremely effective throughout the various stages of the scientific inquiry process when one needs to analyze and interact with large data volumes. With appropriate visualization, one can verify the correctness of a complex simulation model, get insight into the model, and communicate results to other in an intuitive way. With large-scale, high-resolution tiled-display walls, the efficacy of visualizations can be greatly amplified because of the unique way the display walls can present visualizations to users. Modern large-scale display wall technology such as the Scalable Adaptive Graphics Environment (SAGE) and its successor (SAGE-Next) are designed to bring high-resolution visualizations to users over high-speed networks. Moreover, these display wall technologies are unique in that they enable highly collaborative visualization environments by supporting multi-user inputs on multiple visualizations simultaneously.

Traditionally, tiled-display walls have been built from a cluster of computers due to the limited graphics capability of a single display node. With the emergence of today's multi-head technologies however, the graphical capabilities of a single computer node has been greatly amplified. This empowers a single computer to drive a large-scale collaborative display wall, in many cases eliminating the need for a computer cluster, which significantly reduces the cost of ownership and maintenance of these environments. However, large-scale collaborative display wall environments create user interaction patterns that are very different from traditional single-user desktop environments. This poses a challenge in resource management when a collaborative display wall is driven by a single-machine because traditional Operating System’s resource management focuses on system-level fairness and job completion throughput without knowing what users might be interested in.

This dissertation presents a resource scheduling framework for multi-user collaborative thin-client display wall environments where multiple users can interact simultaneously with multiple visualizations that are streamed from distributed computing and storage resources over high-speed network. A model-based application priority assessment that reflects the degree of users’ interests on contents on the wall is presented. The model takes applications’ visual states, user interactions, and the wall’s usage pattern as inputs and quantifies relative importance of applications. Resource estimation and allocation scheme determines perceptually fair resource distribution based on the importance of applications. A user study is conducted to evaluate the framework’s fair resource distribution based on multi-user awareness.

1. INTRODUCTION

The NSF-funded OptIPuter project (Optical networking, Internet Protocol, computer storage, processing and visualization technologies) where distributed computational resources are tightly coupled over high-speed optical network is envisioned by a team of researchers at Electronic Visualization Laboratory (EVL) and University of California, San Diego 10 years ago [Smarr, '03]. The goal of the OptIPuter project is to deliver an architecture where terabytes and petabytes of distributed data generated by e-Science applications can be easily accessed over the optical networks called LambdaGrid in order for scientists to visualize, interact, and analyze their data. The tiled-display wall system called LambdaVision and graphics middleware called Scalable Adaptive Graphics Environment (SAGE) [Jeong, '10] that drives the LambdaVision that is located at users' end are the outcomes of the OptIPuter project. Figure 1 shows the LambdaVision driven by SAGE. Essentially, the OptIPuter project's model where users access remotely distributed resources over network is an early instantiation of today's growing Cloud computing model.

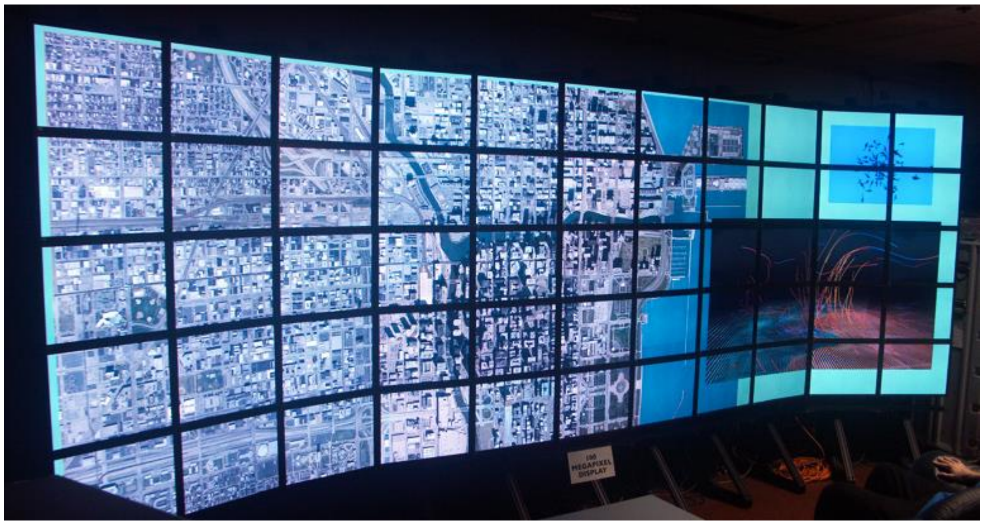


Figure 1. The 100-megapixel LambdaVision tiled-display wall driven by a cluster of 28 computers and run by Scalable Adaptive Graphics Environment (SAGE).

Visualization has an important role in a scientific workflow where a simulation result needs to be presented in a manner that it can be analyzed easily. It can make the process of verifying a scientific model simpler by allowing more insight into the model. Many researchers are now adopting large-scale tiled-display wall environments because it is the only way to view the huge amount of complex data. It is proven that the ability to juxtapose multiple visualizations simultaneously on the display walls that have expansive size and exquisite resolution greatly helps researchers for their scientific analysis and discovery process [Andrews, '10; Ball, '05; Czerwinski, '06; Haller, '10; Leigh, '06; Plaue, '09; Tan, '03; Yost, '07].

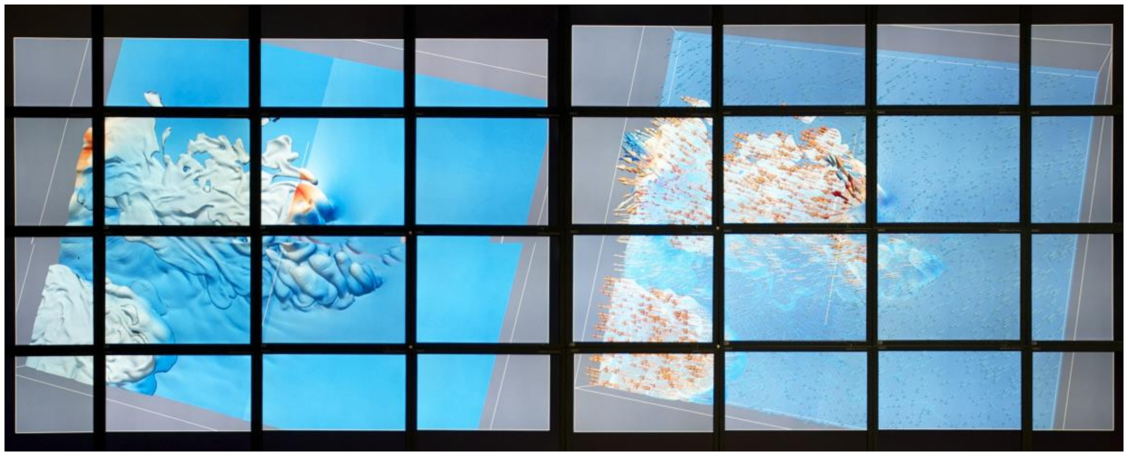


Figure 2. LambdaVision displays two ParaView sessions that render NCSA's storm data remotely on a high-performance render server. The rendered images are streamed in parallel to LambdaVision display wall driven by SAGE.

A common way to build large-scale display walls is to tile multiple individual displays and connect them to a cluster of computers. Cluster middleware is needed to enable users to work with the wall as a single contiguous display surface. Traditional tiled display middleware such as CGLX [Doerr, '11], Chromium [Humphreys, '02] and Equalizer [Eilemann, '09], designed as large-scale visualization platforms, can be regarded as distributed graphic frameworks. They emphasis on parallel rendering of large-scale datasets using a computer cluster, and are typically aimed at cases where a single user interacts with a single application spanning the entire display wall. On the other hand, modern tiled display middleware such as the Scalable Adaptive Graphics Environment (SAGE) [Jeong, '06] and its successor SAGE-Next let users launch distributed visualization applications on remote clusters whose outputs are then streamed directly to display walls. This makes SAGE and SAGE-Next low-cost, “thin-client” visualization endpoints where such visualizations are rendered by remote computing resources and streamed over an optical network to a display wall. An example where SAGE is coupled with a well-known scientific visualization tool, ParaView [Cedilnik, '06] is shown in Figure 2. Details of ParaView and SAGE integration is presented in [Nam, '09]. SAGE and SAGE-Next also provide highly collaborative visualization environments by enabling multiple users to simultaneously view and interact with these streamed visualizations on large-scale display walls [Jagodic, '11]. An overview as well as real-world use cases of the thin-client display wall paradigm is discussed more in [DeFanti, '09; Smarr, '09].

Middleware for large-scale tiled-display walls are typically designed to run on a computer cluster because, historically, a computer cluster has been necessary to drive these large-scale display walls. However, the emergence of multi-headed graphic technologies (such as NVIDIA’s Scalable Visualization Solutions and AMD’s Eyefinity), has greatly amplified the graphical capabilities of a single computer node. Also displays’ physical dimension and image resolution are rapidly increasing. These empower a single computer to drive a large-scale display wall, in many cases eliminating the need for a computer cluster, which significantly reduces the cost of ownership and maintenance of these environments. Furthermore, applications can now run natively on a single-machine without the need to parallelize them, thus simplifying application development for large-scale display walls. Figure 3 shows a 20x6 foot large-scale collaborative tiled-display wall driven by a single computer machine at the Electronic Visualization Laboratory at the University of Illinois at Chicago.

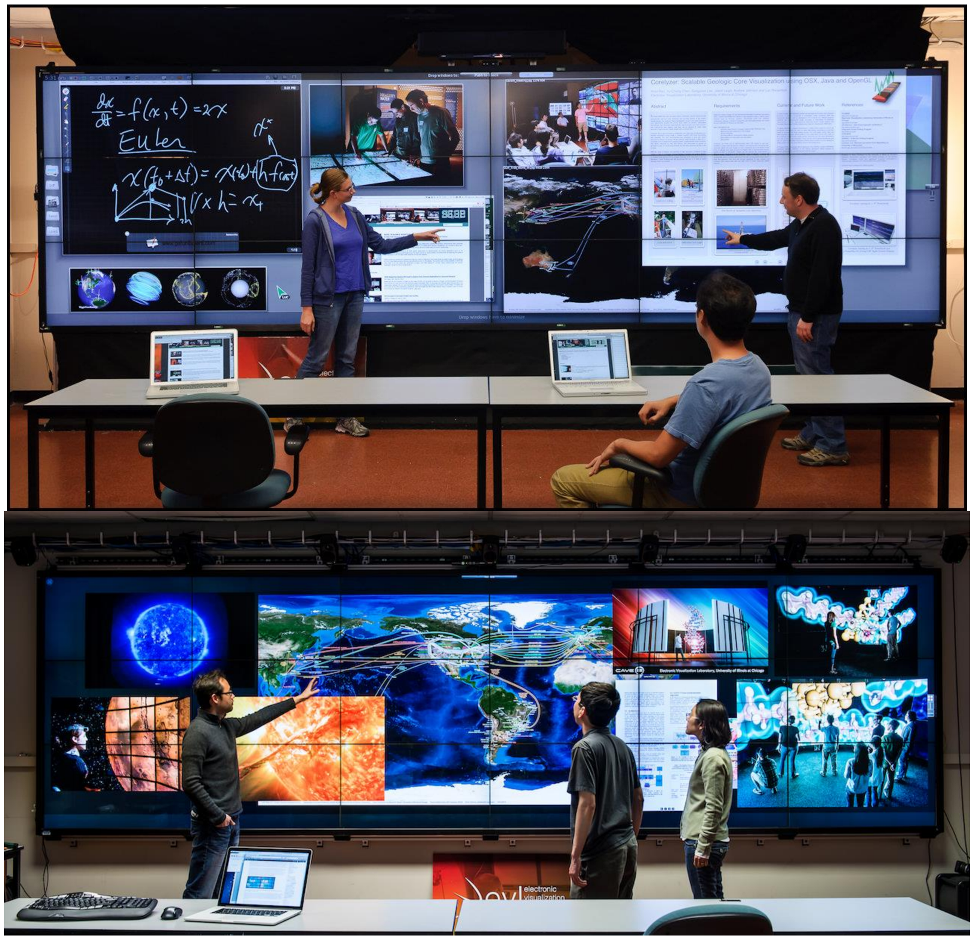


Figure 3. A single machine-driven tiled-display wall run by SAGE (top) and SAGE-Next (bottom) at the Electronic Visualization Laboratory in the University of Illinois at Chicago. The 20′ by 6′ display wall is made up of 18 LCD panels with a total resolution approximately 18.8 megapixels.

Driving a large-scale collaborative display wall with a single computer however, presents significant challenges in resource management. A display wall middleware relying on general-purpose operating system resource scheduling may fail to provide a good user experience in large-scale collaborative display wall environments where multiple users interact simultaneously with multiple applications. A general-purpose operating system schedules resources based on system-wide performance measures such as job completion throughput and fine-grained fairness. In large-scale collaborative display wall environments, multiple users may simultaneously view and interact with Cloud media data such as pictures, documents, and movies, VNC-shared desktop screens, and interactive scientific visualization. Users can also move, resize, and arrange windows on the display wall in a variety of layouts. The number of applications running on the system, their layouts on the display, and the user-interaction pattern in these systems can differ drastically from traditional desktop environments where a single user typically interacts with a limited number of applications. This difference makes traditional resource scheduling schemes unfit for collaborative display wall environments.

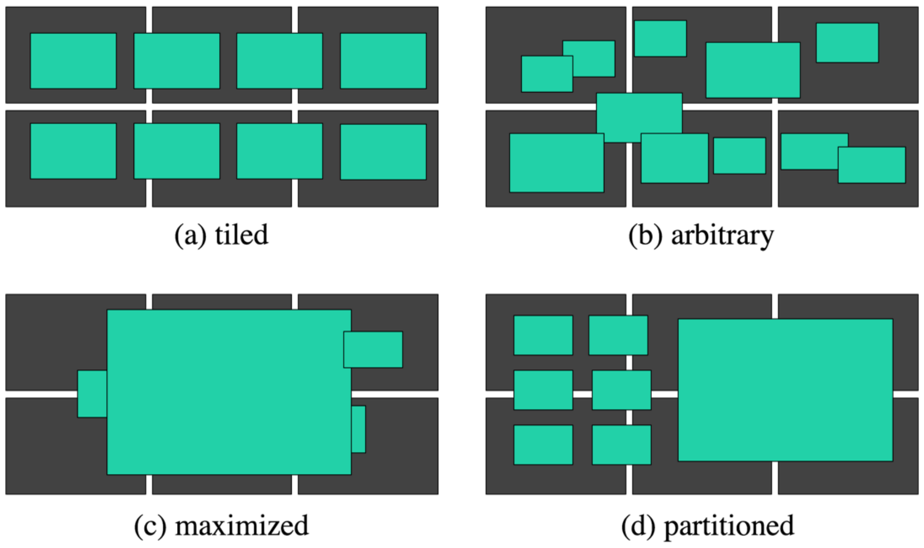


Figure 4. Examples of application layout on a large-scale display wall. From (a) to (c), application layout can be classified by amount of overlapped windows; (a) as least overlapping and (c) as most overlapping. In (d) the wall is partitioned and each partition can employs a different layout.

Figure 4 shows examples of layouts on a large-scale collaborative display wall with varying degrees of window overlap. A traditional operating system will try to ensure fair sharing of system resources in all cases depicted in Figure 4, while fair sharing might only be useful in the case depicted in (a). When the window layout is arbitrary as in (b), giving more system resources to windows with which users are interacting can achieve a better user experience than a fair sharing. Similarly, a better user experience can be achieved in (c) if more system resources are allocated to the application whose window is in the foreground. For the case shown in (d), a fair-sharing scheme is appropriate for applications in the left section of the display, while the right section requires a scheduling scheme similar to (c). This dissertation identifies two issues when a general-purpose operating system scheduler is employed in large-scale collaborative display wall environments.

- Process priority is typically based on process behavior rather than user behavior. Thus the priority in traditional scheduling schemes does not reflect the degree of user interest.
- Resource requirements for applications vary and fine-grained, thread-level fairness in a general-purpose scheduler does not typically consider resource requirement of an application. Thus it can fail to provide fairness in terms of the quality experienced by users.

Given the variety of layouts in large-scale collaborative display wall environments, the scheduling policy should not necessarily be based on system-wide job completion throughput or fine-grained, thread-level fairness. Rather, the scheduling policies should be based on window-layout and user interaction criteria. For example, an appropriate scheduling policy might allocate more resources to applications that occupy the largest space on the wall, to applications that are least occluded, or to applications with which users are interacting. Implementing these policies can increase the perceived performance of the system, therefore providing a better user experience in a multi-user collaborative setting. The goal of our scheduling framework is to fairly distribute system resources to applications to optimize their performances as experienced by users. This dissertation calls this *presentation fairness*.

In this dissertation, a multi-user centered resource scheduling framework targeting a thin-client display wall where visualizations are streamed to the display wall from remote high-performance computers over high-speed networks is presented. To evaluate the effectiveness of the presented scheduling framework, the author conducted a user study where multiple users in groups of three subjects interacted simultaneously with applications on the display wall with and without the scheduling scheme presented in this dissertation.

1.1 Summary of Contributions

This dissertation presents a novel resource scheduling framework and implementation that achieves better responsiveness and interactivity for multi-user collaborative thin-client display wall systems by optimizing system's resource allocations based on the unique usage patterns of the display wall. Although the framework is implemented for a display wall system driven by a single machine, the fundamental resource scheduling concept is applicable to a cloud-based visualization system that allows multi-user interaction on multiple applications simultaneously.

An application priority model for multi-user collaborative display wall environments is presented. The priority model describes how user interactions with display walls can be interpreted to determine the applications that users are interested in at a given time. The model takes applications' visual and interaction factors and the display wall's usage pattern as inputs, and ascribes priorities to indicate the degree of relative importance of applications. This information is used to distribute resources among applications to achieve a better user experience.

A scheme to estimate applications' resource need is presented. Unlike traditional non-clairvoyant schedulers, the scheduling framework in this dissertation takes applications' resource needs into account for the resource allocation in order to achieve user-perceived fairness, rather than system-level fairness. The framework estimates an application's resource need based on user interactions to find the optimal resource allocation for the application.

Finally, the scheduling framework is evaluated in a user study that was designed to simulate typical multi-user interaction patterns in thin-client display wall environments. An implementation of the graphics middleware called SAGE-Next for a collaborative display wall driven by a single machine was designed and developed to support the user study. The study demonstrates the efficacy of the scheduling framework when multiple users are interacting with applications simultaneously on the display wall.

1.2 Document Structure

Chapter 2 gives background of the presented framework in an aspect of graphics middleware that runs multi-user display walls in collaborative work environments. The motivation of the thin-client display wall systems and software architecture of the graphics middleware that implements the scheduling framework are introduced. Chapter 3 describes previous literature in job scheduling in traditional operation systems followed by scheduling schemes focusing on human-centered scheduling and real-time applications. The overview of the scheduling framework is presented in Chapter 4. The details of the scheduling framework follow. In Chapter 5, the priority model for multi-user display walls is presented. In Chapter 6, the resource distribution scheme is described in detail. The evaluation of the scheduling framework is given in Chapter 7. Finally, the summary of the framework and future research direction is presented in Chapter 8 and 9.

2. BACKGROUND

An overview of the collaborative thin-client tiled-display wall and two graphics middleware that run the display wall are introduced in this chapter. A brief history of the display wall LambdaVision and its software compartment SAGE are presented in Chapter 2.1. The next generation of SAGE, called SAGE-Next that was motivated by modern multi-headed graphics hardware trend, and this dissertation focuses on, is discussed in Chapter 2.2.

2.1 <u>Graphics Middleware for Display Walls</u>

The National Science Foundation's OptIPuter project [Smarr, '09] inspired by the rapid growth in wide-area bandwidth with development in optical networking. OptIPuter project's goal is to make large-scale data easily accessible to researchers regardless of their geographic locations by connecting them with optical networks. The ultra-high resolution tiled-display wall driven by commodity PC clusters that are connected to optical networks is one of the outcomes of the OptIPuter project. It is to provide a greater display area that can display multiple high-resolution visualizations that are rendered remotely by high-performance computing resources and streamed to users through the network. Furthermore it enables communications via high-definition video with colleagues while maintaining interactivity. The graphics middleware called SAGE is developed to enable these objectives. Figure 5 shows an array of 55 LCD panels driven by a cluster of 28 computers providing 100-megapixel seamless display surface (termed LambdaVision) run by SAGE.

SAGE can be viewed as a cluster operating system for thin-client display walls. It presents a seamless display surface from clustered display nodes, manages to launch visualization applications remotely, handling multiple parallel streams simultaneously, and provides multi-user interaction interface. A parallel streaming model where an image is partitioned and streamed in parallel from a remote application is illustrated in Figure 6. An application embeds the SAGE's Application Program Interface (SAGE API) in it. SAGE API is responsible for partitioning the image passed internally from the application and streaming of the image fragments to corresponding display nodes in parallel.

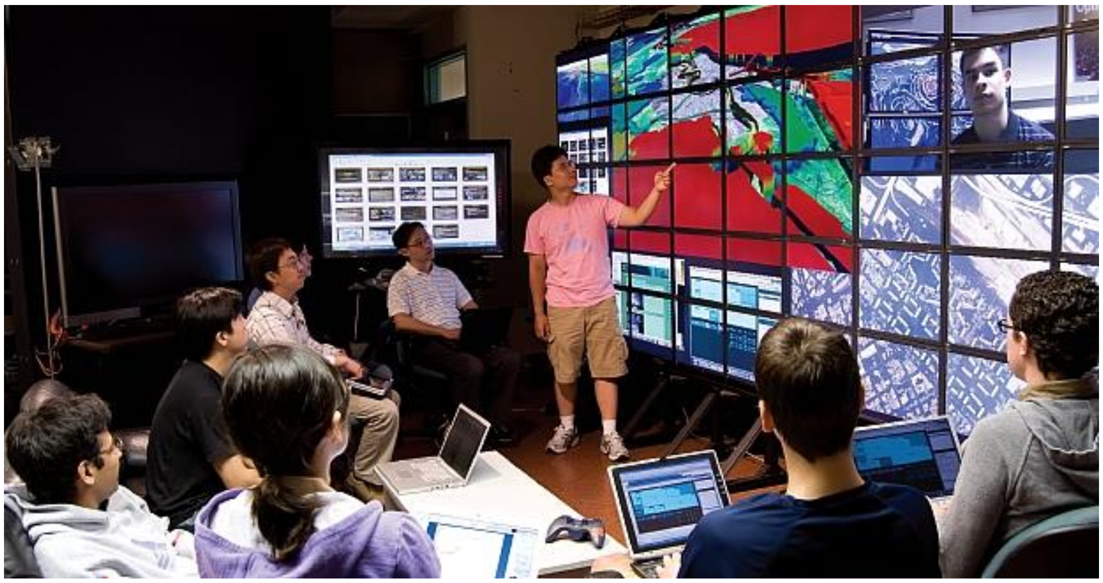


Figure 5. 100-megapixel LambdaVision display used in a research meeting at the Electronic Visualization Laboratory at the University of Illinois at Chicago. The participants stream their desktop screens to the display wall using VNC. Multiple visualizations and HD video of remote participant are streamed to the display wall.

One of the challenges in typical parallel computing is data synchronization. In a clustered display wall system, proper synchronization between display nodes is crucial to present seamless imagery to users. The problem arises when there are multiple visualizations, each can run at different frame rate, simultaneously run on a display wall. A single display node might have to receive and display multiple different image fragments when multiple application windows are overlapped on the display node. Figure 7 depicts this problem. Two visualizations each run at different frame rate are streamed and displayed on the 3 by 3 tiled-display wall in Figure 7. The display node at the center [2,2] receives and displays two different streams simultaneously as shown in the right side of Figure 7. After the initial state at time t0, the next image fragments of each visualization (i+1 and j+1) are received at the display [2,2] while display [1,2] is still waiting for the next image fragment (i+1) of the visualization 1. The graphics swap buffer is needs at display [2,2] at time t1 regardless of the visualization 1's state due to visualization 2's higher frame rate. As a result, the image fragments of the visualization 1 at the display node [1,2] and [2,2] shows inconsistent image frames causing discrepancy in the visualization. Intuitively, one can solve this problem by enforcing each visualization synchronizes its image fragments before uploading them to the graphics memory at each display node. However, this approach is not scalable as the number of display nodes and the number of visualizations increases. A scalable, two-phase inter-node synchronization algorithm where a single synchronization master node ensures each visualization's image synchronization and each display node's graphics swap buffer synchronization is presented in [Nam, '10].

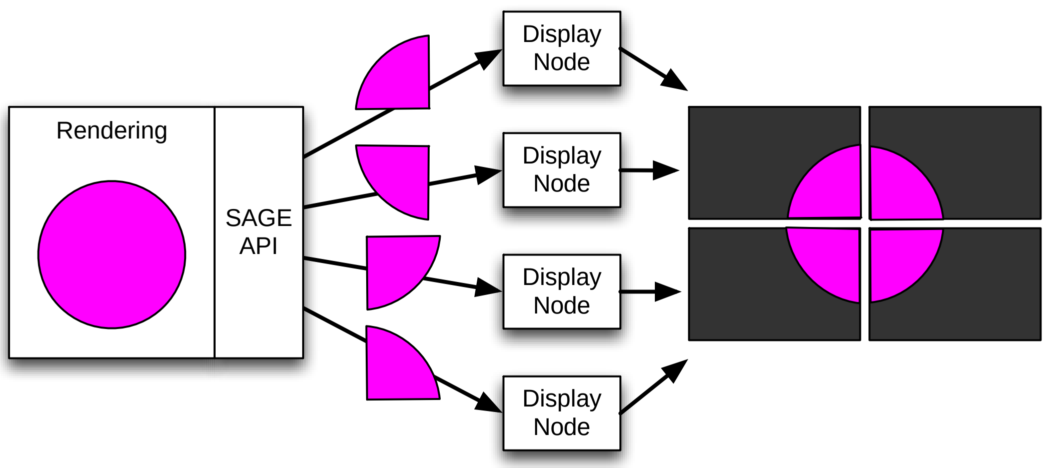


Figure 6. The parallel image streaming model of SAGE. The image rendered by a remote rendering machine is first partitioned into multiple fragments by the SAGE API and streamed in parallel to SAGE's display nodes.

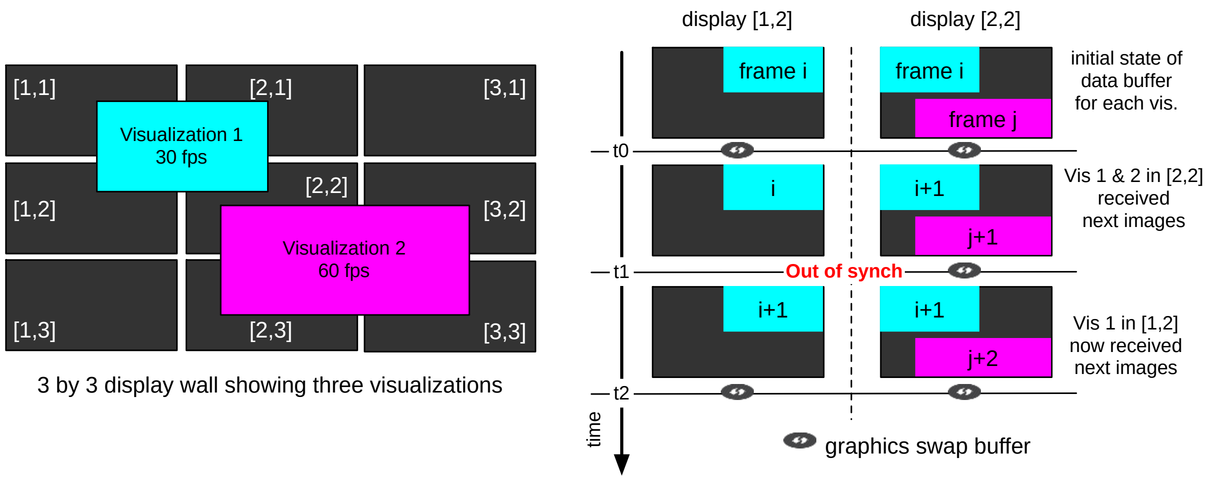


Figure 7. An example where two visualizations each run at different framerate are streamed and displayed on the 3 by 3 display wall. The display node at the center [2,2] receives and displays two different streams simultaneously. After the initial state at time t0, the next image fragments of each visualization (i+1 and j+1) is received at the display [2,2] while display [1,2] is still waiting for the next image fragment (i+1) of the visualization 1. The graphics swap buffer is needs at display [2,2] at time t1 regardless of the visualization 1's state due to visualization 2's higher framerate.

As with the emergence of graphics hardware that can support larger image size and multiple display heads, new software termed SAGE-Next is designed to utilize this multi-headed graphics hardware trend. The motivation that drove the single-machine display wall solution is to make a large-scale display wall system easier to build and manage for users who are not expert in managing a computer cluster hardware and software. Thereby wider range of users can benefit from the highly collaborative work environments.

One of the key differences between SAGE-Next and SAGE is that SAGE-Next simplifies the parallel streaming architecture of SAGE by running a display wall with a single machine. The images rendered remotely do not have to be fragmented and streamed in parallel. The overhead incurred at the display wall side due to the need for synchronization can be lifted. The simplified architecture also allows application developers to build native applications without needing them to parallelize their applications.

However, the simplified architecture poses significant challenges in resource management. Compared to a cluster of computers, a single machine's resource is much limited. High resolution images can not be streamed in parallel in order to distribute network overheads over multiple display nodes as in a cluster-driven display walls. In addition to the SAGE's core abilities (image streaming and multi-user interaction), the unique attribute of the SAGE-Next lies in its resource scheduling framework that is to increase user-perceived performance of a single machine whose resources are limited compared to that of a cluster. Thus, the multi-user centered resource scheduling framework presented in this dissertation is based on the single-machine driven collaborative display wall environments where multiple users can launch and simultaneously interact on multiple visualizations that are streamed from remote compute and storage resources. The scheduling framework presented in this dissertation is implemented as a part of SAGE-Next.

2.2 SAGE-Next Architecture

The core components of SAGE-Next are comprised of the Scene, a Scene Management Layer, a UI server, an Application Launcher, an Interaction Manager, The Performance Monitor, and the scheduler. Figure 8 illustrates these components and their relationships.

Multiple message streams from users are serialized at the UI server. The serialized messages are then pre-processed to be handled by other components. If the message is an interaction type such as interaction with a mouse device then the message is forwarded to the Interaction Manager. If the message is intended for launching an application instance then the message is sent to the Application Launcher.

A user launches an application by sending a request to the Application Launcher through the UI server. Either the Application Launcher or the application itself starts the corresponding remote process that generates and streams contents to the display wall. Therefore, in most cases, an application in the Scene consists of a network thread that receives contents from the remote resources and a mechanism to store and display the streaming contents. Also an application defines how to interact its contents and provides GUI. The Application Launcher then adds the instance of the application to the Scene. The Scene contains all the applications whose contents are displayed on the display wall and interacted by users. The Scene Management Layer included in the Scene manages applications' layout on the display wall. It also provides an interface for users to partition the display wall, save and restore sessions. Once an application is added to the Scene, user interactions are delivered to the application through the Interaction Manager that receives a stream of message strings from the UI server. The Interaction Manager is responsible for finding a valid recipient (typically an application or the Scene itself) of the interaction by querying the Scene Management Layer and triggering a callback operation of the recipient of the interaction.

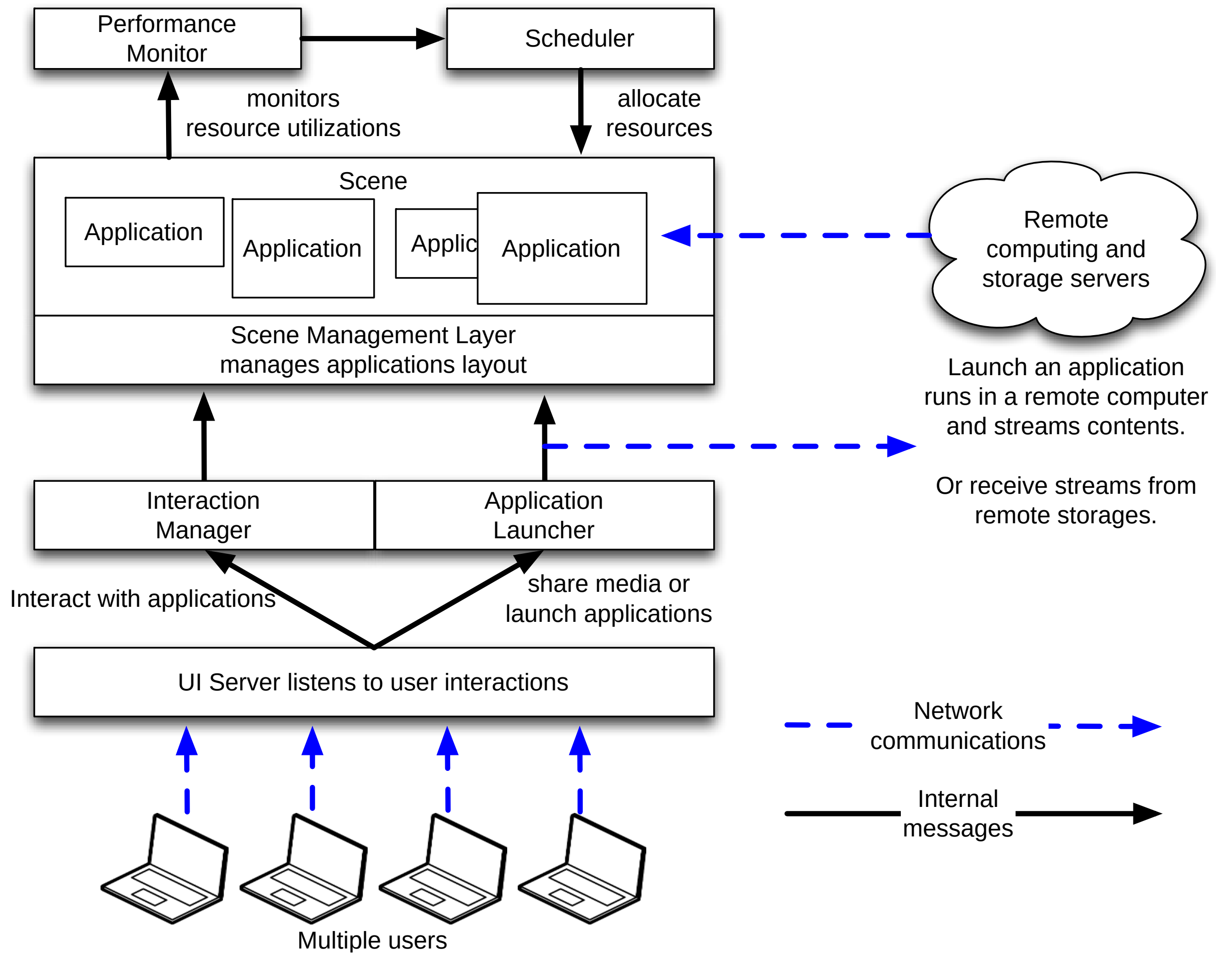

Performance Monitor
Scheduler
monitors resource utilizations
allocate resources
Scene
Application
Application
Applic
Application
Scene Management Layer manages applications layout
Remote computing and storage servers
Launch an application runs in a remote computer and streams contents.
Or receive streams from remote storages.
Interaction Manager
Application Launcher
Interact with applications
share media or launch applications
UI Server listens to user interactions
Network communications
Internal messages
Multiple users

Figure 8. Architectural diagram of SAGE-Next. Control message flow between components are illustrated. User interactions reach the scene and the applications in the scene through UI server. Performance monitor and the scheduler are responsible for scheduling resources.

An application's performance is continuously monitored and analyzed by the Performance Monitor. This information is sent to the Scheduler to make scheduling decisions. Finally, the Scheduler controls resource consumptions of applications by modulating their performances. The details of the Performance Monitor and the Scheduler are discussed in Chapter 4 and 6.

Figure 9 shows the in-depth look of an application runs in SAGE-Next. The basic components of an application are the application-scene interface, GUI components, visual contents, contents receiving thread, and a performance tool. The application-scene interface enables the application to communicate with the scene and allows the Scene to manage the application’s geometry in the Scene. The multi-user-aware GUI components provide base interactivity that is aware of multi-user interactions and communication with the Interaction Manager. Application developers use the GUI components to provide multi-user interactions in their applications. The contents receiving thread receives a network stream of visual contents from remote resources such as Cloud. The visual contents are then presented on the display wall by the Scene. The performance tool in an application measures the application’s performance and reports to the Performance Monitor. It also receives and conforms the performance demand set by the scheduler. This procedure is discussed in detail in Chapter 6.

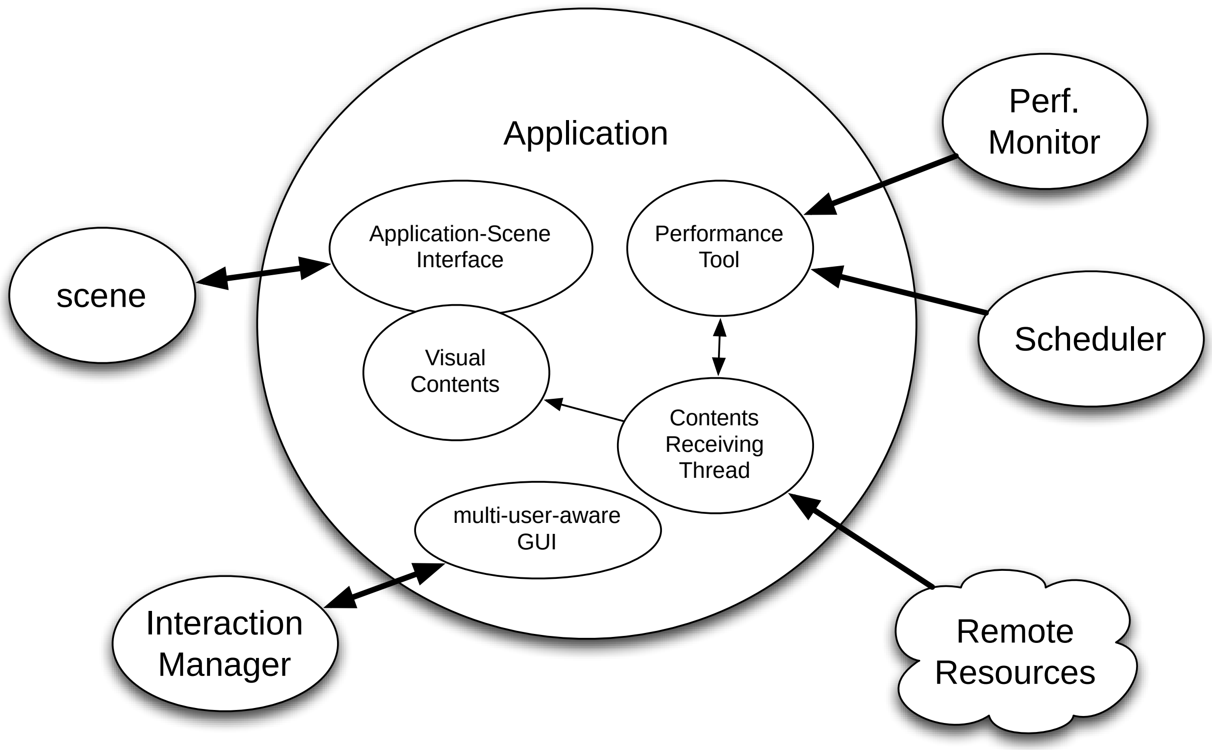


Figure 9. The components of a SAGE-Next application. An application communicates with the Scene through the application-scene interface and displays its visual contents. User interactions are handled by UI components in response to the interaction manager. The performance tool is responsible for monitoring the performance information and conforming the scheduler’s demand.

3. RELATED WORK

The job scheduling is deeply researched area in Computer Science. Typically, main agenda in job scheduling research includes the fairness, no starvation, and resource utilization. This chapter introduces traditional scheduling policy in general-purpose operating systems first in Chapter 3.1 followed by early and recent approaches in human-centered scheduling research in Chapter 3.2. In Chapter 3.3, various scheduling research focusing on real-time applications are introduced.

3.1 <u>Process Scheduler in Operating Systems</u>

The common objectives of modern general-purpose operating system schedulers are high job completion throughput, interactivity, and fair sharing. In particular, the time sharing Linux operating system's scheduling scheme uses a notion of a time slice which sets the maximum time during which a process is allowed to use a processor. Thus, high job completion throughput can be achieved by giving the time slices only to processes that are ready to run. If the process either finishes or has to wait for resources to be available then it can be preempted so that no idle process occupy a processor wasting the computing resource. The preemptive, time-sharing model also enables fair sharing of resources by maintaining a counter that represents a priority of a process. The counter for a process keeps decreasing its value while the process occupies a processor. The scheduler may schedule processes in a manner that can keep the counter values as uniform as possible to be fair. Fast response time for interactive processes can be achieved by using the counter as well. A process waiting for I/O devices increases its counter so that the process can have higher counter value by the time when the process is ready to run. This can make processes with smaller counter values (such as a batch process that consumed lots of processor times while the interactive process was waiting for I/O devices) be preempted by a scheduler. There are books and articles that explain modern Linux scheduler's scheduling policy and designs as in [Bovet, '05; McKusick, '04]. Moreover, there are many research in process scheduling deal with issues with real-time processes where meeting time-constraints of the processes is crucial. This is discussed more in Chapter 3.3. Also the issues with efficient process migration policies in modern multi-core hierarchical memory architecture are well studied too [Chandra, '05; Corbet, '04; Durand, '96; Vishwanath, '08].

Aforementioned modern time sharing, general-purpose operating system schedulers characterized as a non-clairvoyant scheduling where the scheduler does not rely on processes' characteristics [Motwani, '93]. This is because fast turnaround time (a total time taken by a scheduler to finish a job) is important in the time sharing systems. While the non-clairvoyant scheduling scheme for a typical desktop environments achieves its goals as explained, it lacks of ability to discover what users are interested in and schedule jobs in a way that the performance of the system can be perceived to be responsive and fair in multi-user collaborative environments mainly because scheduling decisions are made based on processes' behavior rather than users' interactions with the applications run in the system.

3.2 Human Centered Scheduling

The main goal of human-centered scheduling schemes oriented toward optimizing user perceived interactivity rather than system-wide performance measures. It is based on a hypothesis that performance measures a system wants to achieve might not necessarily be what users' want. The Interactive Scheduling scheme presented in [Evans, '93] identifies interactive processes by monitoring input devices' activities in the X Server. The scheduler then assigns higher priorities to those processes and prioritizes CPU and memory allocation to them in order to increase the responsiveness of interactive processes. Etsion et al. takes a similar approach to improve user interactivity [Etsion, '04]. In addition to monitoring user input events in the X Server, they also use the ratio of pixel-change to window-size to estimate the application's importance to the user. Their experiment results show their approach result very low quality degradation of user interactive jobs even when the system is overloaded. Zheng et al. presents a configurable kernel module that monitors I/O channels to identify interactive processes based on user-access patterns and the usage of those I/O channel [Zheng, '10]. They tested this approach with various combinations of applications and showed wide-range of applications can be benefit from their approach.

While the above work aims to improve user experience with interactive applications, the solutions proposed are limited to traditional desktop environments where a single user interacts with the system using traditional I/O devices such as keyboard and mouse. Therefore, these scheduling schemes cannot be applied directly to large-scale collaborative display wall environments, which introduce user-interaction patterns and application window layouts that are drastically different from desktop computer systems.

3.3 Real-Time Scheduling

Real-time Schedulers are aimed at time-sensitive real-time applications that impose strict completion-time requirements (deadlines) even when the system is overloaded. Real-time schedulers employ an unfair scheduling of resources that is biased towards a specific set of applications to meet their deadlines. Real-time schedulers can be in principle adapted to increase the perceived performance of a system. In the case of multi-user tiled display wall environments, resource allocation can be biased towards applications that are presumed to be receiving most of users’ attention, thus maximizing the perceived performance of the system.

While real-time schedulers have been used in interactive multimedia systems, they have not been tested in large-scale display wall environments. Moreover, large-scale displays offer unique affordances that allow collaborative, multiuser interaction with a large number of applications simultaneously. Therefore, a successful scheduling scheme for these environments should be specifically tailored to address these unique characteristics in order to increase the user-perceived performance of the environment. This section briefly surveys work on various real-time scheduling techniques.

3.3.1 Rate-Monotonic and Earliest Deadline First

Liu and Layland [Liu, '73] showed that their rate-monotonic algorithm meets all periodic tasks’ deadline, bounded on processor utilization from 69% to nearly 100%. In rate-monotonic algorithm, priorities are assigned simply based on the progression rate of periodic tasks; tasks with shorter periods receive high priority. The Earliest Deadline First (EDF) scheduling algorithm in their work assigns highest priority to the task whose deadline is the nearest. EDF achieves higher CPU utilization at a cost of dynamic priority assignment. Such real-time scheduler requires precise prior knowledge in task execution time and cannot be applied when system is overloaded. Moreover this greedy approach is designed to minimize missed deadlines rather than to increase user-perceived performance. However, the principle of rate-monotonic and EDF scheduling is employed in many real-time scheduling schemes.

The scheduling scheme presented in this dissertation is different in that the priorities are assigned based on the degree of users' interests on applications on a large display wall. Also supporting dynamic priority assignment is crucial in order to reflect the temporal changes in users' interests in the priority.

3.3.2 Resource Reservation

Resource Reservation is a restrictive approach to ensure that time-sensitive applications meet their deadlines. To be effective, the admission control is required to provide a guarantee on meeting real-time requirement of running tasks. The admission control rejects an application if the amount of resource that the application requests exceeds the amount of remaining resource in the system. A mechanism to reserve processor capacity in conjunction with the rate-monotonic algorithm is illustrated in [Mercer, '94]. Real-Time Mach adopts the resource reservation technique in its scheduler to support real-time applications [Tokuda, '90]. Less restrictive forms of the reservation scheme where a thread is allowed to negotiate CPU-time based on its rate progression is introduced in [Yau, '97]. Jones et al. present a system with a CPU scheduling algorithm that ensures minimum guaranteed execution rates of real-time processes [Jones, '97]. In resource reservation scheme, estimating the resource requirement of a task prior to the reservation can be challenging. Accurate estimation of resource requirement based on application profiling is well studied in [Urgaonkar, '02].

Conceptually, this dissertation employs similar notion of resource estimation and less restrictive reservation. The scheduling scheme in this dissertation differs in that it does not reject an application nor negotiate resources. The resource allocation (resource reservation in this context) is entirely based on applications' resource requirements and dynamic priorities. Thus, it estimates resource requirements in real-time by monitoring user interactions on the applications

3.3.3 Gang Scheduling

In Gang Scheduling, similar processes are grouped together (ganged) and form a hierarchical structure so that different scheduling policies can be applied to different processes groups. This approach is often combined with the resource reservation scheme. Real-Time Mach groups one or more processors to form a processor set and applies different scheduling policies on processor sets. Golub et al. present an improved scheduling paradigm over Real-Time Mach with emphasis on supporting a combination of time-critical and conventional applications [Golub, '94]. In the CPU allocation framework for multimedia OS proposed by Goyal et al., CPU bandwidth is partitioned hierarchically by different groups of applications each with different resource requirements [Goyal, '01]. Anderson presents an approach to increase cache utilization for real-time applications by grouping processes in multi-core architecture [Anderson, '06]. The Gang Scheduling scheme can be useful for a system that needs to support different types of application instances running simultaneously because it can effectively prevent compute-intensive batch applications from degrading real-time applications performance. The scheduling scheme presented in this dissertation is similar in concept where it prevents applications that are not receiving users interests from degrading the performance of applications that users are interested with. Thus grouping applications based on their importance is important in this dissertation whereas the gang scheduling groups applications that have similar performance characteristics.

3.3.4 Proportional Sharing

The goal of proportional sharing is to distribute system resources to all running tasks proportional to their relative weight. Once the weight of a task is defined, calculating proportional weight of the task is straightforward. Proportional sharing focuses on fair sharing of resources based on weight and is analogous to Weighted Fair Queuing. The EDF scheduling in conjunction with the notion of Virtual Time [Zhang, '90] is introduced in [Stoica, '96]. Stoica et al. also showed proportional sharing combined with resource reservation scheme in [Stoica, '97]. A virtual time algorithm that focuses on meeting real-time requirements while achieving proportional fairness is shown in [Duda, '99]. Nieh and Lam also present a similar scheduling algorithm in detail [Nieh, '03]. Chandra et al. presents how to readjust the weights in their proportional sharing algorithm in a multiprocessor environment [Chandra, '00]. The scheduling scheme in this dissertation adopts a similar notion of proportional sharing in which applications' weights are determined based on a priority assessment model that reflects users' interest in applications. However, the proportional sharing employed in the presented scheduling scheme considers applications' resource needs in addition to their weights to achieve fair perceptual performance rather than fair resource distribution.

4. OVERVIEW OF SCHEDULING FRAMEWORK

An overview of the scheduling framework presented in this dissertation is described. Chapter 4.1 discusses the types and the characteristics of applications in thin-client display wall environments this dissertation is centered on and defines the goals to achieve. The design of the scheduling framework and the components that consist of the scheduling framework are explained in Chapter 4.2.

4.1 <u>Applications</u>

In thin-client display wall environments, the content-generating applications typically run on remote high performance computers such as visualization clusters or Cloud resources. The rendered visualization is then streamed to the display wall system as a series of image frames over a high-speed network. In this dissertation, applications refer to processes running in a display wall whose contents are being streamed over the high-speed network. Therefore, user interactions on the applications on a display wall are interpreted at the display wall system and sent to the content-generating applications over the network.

The types of applications in thin-client display wall environments can be distinguished further. In one case, the content-generating application has an optimal streaming rate that is known *a priori*. For examples, an application that streams a live video feed from a HD camera or a media player that streams a video over the network. Although the actual streaming performance can change depending on various conditions such as the available network bandwidth, the bandwidth needed for optimal performance can be derived beforehand. The resource requirement for the optimal streaming performance in this case can be calculated by multiplying the image size with the frame rate set by the application. In the second case, the application does not specify an optimal streaming rate and usually run in a best-effort manner. The image quality or the frame rate varies as a user interacts with the application and thus the amount of resources needed for the optimal performance cannot be determined beforehand. Assuming unlimited resources, the resource need for the optimal performance at a given time in this case can be derived by the application's actual resource utilization at a given time, which will vary as the user's interaction-rate changes. An example of this is a scientific visualization tool where users can pan, rotate, or scale the visualized data, requiring an update only when users interact with the visualization. Therefore, the amount of resources for the optimal performance is defined to be time-varying in this dissertation.

The goal of the scheduling framework is to fairly distribute system resources to applications to optimize their presentation qualities as perceived by users rather than ensuring fine-grained thread-level fairness from a system point of view. This is called *presentation fairness*. The quality of an application however is highly subjective and multi-dimensional. For example, the quality of a video game involves responsiveness and frame rate while the quality of an animation would be determined by image quality and frame rate. There are studies that focus on this matter [Kuan-Ta, '09; Wu, '09; Zixia, '12]. However, in the context where applications run at remote locations and stream their images to display walls, the quality of an application is defined as the ratio of the amount of resources being consumed by the application to the amount of resources it needs for optimal performance. For interactive applications where the amount of resource needed for optimal performance is not known a priori, this dissertation uses the application's current resource utilization as the basis to estimate the amount needed for optimal performance. Chapter 6.2 explains in detail how to estimate this. Prioritizing applications by estimating users interest in the applications is crucial to achieve improved user-perceived performance. The model-based approach to prioritize applications is discussed in detail in Chapter 5. Once the priority and the resource needs of an application are determined, a weighted max-min fair sharing algorithm with these two variables can be applied to fairly distribute resources in order to achieve the presentation fairness. Chapter 6.4 discusses this in detail.

4.2 Design

The scheduling framework consists of the Priority model, the Performance Monitor, and the scheduling algorithm. Figure 10 depicts the relationship between each component. The Priority model collects information about application states such as the application's window geometry, frequency of user interactions in order to assign a *priority*, reflecting the relative importance for each application. The Performance Monitor keeps track of performance measures from which it calculates the current resource utilization and estimates the resources need for the optimal performance of each application. The scheduling algorithm then uses the assigned priority and the application's resource need to allocate resources.

When an applications starts, the Performance Monitor acquires the resource need of the application. This information is either provided by the application itself or estimated by the Performance Monitor. For instance, the initial geometry of the application window can be used by the Performance Monitor to make an initial estimation of resource need of the application. Finally, the scheduler determines the initial amounts of resources that can be allowed to the application. The application's visibility, user interaction information, and resource utilization changes as users interact with the application. These can change the priority and the resource need for the optimal performance of the application. These two are used by the scheduler to determine the amounts of resource allowed for the application at every scheduling event.

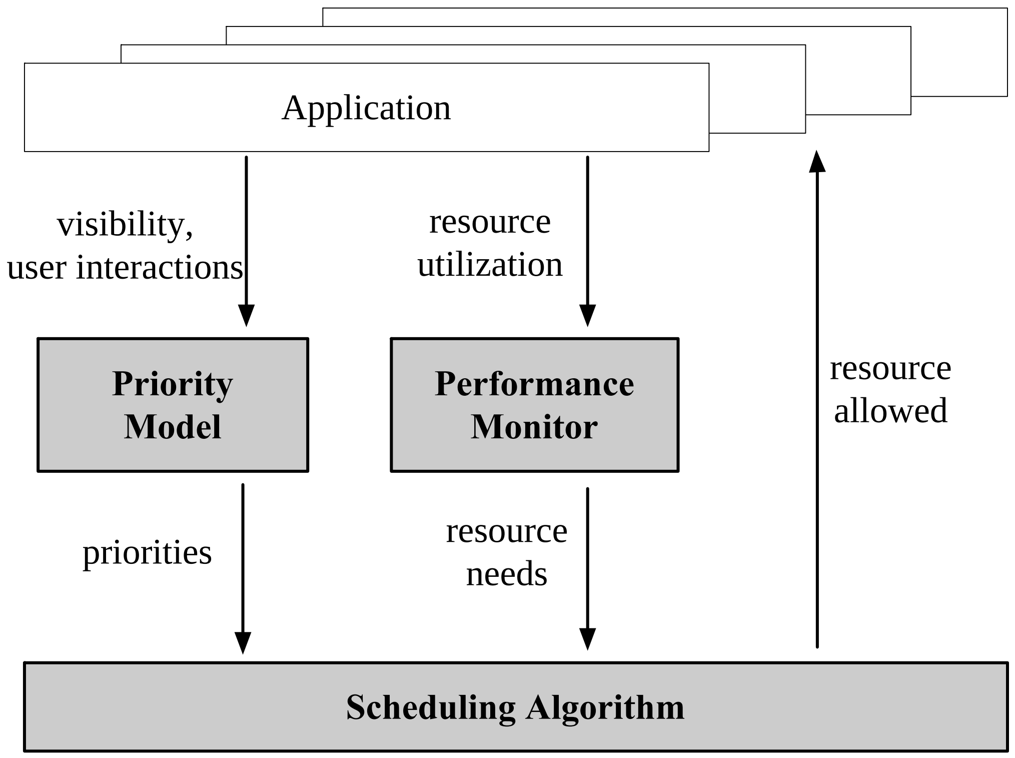


Figure 10. The components of the scheduling framework. Applications' information such as their relative visibility, frequency of user interactions, and resource utilizations are collected and processed by the Priority model and the Performance Monitor. The proportional sharing scheduling algorithm then determines the amount of resources allowed for each application.

5. PRIORITY MODEL

The Priority model describes the degree of users' interests in applications running on a display wall, thus the priority assigned to applications should reflect what users perceive to be important as accurately as possible. To achieve this, the model looks at multiple factors related to the current layout of applications. The model observes three factors and produces a numeric priority value for each application. The *Effective Visible Size* (*EVS*) and the frequency of interaction with an application indicate spatial and temporal importance of an application, respectively. The wall usage pattern describes the historical importance of specific areas of the display wall. These three factors are combined together to produce the priority value. Each of the three factors are discussed and quantified in the following sections.

5.1 Visual Factors

An intuitive visual factor to determine an application's importance is the application layout on the display wall as illustrated in Figure 11. The application layout can be categorized into four with two variables (the percentage window overlap and the number of application windows). A fair sharing might be useful in the tiled case (lower right corner) while more system resources need to be allocated to the application whose window is in the foreground in the stacked case (upper left corner) in Figure 11. However, prioritizing applications based on their visual factors can be simplified by using a single factor that tells how much of the application window is visible to users instead of enumerating various application layouts on the wall.

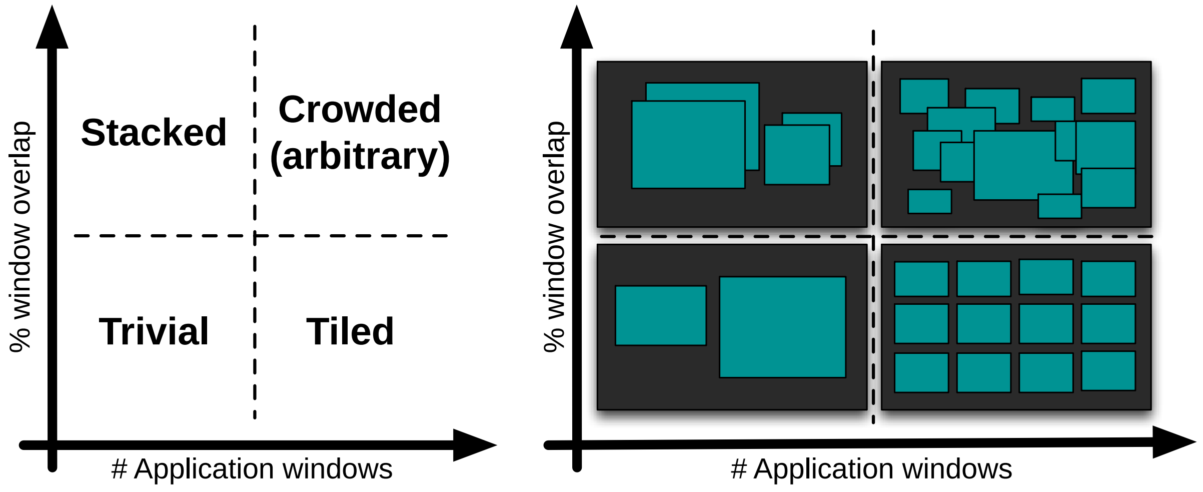


Figure 11. Application window layout on a display wall is categorized into four with two variables (the percentage window overlap and the number of application windows).

The visible window size of an application as an indication of user interest is straightforward. Even though an application might not be receiving user input, a large window size can imply high interest. Similarly, if an application's window covers a significant portion of the display wall, that application is more likely to draw users' attention. Visible window size is defined as the total size of the visible, non-occluded areas of the application's window in pixel. This can be easily calculated by subtracting the sizes of portions occluded by other applications. In this dissertation the size of effective visible area of an application's content window is said to *Effective Visible Size* (*EVS*) and *EVS* of an application *i* at time *t* is denoted with *EVS*(*i*, *t*). The value of *EVS* can range from 0 to a maximum of *i*'s window size. For instances, *EVS*(*i*, *t*) = 0, if an application *i*'s window is completely obscured by other windows at time *t*, and *EVS*(*i*, *t*) = *the size of i's window* if the entirety of *i*'s window is visible at time *t*. The *EVS* is the main factor to determine application's visual importance but how much an application revealing its contents has to be considered too. Let's assume a case where two different application windows A and B as shown in Figure 12. Application A’s window size is larger than B’s window size. However, large portion of A’s contents are hidden by B’s window. Let the window sizes of A and B are 180 and 100 respectably. The *EVS*(A) will be smaller than 180 due to the B’s window on top of the A’s window. Let *EVS*(A) be 100 in this example. The *EVS*(B) is equal to its window size because it is showing all of its content area. As a result, *EVS*(A) = *EVS*(B) giving the same visual importance to both A and B. In some cases a user might be interested equally in both A and B but generally it is safer to assume that a user is more interested in the application B than A in this particular application layout on a large display wall.

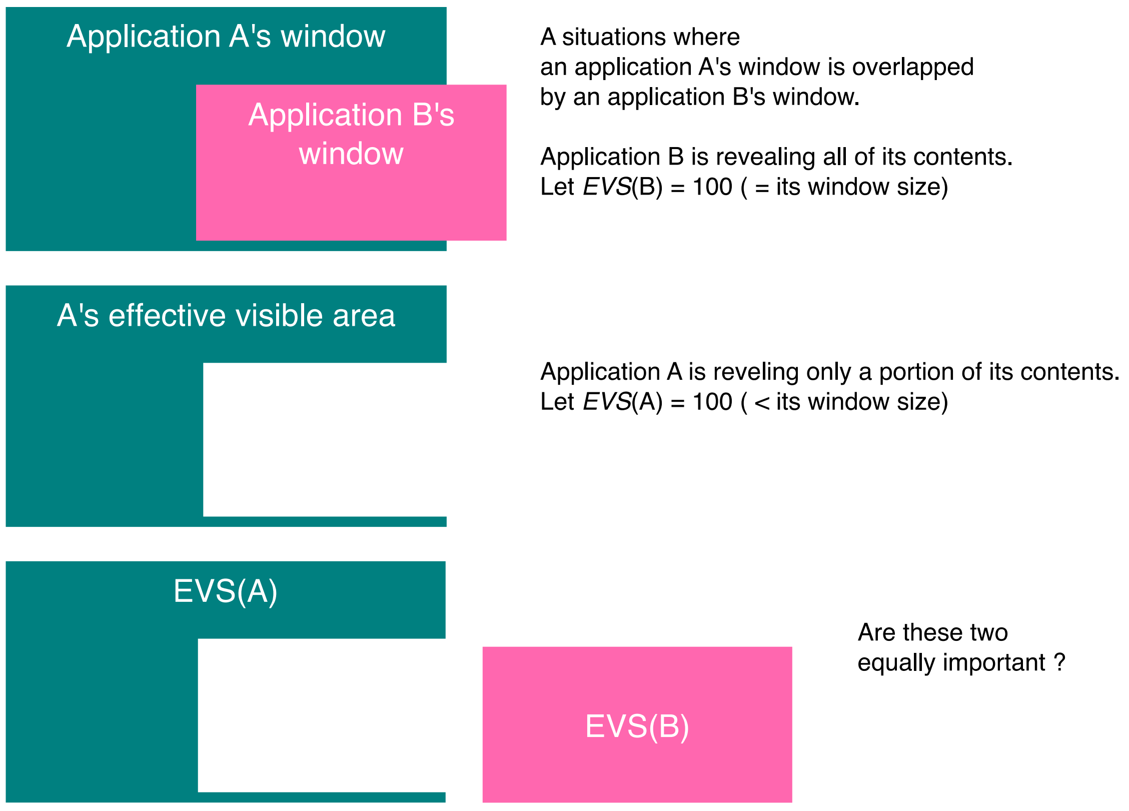

Application A's window
Application B's window
A situations where
an application A's window is overlapped
by an application B's window.
Application B is revealing all of its contents.
Let *EVS*(B) = 100 (= its window size)
A's effective visible area
Application A is reveling only a portion of its contents.
Let *EVS*(A) = 100 (< its window size)
EVS(A)
EVS(B)
Are these two
equally important ?

Figure 12. An example of two application windows A and B where A’s window size is larger than B’s window size. The B’s window is stacked over the A’s window causing a large portion of the A’s contents are not visible to users.

The *Exposure Ratio E* is the ratio of *EVS* to the application's window size. Thus *E* tells how much an application revealing its content on the display wall and it can be denoted by *E* = *EVS* / WindowSize. We multiply the *EVS* by the exposure ratio *E* to obtain the priority determined by the visual factor P_{visual}. An application that has large effective visual area and exposure ratio is considered to be visually important.

$$P_{visual} = E \cdot EVS = \frac{EVS^2}{\text{WindowSize}}$$

5.2 Interaction Factors

The *EVS* alone is not enough to reflect user intentions. Imagine a display wall with tiled application layouts and users are focusing mostly on the applications on the left as depicted in Figure 13. Or assume two application *i* and *j* where $P_{visual}(i, t) >> P_{visual}(j, t)$ at time *t*. A user could be interacting more frequently with application *j* while *i*'s priority is higher because it has a larger P_{visual} value. Typically, user interactions through input devices such as mouse, gyro mouse, touch, gesture, or keyboard indicate user's interest in an application directly.

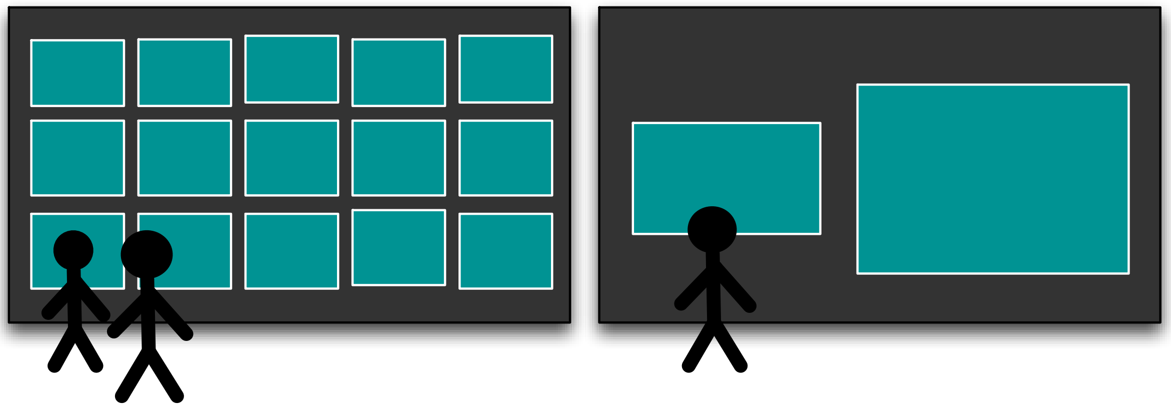


Figure 13. Examples of display walls where users are focusing more and interacting with the applications on the left.

However, it is hard to state exactly how much certain interactions on a specific application should increase (or decrease) its priority. There can be many different types of applications with different user interaction scheme. So, this dissertation provides a simplified mechanism to add interaction factor in the Priority model by letting application developers to call a function in a place where a user interaction event is handled in the application. The function simply increments the interaction counter for the application. Then the rate of changes in the number of interactions during a single scheduling interval is periodically monitored. This way we can tell how intense the recent interactions are for the application. The interaction factor $P_{interact}(i, t)$ is defined as the number of user interactions during the scheduling interval t-1 and t.

5.3 Wall Usage Pattern

Spatial layout and window arrangement patterns are likely to emerge if the display wall is used long enough. For example, in Figure 4 (d) where the wall is partitioned into two sections, the left section employs a tiled-layout which is appropriate for comparisons while the right section shows a single large application window. To get an insight into these patterns, imagine a virtual grid juxtaposed on the display wall, and use that grid to aggregate priority values of applications in each cell of the grid. The grid can be color-coded by the aggregated priority values forming a heat map; high priority values are indicated as high temperature. For example, in the top layout in Figure 14, most of the applications are positioned in the left portions of the wall, causing an increase in the temperature of the left portion of the wall. In the bottom case of Figure 14, the left side of the wall has higher temperature even though applications are scattered arbitrarily on the wall because users are interacting more with the applications on the left side. In these cases, if a user brings an application window from the right side of the wall (cold region) to the left side of the wall (hot region), then the application will get an immediate priority bonus.

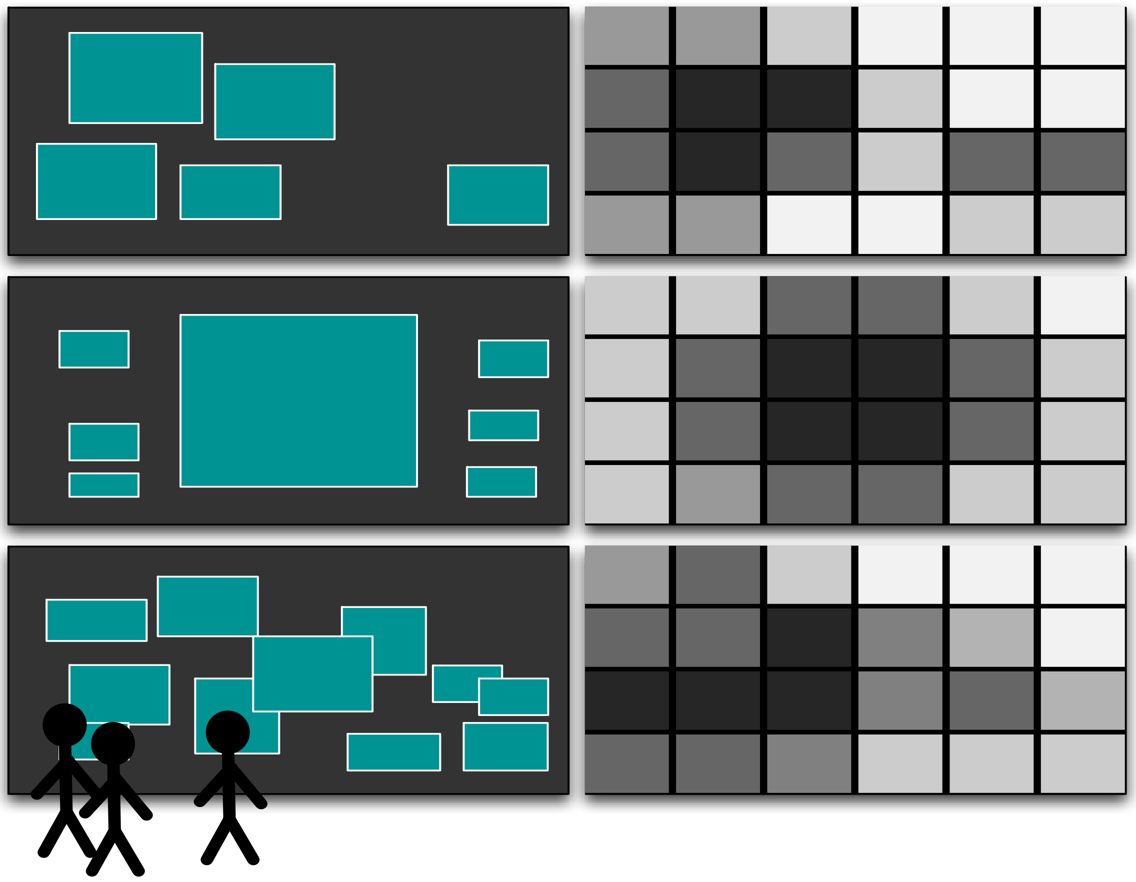


Figure 14. Examples of display wall usage and the corresponding black-hot heat map of the wall. Darker cell indicates higher aggregate priority values. In the top figure, applications are mostly positioned in the left portion of the wall. In the middle figure, one application is maximized in the center of the wall. In the bottom figure, the applications are arbitrarily scattered, but the users are mostly interacting with the applications in the left portion of the display.

A display wall is divided into multiple cells of a virtual grid. Each cell in the grid maintains a priority value calculated for the cell. Each cell *c* of the grid adds the priority value of each application *i* that overlaps with the cell at time *t*, proportion to the percentage overlap. The percentage overlap (%*overlap*(*c, i, t*)) is the ratio of the size of the region of the cell *c* covered by the application *i* to the cell's size. The temperature of the cell *c* at time *t*, *Temp*(*c*, *t*), is denoted as

$$Temp(c,t) = Temp(c,t-1) + \sum_{i \in L}\left(\left(P_{visual}(i,t) + P_{interact}(i,t)\right) \cdot \frac{\%overlap(c,i,t)}{100}\right)$$

where *L* is a set of applications whose window overlaps with the cell *c*. Note that *Temp*(*c*,*t*) only accounts P_{visual} and $P_{interact}$. This is because the visual and interaction factors are time-invariant whereas P_{temp} holds historical meaning. P_{temp} is meant to record only the spatial changes in users' interests as time goes by. Finally, the temperature of an application *i* and time *t*, $P_{temp}(i, t)$, is the proportion of the sum of the temperature values of the cells on which *i*'s window span. This is denoted as

$$P_{temp}(i,t) = \frac{\sum_{c \in I} Temp(c,t)}{\sum_{c \in G} Temp(c,t)}$$

where *I* is a set of cells under the application *i*'s window and *G* is a set of all cells in the grid.

5.4 <u>The Priority Function</u>

The final priority of an application is obtained by combining the three priority factors. The priority *P* of an application *i* at time *t* is defined as

$$P(i, t) = W_v P_{visual}(i, t) + W_i P_{interact}(i, t) + W_t P_{temp}(i, t)$$

where W_v, W_i, and W_t denotes weight factor for each components. In our framework, an absolute value of a priority is not important. The scheduler prioritizes resources based on a proportional basis, with priorities indicating application's relative importance at a given time. The priority proportion of an application *i* at time *t* can be easily calculated by

$$P_{prop}(i,t) = \frac{P(i,t)}{\sum_{i \in L} P(i,t)}$$

where L is the set of all applications in the system.

How to weigh each priority factor to produce a priority value for an application? Weighing each priority factor monotonously for all types of applications is not appropriate because different applications use different ways of presenting information and can possibly employ different interaction schemes. Thus, how to weigh each priority factor depends on the type of each application. For image-centric applications, the P_{visual} can be the most important factor while the $P_{interact}$ can be important for interactive applications. For example, an application such as a movie player where $P_{interact}$ can be very low can still receive enough resources by giving it a high W_v. A scientific visualization tool where a user frequently zoom, rotate, and move a rendered model will want to set high W_i. The P_{temp} will be useful when there is distinct wall usage pattern after long period of display wall usage. While an application developer can determine the application-specific weight factors to weigh each of the three priority factors, the inequality of each weight factors in general can be expressed as $W_i > W_v > W_t$ (where $W_i + W_v + W_t = 1$) based on the degree of straightforwardness of each priority component in reflecting users' interest.

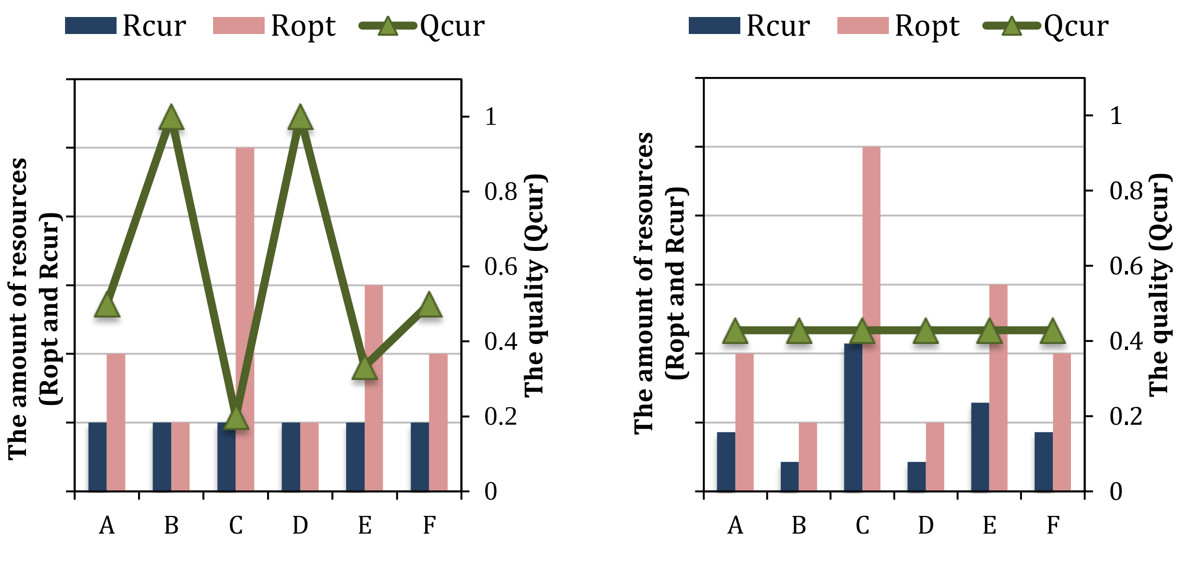
6. RESOURCE DISTRIBUTION

Once the Priority model assigns priorities to applications, the scheduler distributes system resources among application by adjusting their presentation qualities to ensure presentation fairness. In this dissertation, an application's presentation quality is the ratio of the amount of resources the application currently consumes to the amount of resources the application needs to achieve its optimal performance at a given time. This dissertation denotes the actual amount of resources an application i consumes at time t as $R_{cur}(i, t)$ and the amount of resources the application i needs for the optimal performance at time t as $R_{opt}(i, t)$. The presentation quality that the application i currently achieves at time t is denoted as

$$Q_{cur}(i, t) = R_{cur}(i, t) / R_{opt}(i, t) \qquad (1)$$

where R_{cur} can be obtained by measuring the amount of resources consumed by the application and R_{opt} is either provided by the application if the amount resources required for optimal performance is known a priori (such as applications streaming a video at a fixed frame rate) or derived based on R_{cur} for interactive applications. Chapter 6.2 discusses how to derive the amount of resources needed for optimal performance at a given time in the latter case. Presentation fairness is achieved by allocating resources so that the resulting presentation qualities (Q_{cur}) are in accordance with applications' priorities (applications with higher priorities achieve higher presentation qualities) rather than ensuring fine-grained fair sharing of resources (i.e. fair distribution of Q_{cur} rather than R_{cur}).

The Figure 15 illustrates the presentation fairness the scheduling framework in this dissertation tries to achieve. Assume six applications each have different R_{opt} as shown with red columns in the graphs. Operating systems ensure fair resource consumptions (R_{cur}) of all six applications assuming their priorities are equal. This results un-fair presentation qualities between the applications as shown with the marked green line in Figure 15 (a). The fair quality (presentation fairness) can be achieved if resources are allocated proportional to each application's resource need (R_{opt}) as shown in (b).



(a) fair distribution of R_{cur} (b) fair distribution Q_{cur}

Figure 15. The presentation qualities (Q_{cur}) of six applications A to F under different scheduling schemes. Each application has different amount of resource need represented by the bright red columns (R_{opt}) in each graph. The amounts of resources consumed by applications are represented with dark blue columns (R_{cur}). In (a), the amounts of resources are fairly allocated to all applications resulting unfair qualities due to the non-uniform resource needs of the applications. In (b), the amounts of resources are allocated proportional to each application's resource need resulting a fair distribution of qualities as shown by the flat marked line in the graph.

The scheduler also needs a system-wide variable indicating the total amount of available resources. It is because the scheduling framework determines the amounts of resources allowed for each application at every scheduling event instead of giving them an amount that is globally determined a priori. R_{TOTAL} denotes the amount of total available resources seen by the scheduler in the rest of the dissertation. Chapter 6.3 describes how we obtain this amount at every scheduling event.

6.1 The Demanded Quality

At every scheduling instance, the scheduler determines the maximum amount of resources allowed for each application. Thus, combined with R_{opt} of the applications, the scheduler sets the maximum presentation quality each application is allowed to achieve. This is called the demanded quality. The demanded quality set by the scheduler for an application *i* at time *t* can be denoted as

$$Q_{sched}(i, t) = R_{sched}(i, t) / R_{opt}(i, t) \tag{2}$$

where $R_{sched}(i, t)$ is the maximum amount of resources allowed for an application *i* as determined by the scheduler at time *t*. The Q_{sched} ranges from 0 to 1 because the scheduler does not demand resources more than the application needs ($R_{sched} \leq R_{opt}$). A value of 0 indicates that no resources are to be allocated for the application, which implies that application should idle. The scheduling framework defines a special case where an application can consume as much resources as it can utilize if Q_{sched} of that application is set to 1 ($R_{sched} = R_{opt}$). Thus, $\boldsymbol{Q_{sched}(i, t) = 100\%}$ indicates that the scheduler sets no limit on resource consumption for the application *i*, until $Q_{sched}(i, t + l)$ is set to a value less than 1. The amount of resources consumed by an application (R_{cur}) can be greater than the amount demanded (R_{sched}) and the amount derived for the optimal performance (R_{opt}) during the time period *l* under this special condition. This is because the scheduler can not know the global upper bound of R_{opt} of an interactive application. How much resources an interactive application needs varies as a user interacts. Even if the scheduler can set the global upper bound of R_{opt} , it is not desirable because the resource need varies as a user interacts. For example, assume the R_{opt} is set with an upper bound known a priori and the application's actual resource consumption (R_{cur}) is much less than the R_{opt} then the remaining resources (R_{opt} - R_{cur}) will be wasted. On the other hand, imagine a case where the system is overloaded and R_{opt} is not known a priori. Regardless of how actively a user interacts with the application (thus high priority for the application), R_{cur} of the application can not increase because the system is overloaded unless the scheduler finds out this particular situation and allocates more resources to the application (by taking resources from lower priority applications). The special case where Q_{sched} = 100% is to find this situation and properly allocate resources.

6.2 Estimating an Optimal Amount

The optimal amount of resources for an application (expressed by R_{opt}) is the amount of resources the application needs to achieve its optimal presentation performance. This variable is defined as a function of time. In non-interactive applications, the optimal amount of resources is known a priori, thus R_{opt} is a fixed constant at any given time. For instance, a live video feed will have a fixed frame rate. On the other hand, imagine a case where a visualization is being rendered at a remote server and the rendered images are streamed to the display wall. The visualization server renders and streams images only when users interact with the visualization, unless the user plays a predefined animation. When the users interact, the visualization server streams in a best-effort manner. When there is no user interaction however, the scheduler does not have to allocate resources because there are no images that are being streamed. By changing the amount of resources for the optimal performance to reflect the resource needs, which can vary as users interact, the scheduler can allocate resources to applications more effectively by allowing more resources to the ones that actually need those resources. Therefore, R_{opt} of an application has to reflect the resource need at a given time. To achieve this for interactive applications, the Performance Monitor estimates the amount of resource for the optimal performance at a given time based on the application's resource utilization.

```
L is the set of all applications in the system
$R_{opt}\_multiplier := M$
ESTIMATE_$R_{opt}$(t)
 1: for each $i$ in $L$
 2:     if $R_{opt}$ of $i$ is known a priori then
 3:         do nothing
 4:     else if $i$ is newly joined or woken up then
 5:         $R_{opt} := C$                                   // make an initial estimation
 6:     else if $R_{cur} = R_{opt}$ then
 7:         $R_{opt} := R_{opt}\_multiplier \cdot R_{cur}$   // the application could use more
 8:     else                        // $R_{cur}<R_{opt}$ or $R_{cur}>R_{opt}$ because $Q_{sched} = 100\%$
 9:         $R_{opt} := R_{cur}$
```

Figure 16. A function that estimates the optimal amount of resources for interactive applications. The first estimation occurs when the application is newly added to the system or when it is woken up from an idle state. The second estimation is to prevent a situation where the application's potential optimal performance (expressed by R_{opt}) is stuck in a local maximum when the system is overloaded.

Figure 16 illustrates pseudo-code that estimates and updates the optimal amount (R_{opt}) for interactive applications where their resource consumptions (R_{cur}) vary based on user interactions.

R_{opt} is first estimated with an initial value when the application starts or is woken up from an idle state. The initial value can differ by application, and the visual layout of the application's window such as its frame size can be used to set an initial value. Most of the time R_{opt} is simply updated to R_{cur}, except when the application is consuming the optimal amount ($R_{cur} = R_{opt}$). Notice that $R_{cur} = R_{opt}$ implies the amount of resources the scheduler allows (R_{sched}) is equal to the optimal amount (R_{opt}) which means the demanded quality (Q_{sched}) was set to 1 in equation (2). This state further implies that the application might be able to consume more resources as long as there are enough resources in the system. Recall that Q_{sched} = 100% sets no limit on resource consumption for the application for this case. If there exist enough idle resources in the system (the system is underloaded) then R_{cur} of the application is increased as long as the application can consume more, thus perform better. In this case, the condition $R_{cur} > R_{opt}$ can occur and R_{opt} will be increased to the R_{cur} by the line number 9 in Figure 16. The R_{opt} can reflect the increased resource need of the application by letting it to run in best-effort manner.

What happens if there are not enough resources in the system (the system is overloaded) when the Q_{sched} of an application is set to 100%? Since the R_{opt} is updated to R_{cur}, the optimal amount will not reflect the application's capability because the application is unable to consume more (R_{cur} cannot be increased because the system is overloaded). The scheduler allocates resources based on R_{opt} of the application. An application with small R_{opt} (meaning its resource need is small) will be allowed to use that small amount. In this case, the Performance Monitor increases the application's resource need ($R_{opt} := R_{opt}\_multiplier \cdot R_{cur}$) to prevent a situation where R_{opt} is bounded to a local maximum when the system is overloaded (line number 7 in Figure 16). The underestimated R_{opt} is corrected in this way. However, this can lead to an overestimation of R_{opt}. An overestimation, that can happen when the application does not consume the amount of resources it is allowed, is corrected simply by decreasing R_{opt} to the amount it currently consumes (line number 9 in Figure 16) in subsequent scheduling instances.

Employing a notion of the optimal amount of resources (R_{opt}) is necessary to provide presentation fairness where an application's quality is the metric for fairness. This in turn makes our scheduling scheme non *work-conserving* because the two estimations (the initial estimation and the estimated increase with R_{opt}*_multiplier*) can lead to resource waste when they are overestimated. However, the evaluation (Chapter 7) shows that the improved user experience in typical use cases outweighs the waste.

6.3 Total Available Resources

The amount of total available resources in the system (R_{TOTAL}) is bounded by hardware limit. However, having a fixed total available resources bounded to a particular hardware may not correctly reflect the capability of the system. For example, if the amount of total available resources is set to the aggregate bandwidth of all network links in the system, then R_{TOTAL} in this case reflects the upper bound only if all the applications in the system stream their contents over a network link. When applications run locally (i.e. running in the machine driving the display walls) then they will not utilize the network resources. In this case, the amount of total available resource is not necessarily the same as the aggregate capacity of the network links in the system. In general, a metric bounded to a particular hardware limitation is not feasible to abstract the notion of the amount of total available resources in the system

To obtain a better abstract notion of the total available resources, the amount of total available resources in the system seen by the scheduler is defined as the sum of the actual amount of resources (R_{cur}) consumed by applications running at the current moment. Thus R_{TOTAL} ranges from 0 to a constant value that indicates physical limit.

$$R_{TOTAL}(t) = \sum_{i \in L} R_{cur}(i,t) \tag{3}$$

However, equation (3) alone isn't enough for the scheduler to work properly. When a new application is added to the system, the scheduler cannot know the R_{cur} of the newly added application before running it. And the application cannot run before the scheduler determines a quality for that new application. Thus an estimation of R_{TOTAL} is needed whenever an application is added to the system. Figure 17 depicts how the R_{TOTAL} is updated. R_{TOTAL} starts with 0 and increased whenever new application introduced to the system. Although the R_{TOTAL} can be updated with estimations, it eventually converges to a constant value that reflects the hardware capacity in an abstract amount.

```
L is a set of all application in the system
GET_R_TOTAL(t)
  1: temp := 0
  2: inc := 0
  3: for each i in L
  4:     if i has newly added then
  5:         inc := inc + $R_{opt}(i, t)$                  // increase with estimation
  6:     else
  7:         temp := temp + $R_{cur}(i, t)$
  8: $R_{TOTAL}$ := MAX($R_{TOTAL}$, temp)                  // updated only with $R_{cur}$
  9: return $R_{TOTAL}$ + inc
```

Figure 17. A function that updates the amount of total available resources seen by the scheduler with the actual amount of resources applications currently consume. R_{TOTAL} is increased whenever an application is newly added to the system.

6.4 Weighted Proportional Sharing

A weighted max-min algorithm can be applied to achieve presentation fairness with the priorities and resource needs. The weighted max-min fair share algorithm is often used in QoS aware packet scheduling network as described in [Keshav, '97]. In a weighted max-min fair sharing, resources are allocated to applications proportional to their priorities (weights) as well as the amounts the applications demand. Thus the demand is first normalized by priority. No application is allocated more than it demands. An application whose allocation does not meet its demand is maximized as long as there exist resources to allocate.

```
$L$ is an ordered set of all applications (ordered by the priority)
$unitAllocAmount$ := [ ]
$fraction := F$

COMPUTE_UNIT_ALLOC_AMOUNT($t$)
  1:  $sum := \Sigma_{i \in L} P(i, t)$
  2:  for each $i$ in $L$
  3:      if $R_{cur}(i,t) = 0$ then
  4:          $L := L - \{i\}$
  5:          $sum := sum - P(i,t)$
  6:      $unitAllocAmnt[i] := fraction \cdot R_{opt}(i,t) \cdot P(i,t) / sum$
```

Figure 18. A pseudo-code of the algorithm that calculates the unit allocation amount to ensure fine scheduling granularity.

The scheduling algorithm takes the priorities assigned by the Priority model, the optimal amount of resources for each application, and the amount of total available resources in the system. The algorithm then assigns a demanded quality (Q_{sched}) for each application to adjust its quality (Q_{cur}), providing presentation fairness across the display wall.

Figure 18 shows a function used by the scheduling algorithm to determine the unit allocation amount (*unitAllocAmnt*) for each application in the scheduling loop. The *unitAllocAmnt* is an empty array that holds the fraction of the amount of resources that can be allocated for each application at every iteration in the loop in the COMPUTE_Qsched() function shown in Figure 19.

$$unitAllocAmnt(i,t) = fraction \cdot R_{opt}(i,t) \cdot \frac{P(i,t)}{\sum_{i \in L} P(i,t)} \tag{4}$$

where L is a set of all applications. The values in the *unitAllocAmnt* array differ by applications and are calculated for each application from its R_{opt} and the priority proportion at a given time. The *unitAllocAmnt* of application i at time t is defined as the fraction of $R_{opt}(i, t)$ multiplied by i's priority proportion at time t in (4). An application's R_{opt} is first multiplied by the application's priority proportion. This means, at every iteration in the scheduling loop, an application receives only a portion of the optimal amount based on its priority proportion. Then each application's portion of the amount it receives is further fragmented by the global variable *fraction*. This is to ensure fine granularity in resource allocation at a cost of increased execution time of the algorithm.

The scheduling algorithm is illustrated with partial pseudo-code in Figure 19. The *array_*R_{sched} is an empty array that will hold the values of R_{sched} in (2) for each application. The COMPUTE_Qsched() starts with obtaining the current R_{opt} and the *unitAllocAmnt* for each application and the R_{TOTAL}. Then R_{sched} for each application is calculated progressively in the loop (line 11-12) until no more resources can be allocated to any application. This is to maximize the amount of resources the lowest priority application can be allocated (thus it's a weighted max-min algorithm). The termination conditions can arise when either all the available resources are allocated (line number 8 in Figure 19) or all the application is allocated with the amount equal to their R_{opt} (line number 9 in Figure 19). Once the scheduler sets the quality of an application as in line number 15, each application i adjusts its resource consumption to comply with the scheduler's demand.

```
L is an ordered set of all applications (ordered by the priority)
array_R_sched := [ ]
unitAllocAmount := [ ]
fraction := F

COMPUTE_Qsched( )
 1:  t := NOW
 2:  for each i in L
 3:      array_R_sched[i] := 0
 4:  ESTIMATE_R_opt(t)
 5:  COMPUTE_UNIT_ALLOC_AMOUNT(t)
 6:  rt := GET_R_TOTAL(t)
 7:  forever
 8:      if rt ≤ 0 then break
 9:      if for each i in L  array_R_sched[i] = R_opt(i, t) then
10:          break
11:      for each i in L                               // allocate progressively
12:          array_R_sched[i] := array_R_sched[i] + unitAllocAmnt[i]
13:          rt := rt - unitAllocAmnt[i]
14:  for each i in L                               // set the demanded quality
15:      Q_sched(i, t) := array_R_sched[i] / R_opt(i, t)
```

Figure 19. Partial pseudo-code of resource scheduling algorithm. The algorithm progressively allocates small amounts of resources, proportional to an application's priority until no more resources are available, or until all applications receive the resources needed for their optimal performance.

To find the worst case running time of the algorithm, let the R_{TOTAL} be infinite. The algorithm will terminate only after all the applications are allocated with the amounts that they require. Formally, the algorithm terminates when for all application A, R_{sched}(A) = R_{opt}(A). Then the maximum number of iterations the forever loop (line number 7) is determined by the lowest priority application A and the size of the *fraction F* defined globally as shown in Figure 18. Therefore, the maximum number of iterations of the outer loop (line 7-13) of the algorithm shown in Figure 19 is R_{opt}(A, *t*) / *unitAllocAmnt*[A, *t*]. Using the equation (4), we obtain

$$num\_iter(t) = \frac{1}{fraction \cdot \frac{P(i,t)}{\sum_{i \in L} P(i,t)}} \tag{5}$$

where *i* is the application with the lowest priority at time *t*.

The ESTIMATE_R_{opt}() , COMPUTE_UNIT_ALLOC_AMOUNT(), GET_R_{TOTAL}(), and the inner loop of the algorithm (line 11-13) takes O(n) time where n is the number of applications in the system. Therefore, the worst case running time of the algorithm is O(*num_iter*(*t*) * n) using the equation (5).

7. EVALUATION

This chapter describes two experiments to demonstrate that the presented scheduling scheme achieves presentation fairness on a display wall with non-interactive as well as interactive applications, and with multiple users interacting with the display wall simultaneously. The tiled-display wall system employed in the experiments consists of 18 LCD displays, as shown in Figure 3, with a total resolution of approximately 18.8 megapixels (8,196 × 2,304 pixels). A single machine equipped with dual Intel X5650 quad core processors, 12 gigabyte of main memory, and 3 Nvidia GeForce GTX580 dual DVI graphics was used to drive the entire display. A separate, equally powerful machine is used to simulate content-generating applications that stream images to the display wall system over a high-speed network to simulate a thin-client display wall environment. The two machines are network connected with a 10Gbps optical switch. The bit-rate of an application is used as a metric for the amount of resources. Thus the amount of resources an application consumes (R_{cur}) indicates its image streaming bandwidth in bits-per-second. Both machines were run by a 64-bit Linux operating system (kernel 3.1).

The first experiment evaluates the presented scheduling scheme with non-interactive applications streaming at a fixed rate. Thus, the optimal amounts of resources (streaming bandwidth) for these applications are known a priori (R_{opt} is constant). The second experiment evaluates the scheduling scheme with interactive applications, and with multiple users simultaneously interacting with the display wall, causing a variable demand on system resources (R_{opt} can vary). In both experiments the display wall system was overloaded to simulate heavy usage. Each experiment is performed twice, once with the presented scheduler running, and a second time without the presented scheduler, leaving the system to rely solely on the operating system's scheduler.

7.1 Presentation Fairness with Fixed R_{opt}

In the first experiment, the effectiveness of the scheduling scheme with non-interactive applications is evaluated. Ten applications with different image sizes (shown on the X axis in Figure 20) are run simultaneously, streaming their contents to the display wall at 30 frames per second. Thus the bandwidth needed by each application for optimal performance (R_{opt}) is their image size in bits multiplied by 30 Hz. Although the applications' frame sizes are fixed, their window sizes on the display wall, which determines their P_{visual}, can be arbitrary.

Imagine a case where a user wants to compare multiple time-varying visualizations and the bandwidth requirements for the visualizations are not uniform. If the system does not have enough network bandwidth for all of the visualizations the user wants to compare simultaneously, the user will experience disparate perceptual performance because the system tries to achieve low-level fairness between the visualization even if their resource needs are different. The scheduling scheme presented in this dissertation tries to achieve fairness perceived by a user with the approach explained in previous chapters.

To demonstrate presentation fairness, the application windows are arranged in a tiled mode as illustrated in Figure 4 (a), thus giving the applications the same priority. Therefore P_{visual} is the same for all applications, $P_{interact}$ is 0 and, P_{temp} is negligible in this case because the application layout is static. The flat line in Figure 20 shows that the scheduler indeed assigns equal priority proportions to all applications.

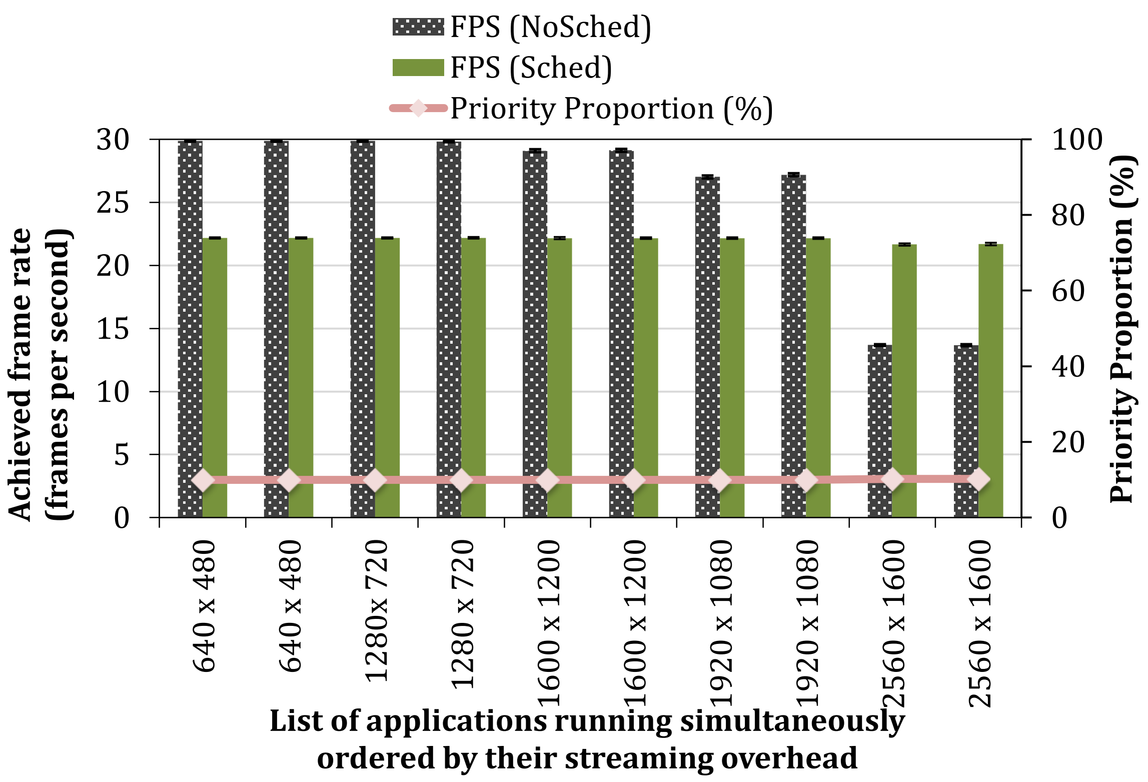


Figure 20. Achieved frame rate for applications used in the experiment under the operating system's scheduler (*NoSched*), and with the presented scheduler running (*Sched*) shown with standard erros. The X axis lists frame sizes of the 10 applications which are run simulateneously in the experiment. Application windows are arranged in a tiled mode as illustrated in Figure 4 (a) giving them equal priority. The flat line shows indeed that the scheduler assigns equal priority proprtions to all applications.

When the presented scheduling scheme is not running (*NoSched* condition), the operating system distributes resources evenly between applications. Since applications have varying frame sizes, this fine-grained distribution of resources leads to diverging performance as evident in Figure 20. Applications with larger frame sizes suffer a big performance hit with their frame rate dropping below 15 FPS, while applications with smaller frame sizes achieve their optimal 30 FPS. This disparity in performance is particularly evident to users, which detracts from the user experience. On the other hand when the presented scheduling scheme is in effect (*Sched* condition), all applications achieve a comparable performance with their frame rate around 22 FPS, thus achieving presentation fairness. This is because the Priority model assigns the same priority to all applications, and the scheduling algorithm allocates resources taking applications' R_{opt} into account.

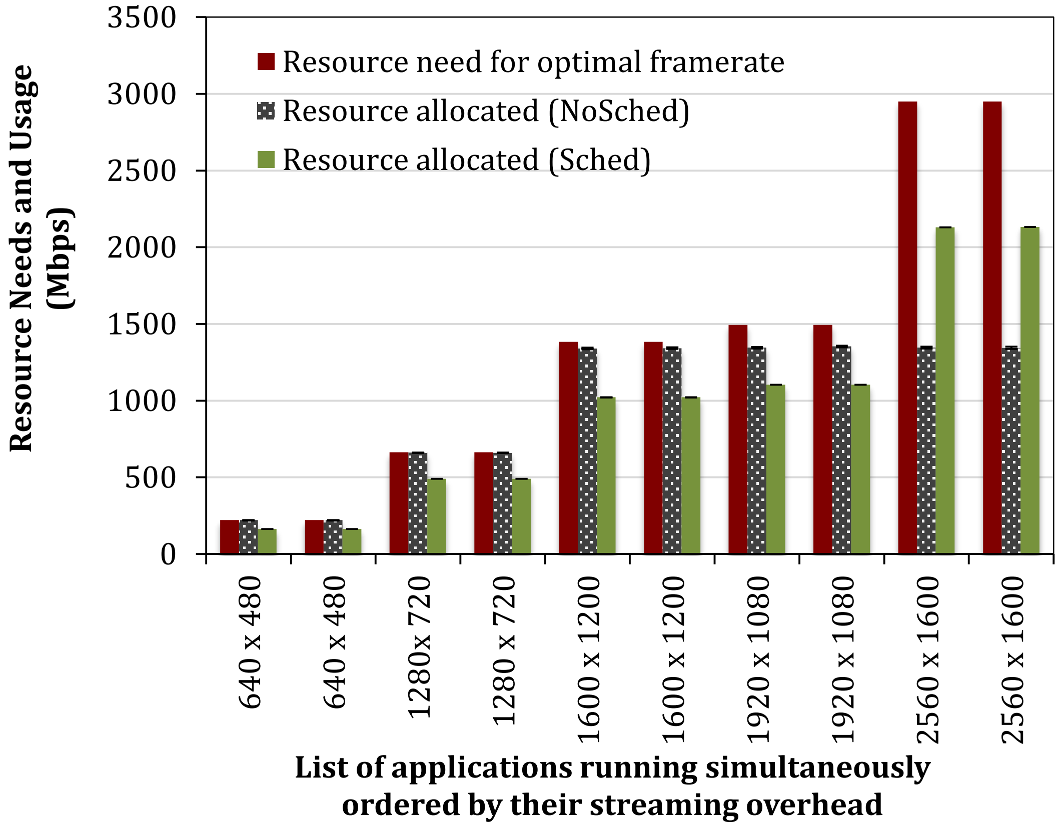


Figure 21. The resource needs for applications (dark red columns) and the amounts of resources actually utilized by applications are shown with standard errors. While the four applications that have relatively low resource requirements receive the amounts they need, the resources are allocated evenly for the rest of the applications regardless of their different resource needs under *NoSched* condition. The resources are distributed based on the applications' resource needs (R_{opt}) under *Sched* condition.

Figure 21 illustrates the resource needs for applications and the amounts of resources actually utilized by applications in this experiment. While the four applications that have relatively low resource requirements receive the amounts they need, the resources are allocated evenly for the rest of the applications regardless of their different resource needs under *NoSched* condition. This is because the *Non-clairvoyant* operating system scheduler is unaware of the different resource need of each application. The resources are allocated in a manner that it is proportional to the applications' resource needs (R_{opt}) under *Sched* condition.

7.2 Interactive Application (User Study)

Imagine a scenario where a user interacts with a scientific visualization tool by rotating, panning, and scaling a 3D model, or a piece of video production software where the user works with multiple media assets, traverses video frames, make rough cuts, etc. The application's responsiveness (as determined by the time the application takes to update its content from the moment of user interaction) is crucial to meeting the users' expectation for these types of applications. This scenario is simulated in the user study in order to evaluate the effectiveness of the presented scheduling scheme with multiple interactive applications.

The applications utilized in the user study comprised of a ***streamer*** that runs on a separate machine and streams images over network to a ***receiver*** that runs on the display wall system, which renders the user interface. Also the receiver applications are referred as user applications throughout the remainder of the dissertation. The streamer is analogous to scientific visualization software that runs on a high-performance computer and generates visual contents. The receiver is analogous to a corresponding GUI process that runs on the display wall system displaying the contents it receives from the visualization software (Figure 22). The streamer streams images to the receiver in a best-effort fashion whenever a user interacts with the receiver. The size of the image streamed by the streamer is fixed at 2560 × 1600 × 24 bits but its frame rate varies and is determined by the rate of user interaction (R_{cur} and R_{opt} vary as the user interacts). Therefore, the P_{visual} of the applications is fixed while the $P_{interact}$ changes as users interact. The scheduler determines R_{opt} based on R_{cur}, which changes depending on user interaction rate as discussed in Chapter 6. The receiver process (user application) conforms to the Q_{sched} determined by the scheduler by altering its frame rate. During the experiment, the display wall system was overloaded with 6 non-interactive applications streaming a 2560 × 1600 video sequence with 24 bits per pixel at a fixed rate of 30Hz (R_{opt} = 2,949.12 Mbps). We refer to these applications as the ***overhead***. A scenario this dissertation adopts with the notion of the overhead for the user study is to simulate a case where users are interacting some of the applications on the display wall that has many application instances. As discussed earlier, the system will allocate uniform amount of resources to all of the applications regardless of their different resource needs and what users are interested in at the moment. What would happen if all applications have to be treated equally important even though users are interacting with only some of the applications? The presented scheduling scheme will ensure fair resource allocation based on the applications' resource needs as shown in Chapter 7.1.

7.2.1 Task

The user study comprised of single and multi-user interaction with user applications on the display wall within groups of 3 subjects at a time. Each user interacts exclusively with a single receiver dedicated to that user using a mouse. The user application's window is displayed on the display wall, with subjects sitting side-by-side approximately 8 foot in front of the display wall. An example of the user study setup is shown in Figure 23. The user application presented the user with a target acquisition task in which the user is asked to move the mouse cursor and click on a target appearing in a random location inside the user application's window. An example of the user application's window is shown in Figure 22. The user application's window is updated only when a new frame is received from the streamer. Therefore, smoothness of the cursor movement, which is the visual feedback the user receives, depends on the frame rate, which will ultimately influences subject performance. If the user application's frame rate is too low, the user's interaction will be lost thereby the user will experience stuttered pointer movement making the user to hard to precisely position the pointer on the target. The rationale behind this task is that performance in the target acquisition task will demonstrate the responsiveness of the system. This will in turn reflect objective performance as well as subjective user experience in more complex scenarios such as scientific visualization and interactive, multimedia applications.

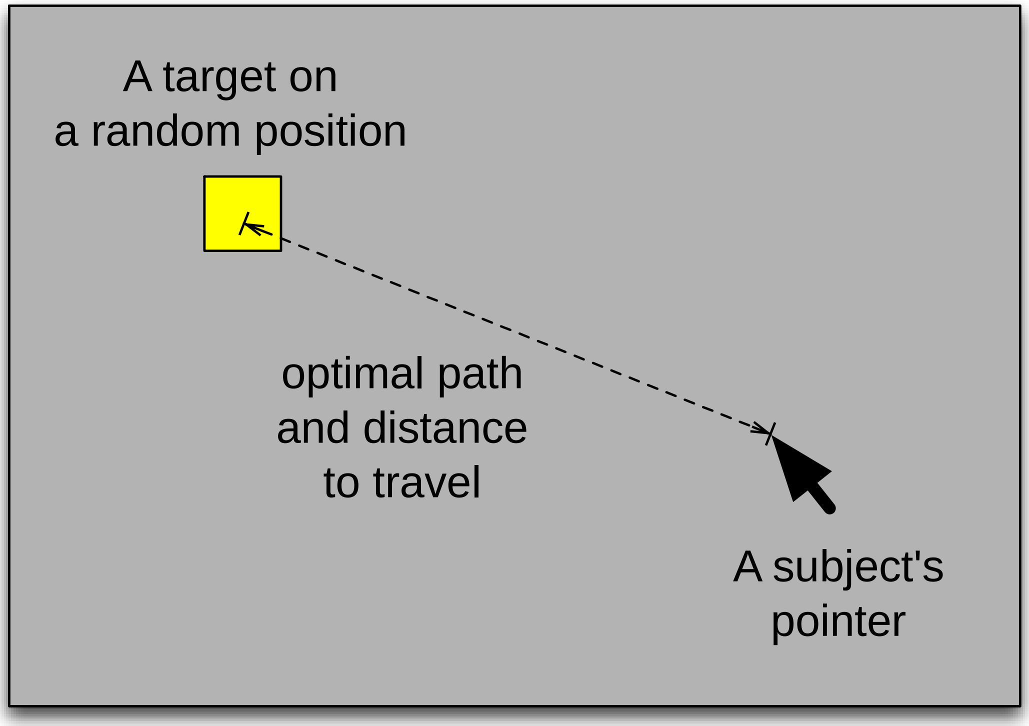


Figure 22. An example of the user application window (receiver's GUI) is shown with the target and the user's pointer. A yellow rectangle target appears on a random position on the application window. A subject is asked to click the target with his/her pointer as fast as possible.

7.2.2 Procedure

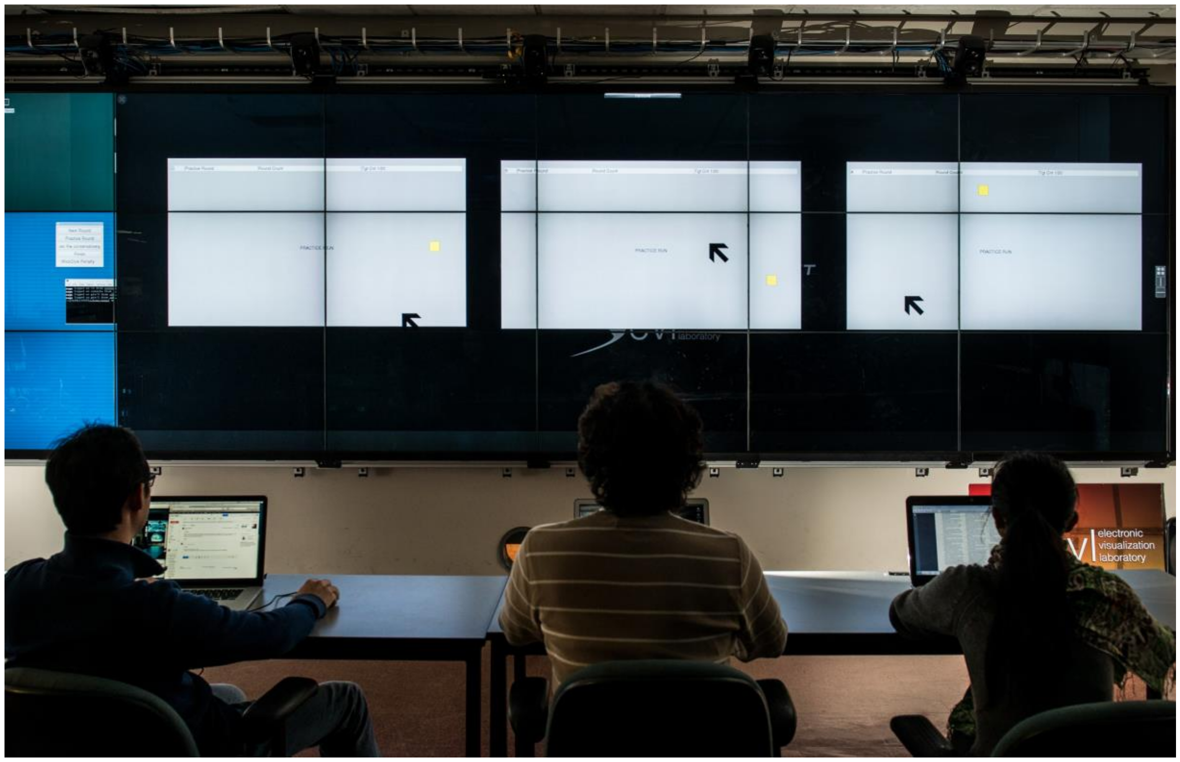


Figure 23. An example of the user study setup. Three users are sitting in front of the wall and interacting with their applications by connecting their laptops to the SAGE-Next.

18 subjects were recruited for the study. All subjects were computer science students (both graduate and undergraduate). Subjects are divided into groups of three, with a total of 6 groups, which referred to as groups A through F. Each group goes through a series of 7 rounds to vary the number of users interacting simultaneously. In the first three rounds, a single subject interacts with the system to perform the task (one of the three subjects in the group per round). In the second set of three rounds, two subjects perform the task simultaneously. In the final round all three subjects perform the task simultaneously. The 7 rounds are repeated twice under two different conditions: once with our scheduling scheme running (referred to as *Sched* condition), and a second time without our scheduling scheme (referred to as *NoSched* condition), leaving the

system to rely solely on operating system scheduling. This order is balanced across the 6 groups. (groups A, B, and C started with the *Sched* condition, while groups D, E, and F started with the *NoSched* condition).

7.2.3 Metrics

For each subject, the hit latency (the time it takes the subject to move the mouse cursor and successfully click the target from the moment it appears in the subject's assigned window) and the miss count (the number of clicks that missed the targets) are measured to compare the subjects' interaction performance with the presented scheduling scheme and operating system default. These indicate the responsiveness of the system that has to support multiple interactive applications simultaneously. Also the user applications' (receivers) frame rates are measured as subjects interact. And lastly, the overall resource utilization is measured to compare the presented scheduling scheme's reduced resource utilization (due to the resource estimation) with operating system default.

7.2.4 Results

A 2 (factor #1: *NoSched* versus *Sched* conditions) x 3 (factor #2: 1, 2, and 3 users interacting simultaneously) factorial ANOVA is computed to see the differences between average hit latencies under various conditions. The result indicates that there are significant main effects (factor #1: $F(1,3234) = 927$, $p < .000$, factor #2: $F(2,3234) = 6.12$, $p = 0.002$) but no interaction effect ($F(2,3234) = 0.13$, $p = 0.88$). Figure 24 shows the average hit latency of all groups with and without the presented scheduling scheme. With the presented scheduling scheme, the average hit latency of all subject groups is reduced by approximately 28% while the system is overloaded with the overheads as explained in Chapter 7.2. This increased user performance is due to the higher frame rate achieved with the scheduling scheme as shown in Figure 25 at a cost of reduced frame rates for overheads which were not interacted by users. The user applications achieve minimum of approximately 17.5Hz with our scheduling scheme. This is higher than a frame rate threshold (10~15Hz) where human performance can be adversely affected [Apteker, '95; Chen, '07; Claypool, '09; Gulliver, '04]. Even with a high workload that might run in background as simulated with the overheads in the experiment and increasing number of users interacting simultaneously, the presented scheduler was able to maintain a sufficient frame rate, which helped subjects maintain their task performance.

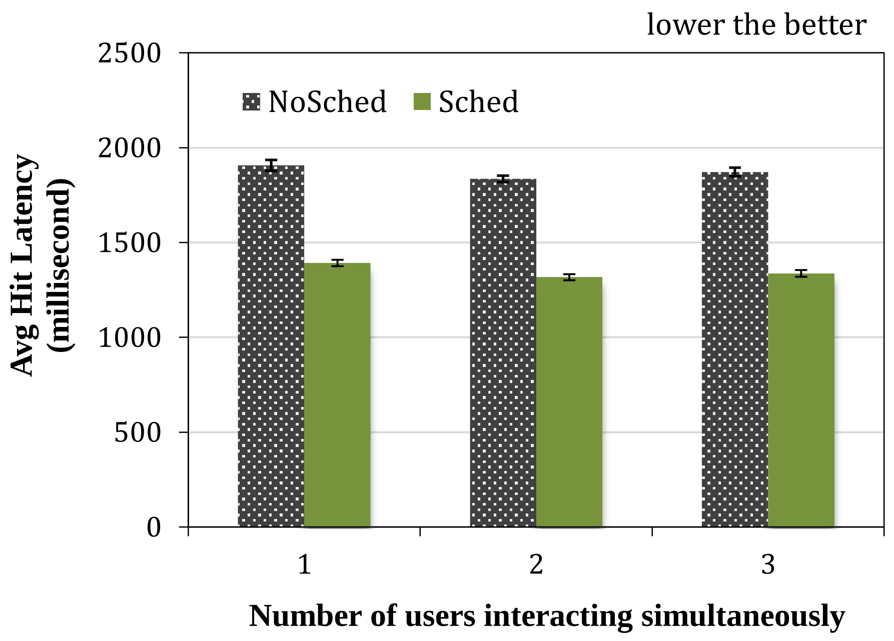


Figure 24. The average hit latency for all groups with and without the presented scheduling scheme are shown with standard errors. The average hit latencies are ~28% better with the presented scheduling scheme.

When the scheduling scheme is not running (*NoSched* condition), all the streaming instances (user applications that subjects were interacting and the overheads that have fixed streaming rates) are treated equally by the operating system. This results in poor interaction performance for the applications that are interacted by the users as shown in Figure 24 and Figure 25.

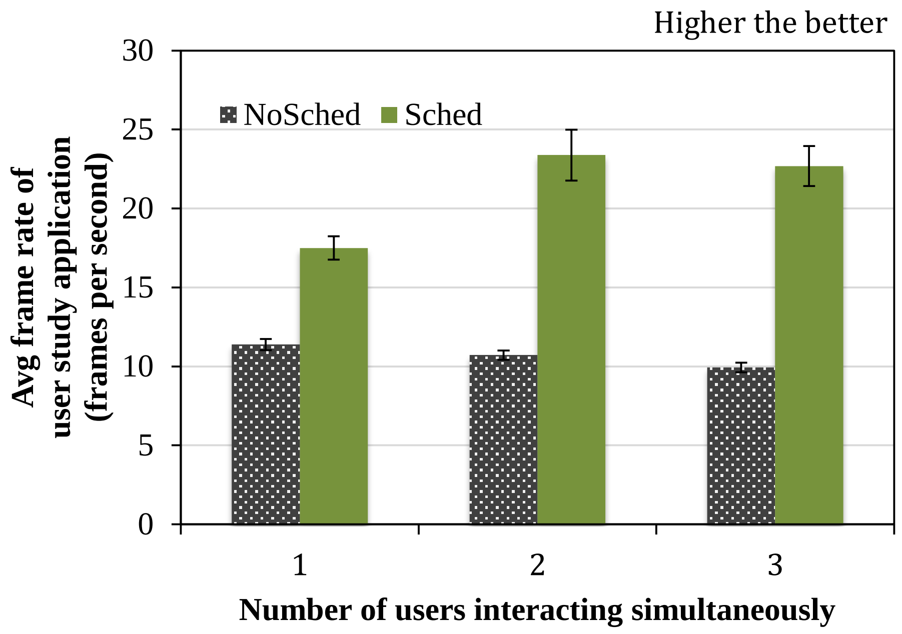


Figure 25. The average frame rate achieved by user applications are shown with standard errors. The frame rates slightly dropped as the number of users increased in the *NoSched* condition, where as the frame rates remained high with the presented scheduling scheme ensuring better user interactivity even when the system is overloaded.

The aggregate number of missed clicks of all subject groups was reduced by 19%, 42%, and 33% for one, two and three users, respectively (shown in Figure 26). However, the difference in the average number of missed clicks is insignificant as shown in Figure 27. During the user study, the author observed that the number of missed clicks is highly dependent on subject interaction characteristics rather than resource scheduling policies. Users who are careful in clicking targets miss the target less often whether the presented scheduler is employed or not.

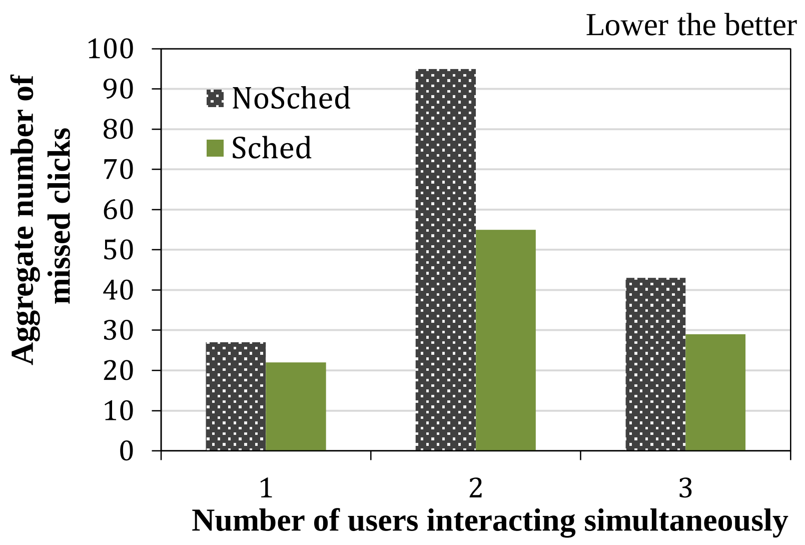


Figure 26. The aggregate number (total counts) of all subject groups' missed clicks with and without the presented scheduling scheme.

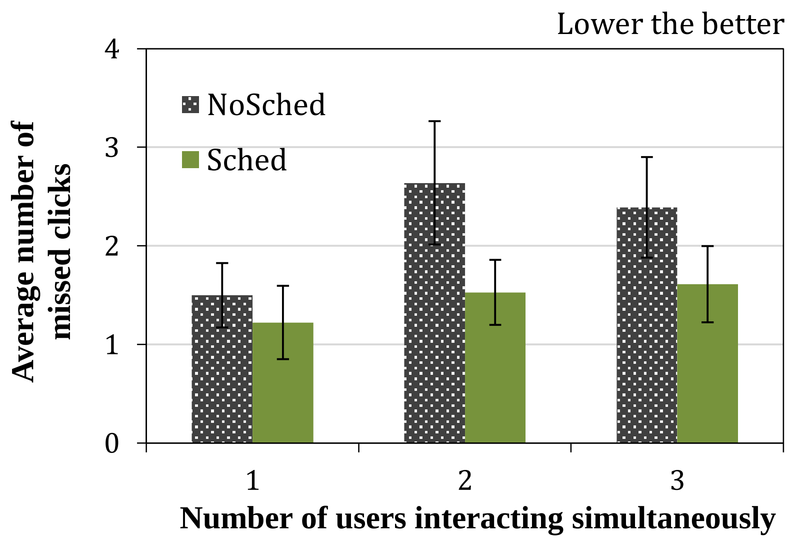


Figure 27. The average number of missed clicks with and without the presented scheduling scheme shown with standard errors.

Figure 28 depicts the total resource utilization breakdown. The graph shows that fewer resources are allocated to the overhead, allocating more to user applications under the presented scheduling scheme as the design dictates, which ultimately led to the improved user performance. As expected, the overall resource utilization of the presented scheduling scheme is lower than the operating system's scheduler. This is because resources are allocated based on an application's R_{opt}, which is not known a priori, and needs to be estimated based on its current performance. In current R_{opt} estimation mechanism, the user's interaction characteristic is the major factor determining the precision of R_{opt} estimation and the estimation tends to be preciser (meaning the R_{opt} precisely reflects the amount of resources a user will utilize thereby the difference between the R_{opt} and the R_{cur} can be small) when the rate of changes in user interactions is steady, thereby increasing resource utilization.

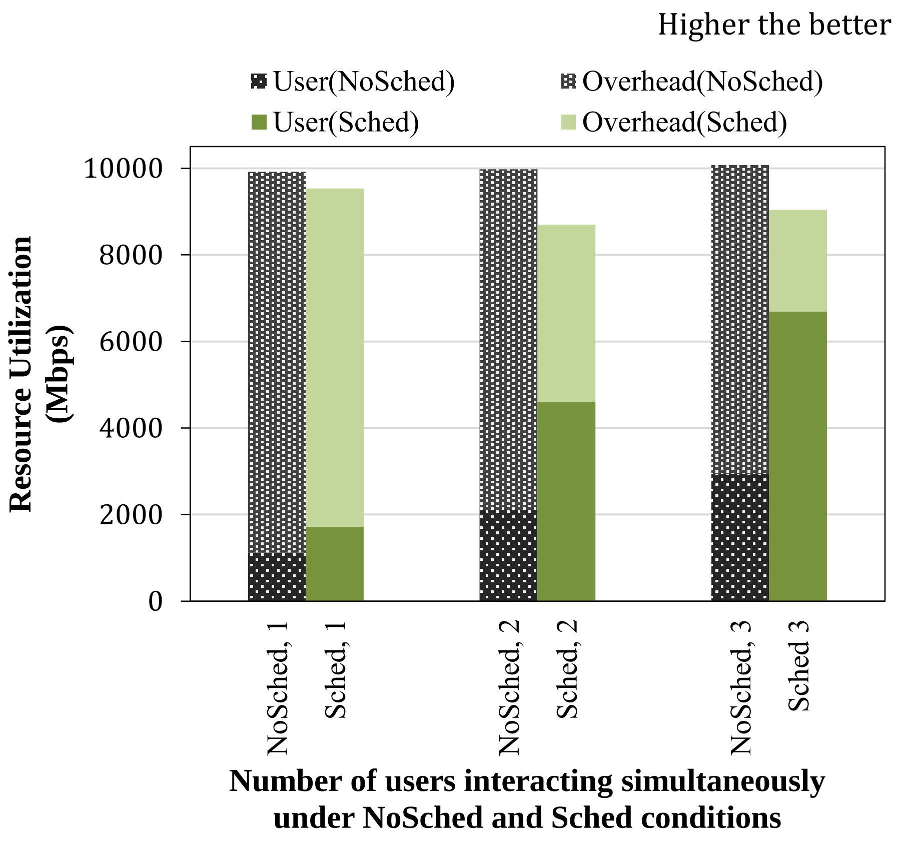


Figure 28. The average of total resource utilization breakdown with and without the presented scheduling scheme. The graph indicates that more resources are utilized by user applications under the presented scheduling scheme.

8. CONCLUSION

Thin-client display wall systems are used for displaying multiple high-resolution visualizations that are rendered at remote resources such as high-performance computers and storage cloud. Modern thin-client display wall systems provide multi-user collaborative environments by enabling multi-user interactions with multiple visualizations simultaneously. While display walls traditionally ran on a computer cluster, recent hardware improvements in multi-headed graphics hardware allows for display walls that can be driven by a single computer. However, this poses challenges in resource management due to the limited capability of a single machine compared to a cluster.

In this dissertation, a novel multi-user centered resource scheduling scheme for collaborative display wall environments is presented. Unlike traditional resource scheduling in modern operating systems, the scheduling scheme presented in this dissertation adopts a user-centered scheduling to maximize user-perceived performance, favoring applications that are most likely to draw user attention when the system is overloaded.

The presented scheduling scheme ensures the presentation fairness by considering applications' visual and interaction factors as well as resource needs of the applications. A Priority model is used to describe the degree of users' interest in applications on the display wall. The effective visible size of an application's window, the frequency of user interactions, and the wall usage patterns are used to determine the priority of an application. For interactive applications where the optimal resource requirement is time-varying and can not be determined a priori, the scheduler estimates these amounts based on current application performance determined by user interactions. The scheduler then finds a proportionally fair distribution of system resources and adjusts resource consumption indirectly by modulating presentation qualities of applications.

Experimental results show that the presented resource scheduler achieves presentation fairness in non-interactive applications where resource needs are known a priori. This is achieved by allocating resources to applications based on the amounts of resources applications need to achieve optimal performance. A user study was conducted to evaluate the effect of the scheduler on user performance with interactive applications running on an overloaded display wall. The user study shows improved user performance in a target acquisition task with the scheduling scheme over general-purpose operating system scheduling. The resources are allocated based on user interactions and estimated resource needs of the applications under the presented scheduling scheme while the general-purpose operating system allocates resources regardless of the user interactions and the resource needs. This demonstrates the effectiveness of the presented scheduling scheme when employed in interactive tiled display walls that are used in collaborative settings where multiple users interact simultaneously with the system.

9. FUTURE RESEARCH DIRECTION

In the future the author plans to extend the Priority model. Currently, the application's visible window size and the frequency of interaction with applications are major factors in determining the priority of the application. The contribution of these two factors is reconsidered only when a user actually interacts with the application. However, a user may desire high performance for an application even though he/she is not currently interacting with the application. To address this case, the Priority model could anticipate user intention without relying on application's internal states by monitoring the users. For example, tracking devices can be used to capture a user's interest in applications by sensing the user's head orientation or his/her location relative to the application. This information can then be incorporated into the Priority model. Furthermore, the current model treats each visualization independently, which may not be the case when multiple different visualizations of the same data need to be analyzed.

The scheduling scheme presented in this dissertation maintains the notion of an optimal amount of resources for an application based on its current resource consumption. In particular, when the system is overloaded, the scheduler increases the optimal amount for an application assuming that the application might be able to consume more. As a result the resource can be wasted when the amount is overestimated. Thus the scheduling framework is not *work-conserving*. Precise estimation of the optimal amount of resources for an interactive application is crucial to keep the system's resource utilization high. A Kalman filter [Brown, '97] where an estimation of variables of interest is obtained using a recursive algorithm might be used. Or an online application profiling technology might be adapted to provide better estimation of resource requirements. However, this is a hard problem for an application whose resource needs can vary greatly as user interacts because the system cannot precisely predict what a user will do.

Also, the optimal amount of resources for an interactive application can have lower and upper bound in order to keep the estimation in a reasonable range. For example, an interactive application would not want to allow its frame rate below certain lower bound (such as 15Hz) to ensure reasonable user performance. A video editing application may set an upper bound to its maximum frame rate. The application can skip some frames instead of increasing its frame rate when a user is scrubbing a video, for instance.

The types of the resources an application requires can be diverse. Therefore, defining a global metric that can be applied a various set of application is challenging. However, a better resource abstraction can make the scheduling scheme to support wider range of applications. Also the current resource consumption rate of an application might not precisely reflect the application's quality as perceived by users. However, a better abstraction for the application quality could be achieved given a set of different types of applications that are mostly used in collaborative display wall environments.

CITED LITERATURE

[Anderson, '06] Anderson, J. H., Calandrino, J. M. and Devi, U. C. 2006. Real-Time Scheduling on Multicore Platforms. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium, 2006. Proceedings of the 12th IEEE*, pages 179-190.

[Andrews, '10] Andrews, C., Endert, A. and North, C. 2010. Space to think: large high-resolution displays for sensemaking. In *Proceedings of the 28th international conference on Human factors in computing systems*, pages 55-64.

[Apteker, '95] Apteker, R. T., Fisher, J. A., Kisimov, V. S. and Neishlos, H. 1995. Video acceptability and frame rate. *MultiMedia, IEEE*, 2(3):32-40.

[Ball, '05] Ball, R. and North, C. 2005. Analysis of User Behavior on High-Resolution Tiled Displays. In *Human-Computer Interaction, INTERACT 2005*, pages 350-363.

[Bovet, '05] Bovet, D. P. and Cesati, M. 2005. *Understanding the Linux Kernel, Third Edition*. O'Reilly Media, Inc.

[Brown, '97] Brown, R. G. and Hwang, P. Y. C. 1997. *Introduction to Random Signals and Applied Kalman Filtering*. Wiley.

[Cedilnik, '06] Cedilnik, A., Geveci, B., Moreland, K., Ahrens, J. and Favre, J. 2006. Remote large data visualization in the paraview framework. In *Proceedings of the Eurographics Parallel Graphics and Visualization*, pages 162-170.

[Chandra, '00] Chandra, A., Adler, M., Goyal, P. and Shenoy, P. 2000. Surplus fair scheduling: a proportional-share CPU scheduling algorithm for symmetric multiprocessors. In *Proceedings of the 4th conference on Symposium on Operating System Design & Implementation - Volume 4*, pages 4-4.

[Chandra, '05] Chandra, D., Guo, F., Kim, S. and Solihin, Y. 2005. Predicting inter-thread cache contention on a chip multi-processor architecture. In *Proceedings of the High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*, pages 340-351.

[Chen, '07] Chen, J. Y. C. and Thropp, J. E. 2007. Review of Low Frame Rate Effects on Human Performance. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 37(6):1063-1076.

[Claypool, '09] Claypool, M. and Claypool, K. 2009. Perspectives, frame rates and resolutions: it's all in the game. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, pages 42-49.

[Corbet, '04] Corbet. Scheduling Domains. http://lwn.net/Articles/80911/

[Czerwinski, '06] Czerwinski, M., Robertson, G., Meyers, B., Smith, G., Robbins, D. and Tan, D. 2006. Large display research overview. In *CHI '06 extended abstracts on Human factors in computing systems*, pages 69-74.

[DeFanti, '09] DeFanti, T. A., Leigh, J., Renambot, L., Jeong, B., Verlo, A., Long, L., Brown, M., Sandin, D. J., Vishwanath, V., Liu, Q., Katz, M. J., Papadopoulos, P., Keefe, J. P., Hidley, G. R., Dawe, G. L., Kaufman, I., Glogowski, B., Doerr, K.-U., Singh, R., Girado, J., Schulze, J. P., Kuester, F. and Smarr, L. 2009. The OptIPortal, a scalable visualization, storage, and computing interface device for the OptiPuter. *Future Generation Computer Systems*, 25(2):114-123.

[Doerr, '11] Doerr, K. and Kuester, F. 2011. CGLX: A Scalable, High-Performance Visualization Framework for Networked Display Environments. *Visualization and Computer Graphics, IEEE Transactions on*, 17(3):320-332.

[Duda, '99] Duda, K. J. and Cheriton, D. R. 1999. Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler. In *Proceedings of the seventeenth ACM symposium on Operating systems principles*, pages 261-276.

[Durand, '96] Durand, D., Montaut, T., Kervella, L. and Jalby, W. 1996. Impact of memory contention on dynamic scheduling on NUMA multiprocessors. *Parallel and Distributed Systems, IEEE Transactions on*, 7(11):1201-1214.

[Eilemann, '09] Eilemann, S., Makhinya, M. and Pajarola, R. 2009. Equalizer: A Scalable Parallel Rendering Framework. *Visualization and Computer Graphics, IEEE Transactions on*, 15(3):436-452.

[Etsion, '04] Etsion, Y., Tsafrir, D. and Feitelson, D. G. 2004. Desktop scheduling: how can we know what the user wants? In *Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*, pages 110-115.

[Evans, '93] Evans, S., Clarke, K., Singleton, D. and Smaalders, B. 1993. Optimizing Unix resource scheduling for user interaction. In *Proceedings of the USENIX Summer 1993 Technical Conference on Summer technical conference - Volume 1*, pages 205-218.

[Golub, '94] Golub, D. B. Operating System Support for Coexistence of Real-Time and Conventional Scheduling. School of Computer Science, Carnegie Mellon University, 1994.

[Goyal, '01] Goyal, P., Guo, X. and Vin, H. M. 2001. A hierarchical CPU scheduler for multimedia operating systems. In *Readings in multimedia computing and networking*. Morgan Kaufmann Publishers Inc., pages 491-505.

[Gulliver, '04] Gulliver, S. R. and Ghinea, G. 2004. Changing frame rate, changing satisfaction? [multimedia quality of perception]. In *Proceedings of the IEEE International Conference on Multimedia and Expo, ICME '04,* pages 177-180.

[Haller, '10] Haller, M., Leitner, J., Seifried, T., Wallace, J. R., Scott, S. D., Richter, C., Brandl, P., Gokcezade, A. and Hunter, S. 2010. The NiCE Discussion Room: Integrating Paper and Digital Media to Support Co-Located Group Meetings. In *Proceedings of the 28th international conference on Human factors in computing systems*, pages 609-618.

[Humphreys, '02] Humphreys, G., Houston, M., Ng, R., Frank, R., Ahern, S., Kirchner, P. D. and Klosowski, J. T. 2002. Chromium: a stream-processing framework for interactive rendering on clusters. *ACM Trans. Graph.,* 21(3):693-702.

[Jagodic, '11] Jagodic, R., Renambot, L., Johnson, A., Leigh, J. and Deshpande, S. 2011. Enabling multi-user interaction in large high-resolution distributed environments. *Future Gener. Comput. Syst.,* 27(7):914-923.

[Jeong, '10] Jeong, B. 2010. Ultrascale Collaborative Visualization Using a Display-Rich Global Cyberinfrastructure. *Computer Graphics and Applications, IEEE,* 30(3):71-83.

[Jeong, '06] Jeong, B., Renambot, L., Jagodic, R., Singh, R., Aguilera, J., Johnson, A. and Leigh, J. 2006. High-Performance Dynamic Graphics Streaming for Scalable Adaptive Graphics Environment. In *Supercomputing, 2006. SC '06. Proceedings of the ACM/IEEE SC 2006 Conference,* pages 24-24.

[Jones, '97] Jones, M. B., Roşu, D. and Roşu, M.-C. 1997. CPU reservations and time constraints: efficient, predictable scheduling of independent activities. *SIGOPS Oper. Syst. Rev.*, 31(5):198-211.

[Keshav, '97] Keshav, S. 1997. *An Engineering Approach to Computer Networking*. Addison-Wesley.

[Kuan-Ta, '09] Kuan-Ta, C., Cheng-Chun, T. and Wei-Cheng, X. 2009. OneClick: A Framework for Measuring Network Quality of Experience. In *Proceedings of the INFOCOM 2009, IEEE,* pages 702-710.

[Leigh, '06] Leigh, J., Renambot, L., Johnson, A., Jeong, B., Jagodic, R., Schwarz, N., Svistula, D., Singh, R., Aguilera, J., Wang, X., Vishwanath, V., Lopez, B., Sandin, D., Peterka, T., Girado, J., Kooima, R., Ge, J., Long, L., Verlo, A., DeFanti, T. A., Brown, M., Cox, D., Patterson, R., Dorn, P., Wefel, P., Levy, S., Talandis, J., Reitzer, J., Prudhomme, T., Coffin, T., Davis, B., Wielinga, P., Stolk, B., Bum Koo, G., Kim, J., Han, S., Kim, J., Corrie, B., Zimmerman, T., Boulanger, P. and Garcia, M. 2006. The global lambda visualization facility: An international ultra-high-definition wide-area visualization collaboratory. *Future Generation Computer Systems*, 22(8):964-971.

[Liu, '73] Liu, C. L. and Layland, J. W. 1973. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. ACM*, 20(1):46-61.

[McKusick, '04] McKusick, M. K. and Neville-Neil, G. V. 2004. Thread Scheduling in FreeBSD 5.2. *Queue*, 2(7):58-64.

[Mercer, '94] Mercer, C. W., Savage, S. and Tokuda, H. 1994. Processor capacity reserves: operating system support for multimedia applications. In *Proceedings of the International Conference on Multimedia Computing and Systems*, pages 90-99.

[Motwani, '93] Motwani, R., Phillips, S. and Torng, E. 1993. Non-clairvoyant scheduling. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 422-431.

[Nam, '10] Nam, S., Deshpande, S., Vishwanath, V., Jeong, B., Renambot, L. and Leigh, J. 2010. Multi-application inter-tile synchronization on ultra-high-resolution display walls. In *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, pages 145-156.

[Nam, '09] Nam, S., Jeong, B., Renambot, L., Johnson, A., Gaither, K. and Leigh, J. 2009. Remote visualization of large scale data for ultra-high resolution display environments. In *Proceedings of the 2009 Workshop on Ultrascale Visualization*, pages 42-44.

[Nieh, '03] Nieh, J. and Lam, M. S. 2003. A SMART scheduler for multimedia applications. *ACM Trans. Comput. Syst.*, 21(2):117-163.

[Plaue, '09] Plaue, C. and Stasko, J. 2009. Presence & placement: exploring the benefits of multiple shared displays on an intellective sensemaking task. In *Proceedings of the ACM 2009 international conference on Supporting group work*, pages 179-188.

[Smarr, '09] Smarr, L., Brown, M. and de Laat, C. 2009. Special section: OptIPlanet -- The OptIPuter global collaboratory. *Future Generation Computer Systems*, 25(2):109-113.

[Smarr, '03] Smarr, L. L., Chien, A. A., DeFanti, T., Leigh, J. and Papadopoulos, P. M. 2003. The OptIPuter. *Commun. ACM*, 46(11):58-67.

[Stoica, '97] Stoica, I., Abdel-Wahab, H. and Jeffay, K. 1997. On the Duality between Resource Reservation and Proportional Share Resource Allocation. In *Proc. of Multimedia Computing and Networking*, pages 207-214.

[Stoica, '96] Stoica, I., Abdel-Wahab, H., Jeffay, K., Baruah, S. K., Gehrke, J. E. and Plaxton, C. G. 1996. A proportional share resource allocation algorithm for real-time, time-shared systems. In *Proceedings of the Real-Time Systems Symposium, 1996., 17th IEEE*, pages 288-299.

[Tan, '03] Tan, D. S., Gergle, D., Scupelli, P. and Pausch, R. 2003. With similar visual angles, larger displays improve spatial performance. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 217-224.

[Tokuda, '90] Tokuda, H., Nakajima, T. and Rao, P. 1990. Real-Time Mach: Towards Predictable Real-Time Systems. In *USENIX Mach Symposium*, pages 73-82.

[Urgaonkar, '02] Urgaonkar, B., Shenoy, P. and Roscoe, T. 2002. Resource overbooking and application profiling in shared hosting platforms. In *Proceedings of the 5th symposium on Operating systems design and implementation*, pages 239-254.

[Vishwanath, '08] Vishwanath, V., Leigh, J., Nam, S., Renambot, L., Shimizu, T., Kamatani, O., Hirokazu, T. and Takizawa, M. 2008. The Rails Toolkit - Enabling End-System Topology-Aware High End Computing. In *4th IEEE International Conference on e-Science*, pages 309-316.

[Wu, '09] Wu, W., Arefin, A., Rivas, R., Nahrstedt, K., Sheppard, R. and Yang, Z. 2009. Quality of experience in distributed interactive multimedia environments: toward a theoretical framework. In *Proceedings of the 17th ACM international conference on Multimedia*, pages 481-490.

[Yau, '97] Yau, D. K. Y. and Lam, S. S. 1997. Adaptive rate-controlled scheduling for multimedia applications. *Networking, IEEE/ACM Transactions on*, 5(4):475-488.

[Yost, '07] Yost, B., Haciahmetoglu, Y. and North, C. 2007. Beyond visual acuity: the perceptual scalability of information visualizations for large displays. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 101-110.

[Zhang, '90] Zhang, L. 1990. Virtual clock: a new traffic control algorithm for packet switching networks. In *Proceedings of the ACM symposium on Communications architectures & protocols*, pages 19-29.

[Zheng, '10]		Zheng, H. and Nieh, J. 2010. RSIO: automatic user interaction detection and scheduling. In *Proceedings of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 263-274.

[Zixia, '12]		Zixia, H., Ahsan, A., Pooja, A., Klara, N. and Wanmin, W. 2012. Towards the understanding of human perceptual quality in tele-immersive shared activity. In *Proceedings of the 3rd Multimedia Systems Conference*, pages 29-34.

APPENDIX

UNIVERSITY OF ILLINOIS
AT CHICAGO

Office for the Protection of Research Subjects (OPRS)
Office of the Vice Chancellor for Research (MC 672)
203 Administrative Office Building
1737 West Polk Street
Chicago, Illinois 60612-7227

Exemption Granted

August 7, 2012

Sungwon Nam, MS

Computer Science

851 S Morgan St. Room 1120 SEO

M/C 152

Chicago, IL 60612

Phone: (312) 996-3002 / Fax: (312) 413-7585

RE: Research Protocol # 2012-0643

"Multiuser-Aware Resource Scheduling for Large-scale Display Wall Environments"

Sponsors: None

Dear Sungwon Nam:

Your Claim of Exemption was reviewed on August 5, 2012 and it was determined that your research protocol meets the criteria for exemption as defined in the U. S. Department of Health and Human Services Regulations for the Protection of Human Subjects [(45 CFR 46.101(b)]. You may now begin your research.

<u>Exemption Period:</u> **August 5, 2012 – August 5, 2015**

Performance Site:	UIC
Subject Population:	Adult (18+ years) subjects only
Number of Subjects:	20

The specific exemption category under 45 CFR 46.101(b) is:

(2) Research involving the use of educational tests (cognitive, diagnostic, aptitude, achievement), survey procedures, interview procedures or observation of public behavior, unless: (i) information obtained is recorded in such a manner that human subjects can be identified, directly or through identifiers linked to the subjects; and (ii) any disclosure of the human subjects' responses outside the research could reasonably place the subjects at risk of criminal or civil liability or be damaging to the subjects' financial standing, employability, or reputation.

You are reminded that investigators whose research involving human subjects is determined to be exempt from the federal regulations for the protection of human subjects still have responsibilities for the ethical conduct of the research under state law and UIC policy. Please be aware of the following UIC policies and responsibilities for investigators:

1. Amendments You are responsible for reporting any amendments to your research protocol that may affect the determination of the exemption and may result in your research no longer being eligible for the exemption that has been granted.

2. Record Keeping You are responsible for maintaining a copy all research related records in a secure location in the event future verification is necessary, at a minimum these documents include: the research protocol, the claim of exemption application, all questionnaires, survey instruments, interview questions and/or data collection instruments associated with this research protocol, recruiting or advertising materials, any consent forms or information sheets given to subjects, or any other pertinent documents.

3. Final Report When you have completed work on your research protocol, you should submit a final report to the Office for Protection of Research Subjects (OPRS).

4. Information for Human Subjects UIC Policy requires investigators to provide information about the research protocol to subjects and to obtain their permission prior to their participating in the research. The information about the research protocol should be presented to subjects in writing or orally from a written script. When appropriate, the following information must be provided to all research subjects participating in exempt studies:
   a. The researchers affiliation; UIC, JBVMAC or other institutions,
   b. The purpose of the research,
   c. The extent of the subject’s involvement and an explanation of the procedures to be followed,
   d. Whether the information being collected will be used for any purposes other than the proposed research,

e. A description of the procedures to protect the privacy of subjects and the confidentiality of the research information and data,
f. Description of any reasonable foreseeable risks,
g. Description of anticipated benefit,
h. A statement that participation is voluntary and subjects can refuse to participate or can stop at any time,
i. A statement that the researcher is available to answer any questions that the subject may have and which includes the name and phone number of the investigator(s).
j. A statement that the UIC IRB/OPRS or JBVMAC Patient Advocate Office is available if there are questions about subject's rights, which includes the appropriate phone numbers.

Please be sure to:

→Use your research protocol number (listed above) on any documents or correspondence with the IRB concerning your research protocol.

We wish you the best as you conduct your research. If you have any questions or need further help, please contact me at (312) 355-2908 or the OPRS office at (312) 996-1711. Please send any correspondence about this protocol to OPRS at 203 AOB, M/C 672.

Sincerely,

Charles W. Hoehne, B.S., C.I.P.

Assistant Director, IRB # 2

Office for the Protection of Research Subjects

cc: Peter C. Nelson, Computer Science, M/C 152

Jason Leigh, Computer Science, M/C 152

VITA

EDUCATION

Ph.D., Electronic Visualization Laboratory (EVL), Computer Science Department,

University of Illinois at Chicago (UIC) [Aug, 2006 – May, 2013]

M.S. Computer Science Department,

University of Southern California (USC) [Aug, 2003 – May, 2006]

B.E. Electronic and Electric Engineering Department,

Hong-Ik University, Seoul, South Korea [Mar, 1995 – Feb, 2002]

PROFESSIONAL EXPERIENCE

Research Assistant, EVL, UIC [2008 – 2013]

Teaching Assistant, UIC [2007 – 2008]

PROFESSIONAL ACTIVITIES

Demonstrations:

- Electronic Visualization Laboratory (EVL), Chicago, IL. Assist with EVL technology demonstrations to University administration, students (Engineering Week, CS Open House, Engineering 100), and VIPs (Disney Studios, NTT Network Innovation Laboratories, etc.)
- Supercomputing 2011 (SC'11), Seattle, WA. Demonstrated SAGE's multi-user collaboration and interaction technologies and real-time high-definition (HD) image streaming on tiled-display walls.
- Supercomputing 2010 (SC'10), New Orleans, LA. Demonstrated SAGE.
- ON*VECTOR International Photonics Workshop 2009, Calit2, UCSD, San Diego, CA. Demonstrated SAGE's ability to stream uncompressed 4K (4 times the resolution of HDTV) to a tiled-display wall.

Presentations:

- Supercomputing 2011 (SC'11), Seattle, WA. Presented "Single machine driven tiled-display wall: SAGE-Next," SAGE Birds-of-a-Feather session attended by ~50 international SAGE users.
- ACM Multimedia System 2010, Scottsdale, AZ. Presented the paper "Multi-Application Inter-Tile Synchronization on Ultra-High-Resolution Display Walls".
- ACM Ultrascale Visualization Workshop, Supercomputing 2009, Portland, OR. Presented "Remote Visualization of Large Scale Data for Ultra-High Resolution Display Environments".

JOURNAL PUBLICATIONS

1. **Nam, S.**, Reda, K., Renambot, L., Johnson, A., Leigh, J., "Multiuser-Centered Resource Scheduling for Collaborative Display Wall Environments," Future Generation Computer Systems. (Under review)
2. Leigh, J., Johnson, A., Renambot, L., Peterka, T., Jeong, B., Sandin, D.J., Talandis, J., Jagodic, R., **Nam, S.**, Hur, H., Sun, Y., "Scalable Resolution Display Walls," Proceedings of the IEEE, Vol. 101, No. 1, pp. 115-129, Jan. 2013.
3. Jeong, B., Leigh, J., Johnson, A., Renambot, L., Brown, M., Jagodic, R., **Nam, S.**, Hur, H., "Ultrascale Collaborative Visualization Using Display-Rich Global Cyberinfrastructure," IEEE Computer Graphics and Applications, Vol. 30, No. 3, May/June 2010.

CONFERENCE PUBLICATIONS

4. **Nam, S.**, Deshpande, S., Vishwanath, V., Jeong, B., Renambot, L., and Leigh, J., "Multi-Application Inter-Tile Synchronization on Ultra-High-Resolution Display Walls" Multimedia Systems, Arizona, ACM, 2010.
5. Takahashi, H., Yamamoto, T., Takizawa, M., Kamatani, O., **Nam, S.**, Renambot, L., Leigh, J., Vishwanath, V., "Leveraging end-host parallelism to achieve scalable communication bandwidth utilization," Photonics Society Winter Topical Meeting Series, IEEE, 2010.
6. **Nam, S.**, Jeong, B., Renambot, L., Johnson, A., Gaither, K., and Leigh, J., "Remote Visualization of Large Scale Data for Ultra-High Resolution Display Environments," Ultrascale Visualization Workshop, Portland, Oregon, ACM/IEEE Supercomputing 2009.
7. Vishwanath, V., **Nam, S.,** Renambot, L., Leigh, J., Takahashi, H., Takizawa, M., Kobayashi, S., Kamatani, O., Ishida, "Achieving large bandwidth by leveraging parallelism in end-hosts and networks," Summer Topical Meeting, 2009. LEOSST '09. IEEE/LEOS, 2009.
8. Vishwanath, V., Leigh, J., **Nam, S.**, Renambot, L., Shimizu, T., Kamatani, O., Hirokazu, T., Takizawa, M. "The Rails Toolkit - Enabling End-System Topology-Aware High End Computing," 4th IEEE International Conference on e-Science, Indianapolis, Indiana, IEEE, 2008.

9. Tsukishima, Y., Hirano, A., Taniguchi, A., Imajuku, W., Jinno, M., Hibino, Y., Takigawa, Y., Hagimoto, K., Xi, W., Renambot, L., Jeong, B., Jagodic, R., **Nam, S.**, Leigh, J., DeFanti, T., and Verlo, A., "The First Optically-Virtual-Concatenated Lambdas over Multiple Domains in Chicago Metro Area Network Achieved Through Interworking of Network Resource Managers," 12th Optoelectronics and Communications Conference (OECC 2007) / 16th International Conference on Integrated Optics and Optical Fiber Communication (IOOC), Yokohama, Japan, July 9-13, 2007. (Best Paper Award)

SKILLS

C/C++, Object Oriented Programming, UI framework (Qt SDK), Multithreading, Message Passing Interface (MPICH2), Scientific Visualization (VTK), Scripting languages (Shell, Perl, Tcl/Tk), Revision Control (SVN/GIT), and Unix/Linux servers.

PROFESSIONAL MEMBERSHIP

[2009 – present] ACM and IEEE student member.