Future Generation Computer Systems ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Multiuser-centered resource scheduling for collaborative display wall environments

Sungwon Nam *, Khairi Reda, Luc Renambot, Andrew Johnson, Jason Leigh

Electronic Visualization Laboratory, 842 W. Taylor St., Chicago, IL 60607, United States

HIGHLIGHTS

- We present a model that prioritizes applications based on how they are presented.
- We propose a resource scheduling scheme that achieves presentation fairness.
- User study evaluates the proposed scheduler in a multiuser collaborative session.

ARTICLE INFO

Article history:
Received 30 November 2012
Received in revised form
10 March 2014
Accepted 15 August 2014
Available online xxxx

Keywords:
Scheduling
Human factors
Algorithms
Interactive systems
Distributed graphics

ABSTRACT

The popularity of large-scale, high-resolution display walls, as visualization endpoints in eScience infrastructure, is rapidly growing. These displays can be connected to distributed computing resources over high-speed network, providing effective means for researchers to visualize, interact with, and understand large volumes of datasets. Typically large display walls are built by tiling multiple physical displays together and running a tiled display wall required a cluster of computers. With the advent of advanced graphics hardware, a single computer can now drive over a dozen displays, thereby greatly reducing the cost of ownership and maintenance of a tiled display wall system. This in turn enables a broader user base to take advantage of such technologies. Since tiled display walls are also well suited to collaborative work, users tend to launch and operate multiple applications simultaneously. To ensure that applications maintain a high degree of responsiveness to the users even under heavy use loads, the display wall must now ensure that the limited system resources are prioritized to maximize interactivity rather than thread-level fair sharing or overall job-completion throughput. In this paper, we present a new resource scheduling scheme that is specifically designed to prioritize responsiveness in collaborative large display wall environments where multiple users can interact with multiple applications simultaneously. We evaluate our scheduling scheme with a user study involving groups of users interacting simultaneously on a tiled display wall with multiple applications. Results show that our scheduling framework provided a higher frame-rate for applications, which led to a significantly higher user performance (approx. 25%) in a target acquisition test when compared against traditional operating system scheduling scheme.

1. Introduction

Data-intensive eScience applications are driven by global cyberinfrastructure where visualization platforms, computing and storage resources and instruments are distributed and interconnected with high-speed network. The vast amount of data produced by these eScience applications creates a major challenge for researchers who must manage and interpret the increased scale of their work.

Visualization provides effective means in scientific discovery process. One can verify the correctness of a complex simulation model, provide more insight into the model, and present results in a way that it can be more easily understood. Large-scale, high-resolution display walls are used in scientific disciplines because they are an effective way to provide both context and details when visualizing high-resolution data. Furthermore the expansive size and exquisite resolution of the display has been conclusively shown to positively impact the scientific discovery process by allowing researchers to juxtapose multiple high-resolution visualizations simultaneously [1–8].

* Corresponding author. Tel.: +1 312 996 3002.
E-mail addresses: snam5@uic.edu, schiktol@gmail.com (S. Nam), mreda2@uic.edu (K. Reda), renambot@uic.edu (L. Renambot), ajohnson@uic.edu (A. Johnson), spiff@uic.edu (J. Leigh).

http://dx.doi.org/10.1016/j.future.2014.08.012
0167-739X/© 2014 Elsevier B.V. All rights reserved.

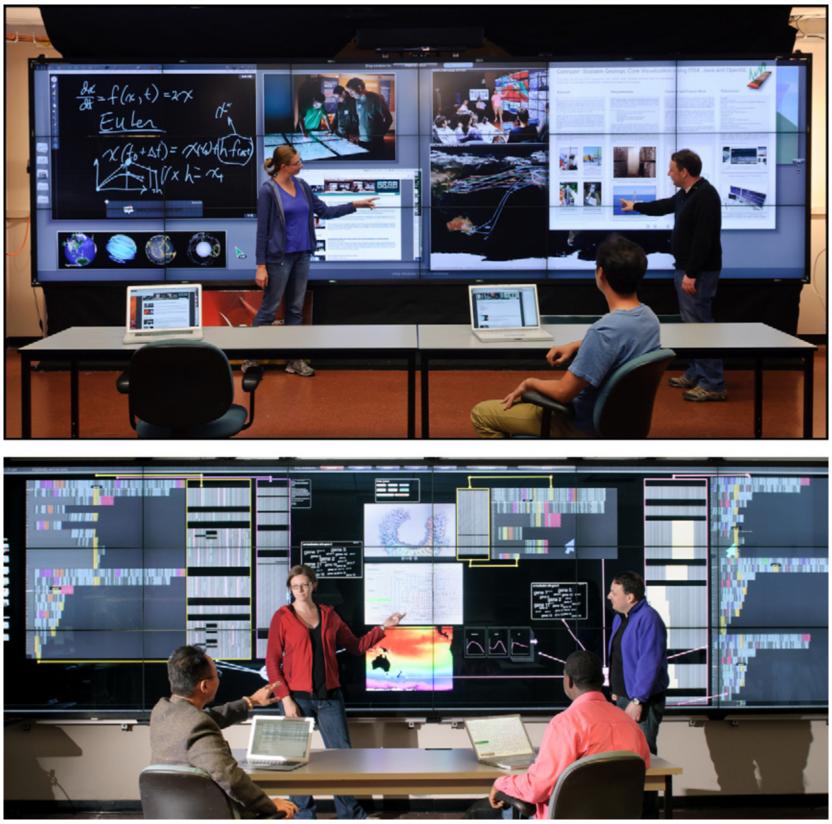

Fig. 1. A single machine-driven tiled-display wall running SAGE at the Electronic Visualization Laboratory in the University of Illinois at Chicago. The 20′ by 6′ display wall is made up of 18 LCD panels with a total resolution approximately 17 megapixels.

A common way to build large-scale display walls is to tile multiple individual displays and connect them to a cluster of computers. Thus 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 [9], Chromium [10] and Equalizer [11], that are designed as large-scale visualization platforms, can be regarded as distributed graphic frameworks. They focus 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, tiled display middleware such as the scalable adaptive graphics environment (SAGE) [12] lets users launch distributed visualization applications on remote clusters whose outputs are then streamed directly to display walls. This makes SAGE a low-cost, "thin-client" visualization endpoint where such visualizations are rendered by remote computing resources and streamed over an optical network to a display wall. SAGE also provides highly collaborative visualization environments by enabling multiple users to simultaneously view and interact with these applications on large-scale display walls [13]. An overview as well as real-world use cases of the thin-client display wall paradigm is discussed in [14,15]. In [16], we show how SAGE coupled with ParaView [17] can be used as a thin-client display for scientific visualization applications.

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. This empowers 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. Fig. 1 shows a 20×6 foot large-scale tiled display wall driven by a single computer machine at the Electronic Visualization Laboratory in the University of Illinois at Chicago.

Driving a multiuser collaborative large-scale tiled 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 collaborative large-scale display 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 tiled display wall environments.

Fig. 2 shows examples of layouts on a large-scale 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 Fig. 2, 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

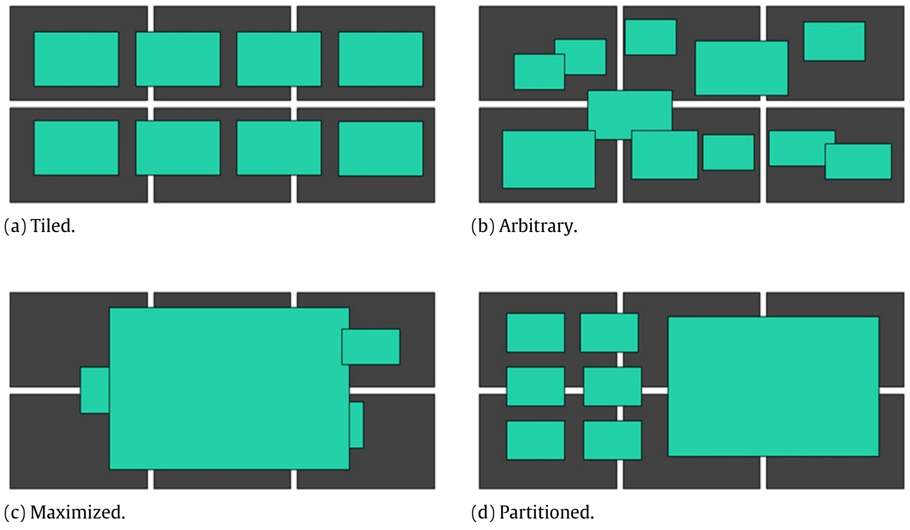
(a) Tiled. (b) Arbitrary. (c) Maximized. (d) Partitioned.

Fig. 2. 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 employ a different layout.

is appropriate for applications in the left section of the display, while the right section requires a scheduling scheme similar to (c). We identify two issues when a general-purpose operating system scheduler is employed in display wall environments.

1. Since fine-grained fairness in a general-purpose scheduler does not typically consider resource requirement of an application, it will fail to provide fairness in terms of the quality experienced by users.
2. 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.

Given the variety of layouts in collaborative multiuser tiled 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 windows that occupy the largest space on the wall, to windows 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 collaborative, multiuser setting. The goal of our scheduling framework is to fairly distribute system resources to applications to optimize their performances as experienced by users. We call this *presentation fairness*.

In this paper, we present a multiuser resource scheduling framework targeting thin-client display wall where contents are streamed to the display from high-performance computers over high-speed networks. To evaluate the effectiveness of our scheduling framework, we conduct a user study where multiple users in groups of three subjects interacted simultaneously with applications on the display wall with and without the proposed scheduling scheme. The rest of this paper is divided as follows. In Section 2 we survey the literature on resource scheduling schemes for real-time and interactive applications. The overall design of the scheduling framework is explained in Section 3. Section 4 describes the Priority model in detail. In Section 5 we illustrate the details of our resource distribution scheme. In Section 6 we describe the user study and present our findings. We conclude the paper in Section 7 and give future research directions in Section 8.

2. Related work

2.1. General-purpose schedulers

The modern general-purpose operating system schedulers are designed to achieve high job completion throughput, interactivity, and fair sharing. For instance, 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. The scheduler ensures no idle process occupies a processor wasting the computing resource by preempting processes that are waiting for resources or user inputs. The preemptive, time-sharing model also enables fair sharing of resources by maintaining a counter that represents a priority of a process. The scheduler decreases the counter for a process while the process occupies a processor and increases it while the process is waiting. The scheduler may schedule processes in a manner that it can keep the counter values as uniform as possible to ensure the fair utilization of processors. Fast response time for interactive processes is achieved by using the counter as well. For example, a process waiting for I/O devices increases its counter so that the process can a have higher counter value by the time when it is ready to run again. This can make the processes with smaller counter values (such as a batch process that consumed lots of processor time while an interactive process was waiting for I/O devices) be preempted by the scheduler. The details of modern Linux scheduler are well explained in [18,19]. Similarly, the process scheduler in modern Microsoft Windows operating systems maintains the dynamic priority for threads to ensure the fairness and the high interactivity for latency-sensitive tasks. For example, the scheduler raises the priority of a process that is in foreground, receiving user inputs, or ready to run again after waiting for I/O devices [20].

Aforementioned modern time-sharing, general-purpose operating system schedulers are characterized as a non-clairvoyant scheduling where the scheduler does not rely on processes' characteristics [21]. This is because the fast turnaround time (the total time taken by a scheduler to finish a job) is an important factor for high job completion throughput in the time-sharing systems. While the non-clairvoyant scheduling scheme for typical desktop environments excels at achieving high system-wide performance, it lacks the ability to discover what users are interested in and to schedule jobs in a way that the performance of the system can be perceived to be responsive and fair in multiuser collaborative environments due to the scheduling decisions that are made based on processes' behavior rather than users' interactions with the applications run in the system.

2.2. Human-centered schedulers

Human-centered scheduling schemes focus on optimizing user-perceived interactivity rather than system-wide performance measures [22]. The Interactive Scheduling scheme identifies interactive processes by monitoring input devices through the X Server. Etsion et al. take a similar approach by focusing on improving user interactivity [23]. 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. Zheng and Nieh present 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 [24]. 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. Therefore, these scheduling schemes cannot be applied directly to large-scale collaborative display wall environments, which introduce user-interaction patterns that are different from desktop computer systems mainly due to that there can be multiple users interacting simultaneously in multiuser collaborative display wall environments.

2.3. Real-time schedulers

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 multiuser tiled display environments, resources allocation can be biased towards applications that are presumed to be receiving most of users' attention, thus maximizing the perceived performance of the system. This section briefly surveys work on various real-time scheduling techniques.

2.3.1. Rate-monotonic and earliest deadline first

Liu and Layland [25] showed that their rate-monotonic algorithm can meet all periodic tasks deadline, bounded on processor utilization from 69% to nearly 100%. In the 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. The property of the EDF algorithm is important in real-time systems in that it is the optimum scheduling algorithm when the priorities are fixed. 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. However, the principle of rate-monotonic and EDF scheduling is employed in many real-time scheduling schemes. The fundamental concept of EDF scheduling can be adapted to our approach when several real-time applications on the display wall compete for resources.

2.3.2. 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 [26]. Real-Time Mach adopts the resource reservation technique in its scheduler to support real-time applications [27]. Less restrictive forms of the reservation scheme where a thread is allowed to negotiate CPU-time based on its rate progression is introduced in [28]. Jones et al. present a system with a CPU scheduling algorithm that ensures minimum guaranteed execution rates of real-time processes [29]. 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 has been well studied in [30]. This restrictive approach can be applied to a set of high priority computation-intensive, and time-critical applications in the display wall environments.

2.3.3. 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 apply different scheduling policies on processor sets. Golub presents an improved scheduling paradigm over Real-Time Mach with emphasis on supporting a combination of time-critical and conventional applications [31]. 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 [32]. Recently, real-time scheduling on multicore platforms focusing on grouping processes in a way that increases cache utilization was presented in [33]. The Gang Scheduling scheme can be useful for a task that needs multiple application instances running on the display wall. A group of applications can be treated as a single schedulable entity for a task to provide better seamlessness.

2.3.4. 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 [34] is introduced in [35]. Stoica et al. also showed proportional sharing combined with resource reservation scheme in [36]. A virtual time algorithm that focuses on meeting real-time requirements while achieving proportional fairness is shown in [37]. Nieh and Lam also present a similar scheduling algorithm in detail [38]. Chandra et al. present how to readjust the weights in their proportional sharing algorithm in a multiprocessor environment [39]. Our scheduler applies a similar notion of proportional sharing in which applications' weights are determined based on a priority assessment model that predicts users' interest in applications.

While real-time schedulers have been used in interactive multimedia systems, they have not been tested in large-scale display environments. Moreover, large-scale displays offer unique capabilities 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.

3. Overview

This section describes the overall design of our scheduling framework. First we discuss general requirements for fair scheduling in thin-client multiuser display walls. Then we present a design overview of our proposed framework.

3.1. Applications in thin-client display walls

In thin-client display wall environments such as SAGE, the content-generating applications (*senders*) typically run on 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 our discussion, applications (*receivers*) refer to processes running in a display wall whose contents are being streamed over the high-speed network. The proposed scheduling framework considers only receiving applications (receivers) that run in the display wall system. Furthermore, we simplify the receivers by assuming uncompressed contents stream from senders in this article.

We can further distinguish types of applications (receivers) in thin-client display wall environments. In one case, the content-generating application (the sender) has an optimal streaming rate that is known *a priori*. For example, an application streaming a live video feed from a camera or a media player streaming 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 assuming their streams are uncompressed. The resource requirement for 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 runs 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 in this case can be derived by the receiver application's current resource utilization, 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. We define the amount of resources for the optimal performance to be a time variable.

The goal of our proposed scheduling framework is to fairly distribute system resources to receiver applications to optimize their presentation quality as perceived by users rather than ensuring fine-grained thread-level fairness from a system point of view. We call this *presentation fairness*. The perceptual 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 [40–42]. However, in our discussion where content-generating applications run at remote locations and stream their images to receiver applications run on display walls, we define the quality of a receiver application as the ratio of the amount of resources being consumed by the application to the amount of resources it needs for optimal performance. Also we assume content-generating applications run in a best-effort manner and their resource usage is governed by systems they reside. For interactive applications where the amount of resource needed for optimal performance is not known a priori, we use the application's current resource utilization as the basis to estimate the amount needed for optimal performance. Section 5.2 explains in detail how we estimate this. We also prioritize applications by estimating users interest in the applications. This is discussed in detail in Section 4. We then apply a weighted max–min fair sharing algorithm with these two variables in order to achieve presentation fairness.

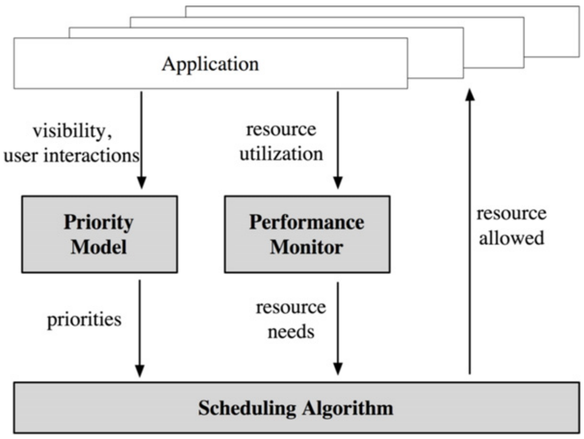

Fig. 3. Components of the proposed scheduling framework. Applications' information such as their relative visibility, frequency of user interactions, and resource utilizations are collected and processed by the Priority model and the Performance Monitor. The proportional sharing scheduling algorithm then determines the amount of resources allowed for each application.

3.2. Design

The proposed scheduling framework consists of a Priority model, a Performance Monitor, and a scheduling algorithm. Fig. 3 depicts the relationship among the components. The Priority model collects information about application states such as the application's window geometry and frequency of user interactions in order to assign a *priority* that reflects the relative importance of 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 uses the assigned priority as well as the application's resource need to allocate resources.

4. Priority model

The Priority model describes the degree of users' interests in applications running on the 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 to get a measure of users interests in 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 regional importance of specific areas of the display wall. These three factors are combined together to produce the priority value. We discuss each of the three factors and quantify them below.

4.1. Visual factors

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. We denote the *Effective Visible Size* of an application i at time t with $EVS(i, t)$. The value

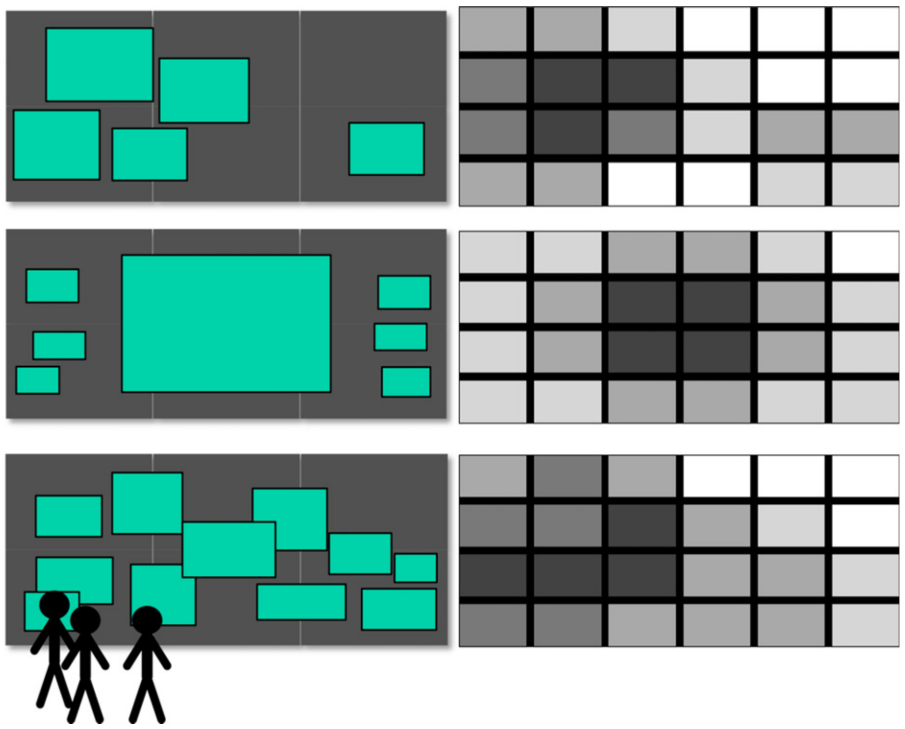

Fig. 4. Examples of display wall usage and the corresponding black-hot heat map of the wall. Darker cell indicates higher aggregate priority values. In the top figure, applications are mostly positioned on the left portion of the wall. In the middle figure, one application is maximized on the center of the wall. In the bottom figure, the applications are arbitrarily scattered, but the users are mostly interacting with the applications on the left portion of the display.

of *EVS* can range from 0 to a maximum of *i*'s window size. For instance, $EVS(i, t) = 0$, 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 we also consider how much an application reveals its contents.

The *Exposure Ratio E* is the ratio of *EVS* to the application's window size. Thus *E* tells how much an application reveals its content on the wall and it can be denoted by $E = EVS/\text{WindowSize}$. We multiply the *EVS* by the exposure ratio *E* to obtain the priority determined by the visual factor P_{visual}. Thus $P_{visual} = E \cdot EVS = EVS^2/\text{WindowSize}$.

4.2. Interaction factor

The EVS alone might not be enough to reflect user intentions. Assume two applications *i* and *j* at time *t*, where $P_{visual}(i, t) \gg P_{visual}(j, t)$. A user could be interacting more frequently with application *j* while *i*'s priority is higher because it has a larger P_{visual} value. User interactions through input devices such as mouse, gyro mouse, touch, gesture, or keyboard indicate user's interest in an application directly.

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 schemes. So, we provide a simplified mechanism to add the interaction factor in the Priority model. Currently, we let 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. We then periodically monitor the rate of changes in the number of interactions during a single scheduling interval. 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 $(t - 1, t]$.

4.3. Wall usage pattern

Spatial layout and window arrangement patterns are likely to emerge if the display wall is used long enough. For example, in Fig. 2(d) where the wall is partitioned into two sections, the left section employs tiled-layout which is appropriate for comparisons while the right section shows a single large application window. Also imagine a case where users take turns to present their data on the display wall during a meeting. In this case, the wall usage pattern is likely to be one of the layouts shown in Fig. 2(c) and (d). When a user can expect static application window layout patterns on the wall, the user might want to save the priority values (maybe in a separate file) in order to apply them immediately for a similar use case later. This can help the scheduler to know the region of the wall beforehand that needs to be focused more.

To get an insight into these patterns, we can 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 Fig. 4, most of the applications are positioned on the left portions of the wall, causing an increase in the temperature of the left portion of the wall. In the bottom case of Fig. 4, 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.

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 a scheduling event 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

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

where L is a set of applications whose window overlaps with the cell c and $Temp(c, t-1)$ is the temperature value at the scheduling event at time $t-1$ that immediately precedes the scheduling event at time t. 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. We denote this as

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

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

4.4. The priority function

We obtain the priority 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 denote 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. How do we 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. The P_{temp} will be useful when there is a distinct wall usage pattern after long period of display wall usage. While an application developer can determine the application-specific weight factors or provide a user-interface for users to adjust the weights in real-time, the inequality of each weight factors in general can be expressed as $W_i > W_v > W_t$ based on the degree of straightforwardness of each priority component in reflecting users' interest.

5. Resource distribution

Once the Priority model assigns priorities to applications, the scheduler distributes system resources among application by adjusting their presentation quality to ensure presentation fairness. In our discussion, 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. We denote the actual amount of resources an application i consumes at time t as $R_{cur}(i, t)$ and the amount of resources i needs for the optimal performance at time t as $R_{opt}(i, t)$. The presentation quality that an application i currently achieves at time t is denoted as

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

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. We discuss how to derive the amount of resources needed for optimal performance at a given time in the latter case in Section 5.2. 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 scheduler also needs a system-wide variable indicating the total amount of available resources so that these resources can be distributed and allocated to applications at every scheduling instance. We denote the amount of total available resources seen by the scheduler as R_{TOTAL}. We describe how we obtain this amount in Section 5.3.

5.1. The demanded quality for presentation fairness

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 ($R_{sched} \le R_{opt}$). A value of 0 indicates that no resources are to be allocated for the application, which implies that the application should idle. We define 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, $Q_{sched}(i, t) = 1$ indicates that the scheduler sets no limit on resource consumption for application i, until $Q_{sched}(i, t + l)$ is set to a value less than 1. In this case, the amount of resources consumed by an application (R_{cur}) can be greater than the amount demanded (R_{sched}) and the amount derived for optimal performance (R_{opt}) during the time period l under this special condition. This special condition is needed because the scheduler cannot know the global maximum of R_{opt} of an interactive application. The scheduler lets the application (that has its Q_{sched} was set to 1) consume as much resources as it wants regardless of its current R_{sched} in order to know the maximum R_{opt} of the application. We discuss this special condition in detail in Section 5.2.

5.2. Estimating the time varying optimal amount

The optimal amount of resources for an application (expressed by R_{opt}) is the amount of resources the application needs to achieve optimal presentation performance. We define it as a function of time. For non-interactive applications, the optimal amount of resources is known a priori, thus R_{opt} is a fixed constant at any given time assuming uncompressed stream as we discussed in Section 3. For instance, a live video feed where a user at the source (sending side) can set a desired frame rate for the feed can have a fixed desired 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

```
L is the set of all applications in the system
$R_{opt}\_multiplier \leftarrow M$ // a system-wide constant
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:         $R_{opt} \leftarrow C$                  // initial estimation
 6:     else if $R_{cur} = R_{opt}$ then
 7:         $R_{opt} \leftarrow R_{opt}\_multiplier \cdot R_{cur}$    // could use more
 8:     else   // $R_{cur}<R_{opt}$ or $R_{cur}>R_{opt}$ because of $Q_{sched} = 1$
 9:         $R_{opt} \leftarrow R_{cur}$
```

Fig. 5. 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.

scheduler does not have to allocate resources for the application that receives the rendered images in the display wall 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 the 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.

Fig. 5 illustrates pseudo-code that estimates and updates the optimal amount (R_{opt}) for interactive applications where their resource consumption (R_{cur}) varies 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 that the amount of resources the scheduler allows (R_{sched}) is equal to the optimal amount (R_{opt}) which means the demanded quality was set to 1 (Eq. (2)). This state implies that the application might be able to consume more resources as long as there are enough resources in the system. Recall that $Q_{sched} = 1$ 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 performs 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 Fig. 5. 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 1? 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). Then the scheduler will have an underestimated R_{opt} for the application. Since the scheduler allocates resources based on R_{opt} of the application, this can lead the scheduler to allocate less resources than the application can actually utilize even though enough resources become available later. The Performance Monitor increases the application's resource need ($R_{opt} \leftarrow R_{opt\_multiplier} \cdot R_{cur}$) more than the R_{cur} of the application to prevent a situation where R_{opt} is bounded to a local maximum when the system is overloaded. The underestimated R_{opt} is corrected in this way. In our experiment, the $R_{opt\_multiplier}$ was set to 1.1 (10% more). 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 9 in Fig. 5) 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 (Eqs. (1) and (2)) 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, our evaluation (Section 6) shows that the improved user experience in typical use cases outweighs the waste.

5.3. 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, we define the amount of total available resources in the system seen by the scheduler 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.

$$R_{TOTAL}(t) = \Sigma_{i \in L} R_{cur}(i, t). \qquad (3)$$

However, Eq. (3) alone is not 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. Fig. 6 depicts how the R_{TOTAL} is updated. R_{TOTAL} starts with 0 and increased whenever a new application is 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.

5.4. Proportional sharing algorithm

We apply a weighted max–min fair-sharing algorithm to achieve presentation fairness. The 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.

The algorithm is illustrated with a partial pseudo-code in Fig. 7. The *array_*R_{sched} is an empty array that will hold the values of R_{sched} shown in Eq. (2) for each application. Each application i adjusts its resource consumption to comply with the scheduler's demand.

```
L is a set of all applications 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
```

Fig. 6. 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.

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_Qd function. The fraction that determines a granularity of the resource allocation (which affects the time complexity of the algorithm) is initialized with a constant value $F(0 < F \leq 1)$.

$$unitAllocAmnt[i] = fraction \cdot R_{opt}(i, t) \cdot (P(i, t)/\Sigma P(i, t)) \qquad (4)$$

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. Eq. (4) conveys the main idea of the scheduling algorithm. A small amount of resources proportional to a priority value of an application is allocated progressively. The temporary priority sum (*sum_tmp*) needs to be reset (line 10) before the inner loop (line 11) because some applications can be excluded during the iterations as in the conditional block (line 12–14). 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 (Eq. (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 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 algorithm starts with obtaining the current R_{opt} and the R_{TOTAL}. Then R_{sched} for each application is calculated progressively in the inner loop (line 11) until no more resources can be allocated to any application. This condition can arise when either all the available resources are allocated (line 7) or all the application is allocated with the amount equal to their R_{opt} (line 8).

To find the worst case running time of the algorithm, let the R_{TOTAL} be infinite and for all application, let $R_{cur} > 0$. Then the algorithm will terminate only after all the applications are allocated with the amounts that they require (when $R_{sched} = R_{opt}$). Therefore the maximum number of iterations of the inner loop (line 11) will be determined by the lowest priority application (because the list L is ordered by the priority and an application always receives a small portion of the amount it wants) and the value of the fraction F as in Eq. (4). The maximum number of iterations (denoted as *numit* in Eq. (5)) of the outer loop (line 7) is then determined by the number of allocation for the least important application A ($R_{opt}(A)/unitAllocAmnt[A]$). Using Eq. (4), we obtain

$$numit = fraction \cdot (P(A)/\Sigma P(A)) \qquad (5)$$

where A is the application with the lowest priority.

The ESTIMATE_Ropt(), GET_R_{TOTAL}(), and the inner loop of the algorithm (line 11) take $O(n)$ time where n is the number of applications in the system. Therefore, the worst case running time of the algorithm is $O(numit\ n)$ using Eq. (5).

```
L is an ordered set of all applications (ordered by the priority)
array_$R_{sched}$ ← [ ]
unitAllocAmount ← [ ]
fraction ← F        // $0 < F \leq 1$

COMPUTE_Qd( )
 1: t ← NOW
 2: sum ← $\Sigma_{i \in L} P(i, t)$
 3: for each i in L
 4:     array_$R_{sched}$[i] ← 0  // init the array
 5: ESTIMATE_$R_{opt}$(t)
 6: rt ← GET_$R_{TOTAL}$(t)

 7: while (rt > 0)
 8:     if for each i in L  array_$R_{sched}$[i] = $R_{opt}(i, t)$ then
 9:         break
10:     sum_tmp = sum // reset sum_tmp in each iteration
11:     for each i in L   // allocate a fraction of what it needs
            // exclude one that does not more need resources
12:         if $R_{cur}(i,t)$ = 0 or array_$R_{sched}$[i] = $R_{opt}(i, t)$ then
13:             L ← L – {i}
14:             sum_tmp ← sum_tmp – P(i,t)
            // allocate small fraction for finer granularity
15:         unitAllocAmnt[i] ← fraction·$R_{opt}(i,t)$·P(i,t) / sum_tmp
16:         array_$R_{sched}$[i] ← array_$R_{sched}$[i] + unitAllocAmnt[i]
17:         rt ← rt - unitAllocAmnt[i]

18: for each i in L            // finally set the demanded quality
19:     $Q_{sched}(i, t)$ ← array_$R_{sched}$[i] / $R_{opt}(i, t)$
```

Fig. 7. Partial pseudo-code of our proportional sharing 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.

6. Evaluation

In this section, we describe two experiments to demonstrate that our scheduler achieves presentation fairness on a tiled display with non-interactive as well as interactive applications, and with multiple users interacting with the system simultaneously. The tiled-display wall system employed in the experiments consists of 18 LCD displays, as shown in Fig. 1, with a total resolution of approximately 17 megapixels. A single machine equipped with dual Intel X5650 quad core processors, 12 GB of main memory, and 3 Nvidia GeForce GTX580 dual DVI graphics was used to drive the entire display. We use a separate, equally powerful machine to run *sender* applications that stream images to the display wall system over network to simulate a thin-client display environment. Thus there exist a *receiver* application for each sender application.

Both machines run 64bit OpenSUSE 12.1 (Linux kernel 3.1). The display wall middleware we ran in our experiments mainly consists of receiver applications that receive streams of images from sender applications, the proposed scheduler that manages resources for the receiver applications, and graphics part that provides a large screen surface on the display wall. While each sender is an individual process in the sending machine, the receiver runs in a thread within the display wall middleware in the display wall system. When the proposed scheduling scheme is enabled, the receivers' resource usages are monitored and scheduled as we explained in previous sections. This is denoted as *Sched* throughout the remainder of the paper. And we denote the condition where the proposed scheduling algorithm was not running as *NoSched*. Our display wall middleware runs as a user-level application. Thus the operating system's scheduler was running in both *Sched* and *NoSched* conditions. The two machines are network connected with a 10 Gbps optical switch. We use the bit-rate of an application as a metric for the amount of resources. Thus the amount of

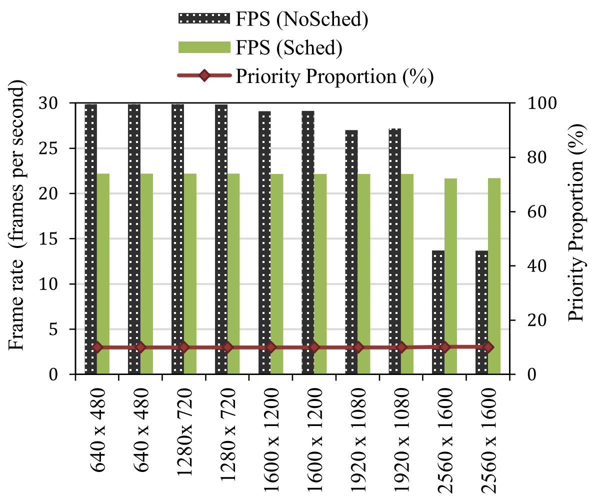

Fig. 8. Achieved frame rate for applications used in the experiment under the operating system's scheduler (NoSched), and with our scheduler running (Sched). The *X* axis lists frame sizes of the 10 applications which are run simultaneously in the experiment. Application windows are arranged in a tiled mode as illustrated in Fig. 2(a) giving them equal priority. The flat line shows indeed that the scheduler assigns equal priority proportions to all applications.

resources an application consumes (R_{cur}) indicates its image streaming bandwidth in bits-per-second.

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

6.1. *Presentation fairness in non-interactive applications*

In the first experiment, we evaluate the effectiveness of our scheduler with non-interactive applications. We developed a simple non-interactive sender application that streams a fixed size buffer (in memory) at a fixed rate to the corresponding receiver application in the display wall over network. Ten sender applications with different image sizes (shown on the *X* axis in Fig. 8) are run simultaneously, streaming their contents to the display wall at 30 frames per second. Thus the bandwidth needed by each corresponding receiver application for optimal performance (R_{opt}) is their image frame size in bits multiplied by 30 Hz. Although the image frame sizes of the sender applications are fixed, their window sizes on the display wall, which determine their P_{visual}, can be arbitrary.

To demonstrate presentation fairness, we arrange the windows in a tiled mode as illustrated in Fig. 2(a), thus giving 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 Fig. 8 shows that the scheduler indeed assigns equal priority proportions to all applications. When our scheduler is not running (*NoSched* condition), the operating system's scheduler distributes resources evenly among applications. Since applications have varying frame sizes, this fine-grained distribution of resources leads to diverging performance as evident in Fig. 8. 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 our scheduler is running (*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.

6.2. *Evaluating responsiveness in interactive applications*

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. We simulate this scenario in a user study in order to evaluate the effectiveness of our scheduler with interactive applications and multiuser interaction.

We developed a simple sender and an interactive receiver application where the sender streams a fixed size buffer (in memory) to the receiver. The sender application is called a *streamer* and the corresponding receiver application is called a *receiver*. We also refer the receiver as user applications throughout the remainder of the paper. 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. 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. 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 30 Hz ($R_{opt} = 2949.12$ Mbps). We refer to these applications as the *overhead*.

6.2.1. *Task*

The user study comprised single and multiuser interactions 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. 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 as quickly as possible. However, a target can be acquired and a next target appears only when the subject successfully clicks the target. Thus subjects were implicitly asked to be as precise as possible at the same time. 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 influence subject performance. The rationale behind this task is that performance in the target acquisition task will demonstrate the responsiveness of the system. This will in turn

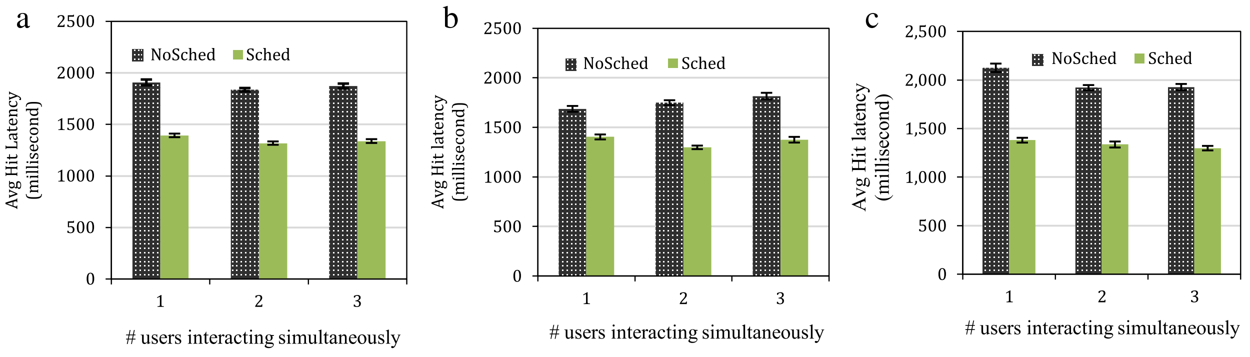

Fig. 9. (a) The average hit latency for all groups with and without the proposed scheduling scheme. Error bars reflect the standard errors. The hit latency was reduced significantly with the proposed scheduling scheme. (b) The average hit latency for groups A, B, and C where the experiment began with the *Sched* condition. (c) The average hit latency for groups D, E, and F where the experiment began with the *NoSched* condition.

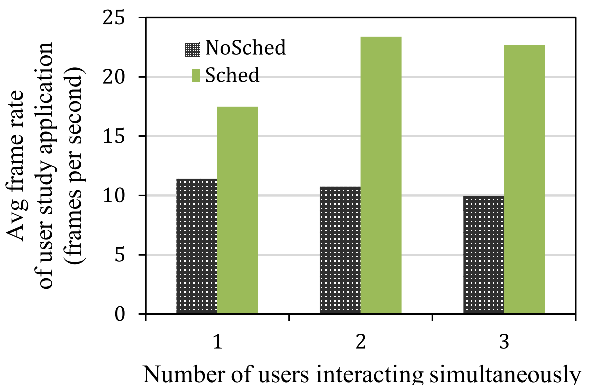

Fig. 10. The average frame rate achieved by user applications. The frame rates slightly dropped as the number of users increased in the *NoSched* condition, where as the frame rates remained high with our scheduling scheme ensuring better user interactivity.

reflect objective performance (target acquisition speed and precision) as well as subjective user experience in more complex scenarios such as scientific visualization and interactive, multimedia applications.

6.2.2. Procedure

We recruited 18 subjects. All subjects were computer science students (both graduate and undergraduate). Subjects are divided into groups of three, with a total of 6 groups, which we refer 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).

6.2.3. Results

For each subject, we measure the hit latency (the time it takes for 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). We first illustrate the effect of our scheduler on subject performance. We then show our scheduling scheme total resource utilization.

We compute a 2 (factor #1: *NoSched* versus *Sched* conditions) x 3 (factor #2: 1, 2, and 3 users interacting simultaneously) factorial ANOVA to see the differences between average hit latencies under various conditions. The result indicates that there are significant main effects (factor #1: $F(1, 3234) = 927$, $p < 0.000$, factor #2: $F(2, 3234) = 6.12$, $p = 0.002$) but no interaction effect ($F(2, 3234) = 0.13$, $p = 0.88$). Fig. 9(a) shows the average hit latency of all groups with and without our scheduling scheme. With our scheduling scheme, the average hit latency of all subject groups is reduced by approximately 28%. This increased user performance is due to the higher frame rate achieved with our scheduling scheme as shown in Fig. 10. The user applications achieve minimum of approximately 17.5 Hz with our scheduling scheme. This is higher than a frame rate threshold (10–15 Hz) where human performance can be adversely affected [43–46]. Even with a high workload and increasing number of users interacting simultaneously, our scheduler was able to maintain a sufficient frame rate, which helped subjects maintain their task performance as shown in Fig. 9(a)–(c).

The frame rates under *Sched* condition in Fig. 10 are not linear as *NoSched* condition. This is due to multiuser interactions. We observed that subjects tend to compete with others especially when multiple subjects were interacting simultaneously. The reason that the frame rates under NoSched condition in Fig. 10 shows linearity is because of the operating system scheduler's fair scheduling policy where it tries to distribute resources evenly among applications.

Under the *NoSched* condition, subjects' performance decreases as the number of users interacting simultaneously increases as depicted in Fig. 9(b). On the other hand, subjects' performance improved slightly as the number of users interacting simultaneously increases as depicted in Fig. 9(c). We believe this trend is caused by an effect where subjects gradually adapt to performing the task at low frame-rates.

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 Fig. 11(a)). However, the difference in the average number of missed clicks is insignificant as shown in Fig. 11(b). During the user study, we 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 our scheduler is employed or not.

Fig. 12 depicts the total resource utilization breakdown. The graph shows that fewer resources are allocated to the overhead, allocating more to user applications under our scheduling scheme as the design dictates, which ultimately led to the improved user

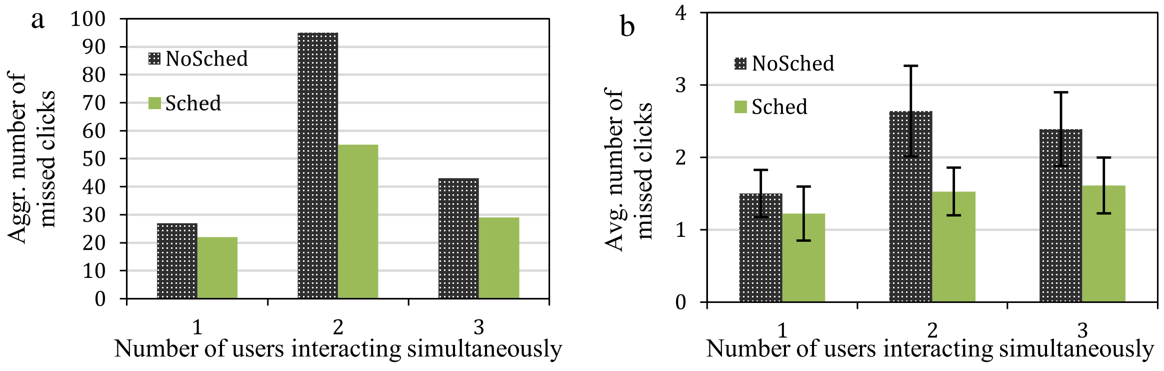

Fig. 11. (a) The aggregate number of all subject groups' that missed clicks with and without the proposed scheduling scheme. (b) The average number of missed clicks with and without the proposed scheduling scheme shown with standard errors.

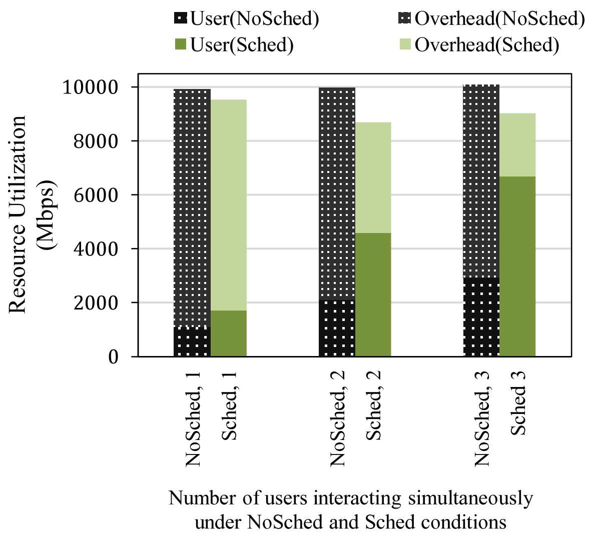

Fig. 12. The average of total resource utilization breakdown with and without our scheduling scheme. The graph indicates that more resources are utilized by user applications under the proposed scheduling scheme.

performance. As expected, the overall resource utilization of the proposed 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 the 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 more precise (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) better when the rate of changes in user interactions is steady, thereby increasing resource utilization.

7. Conclusion

Thin-client display wall systems are used for displaying high-resolution visual contents that are rendered at remote resources (such as high-performance computers and media providers) and providing multiuser interactions on them. In this paper, we presented a novel resource scheduling scheme for these highly collaborative display wall environments. Unlike traditional resource scheduling in modern operating systems, our multiuser resource scheduling scheme adopts a user-centered scheduling to maximize user-perceived performance, favoring applications that are most likely to draw user attention.

Our scheduling framework ensures the presentation fairness. 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 not known a priori, the scheduler estimates these amounts based on current application performance. The scheduler then finds a fair distribution of system resources and adjusts resource consumption indirectly by modulating presentation qualities of applications.

Experimental results show that our resource scheduler achieves presentation fairness in non-interactive applications where resource needs are known a priori. We also conducted a user study to evaluate the effect of our scheduler on user performance with interactive applications running on an overloaded tiled display wall. The user study shows the subject performance in a target acquisition task improved with the proposed scheduling scheme over general-purpose operating system scheduling. This demonstrates the effectiveness of our scheduling scheme when employed in interactive tiled display walls that are used in collaborative settings where multiple users interact simultaneously with the system.

8. Future work

In the future we plan to extend our Priority model. Currently, the application 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 states. 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. Also in our Priority model, we simplified the mechanism to obtain the interaction factor by having the interaction counter that needs to be implemented by an application developer. This can be prone to programming error and can have an adverse effect on a display wall. In the future we plan to move this part into the display wall's application programming interface.

Our scheduling scheme maintains the notion of an optimal amount of resource 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. The resource can be wasted when the amount is overestimated. Thus our 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. This is a hard problem because the system cannot precisely predict what a user will do. However, we plan to do further research on estimating the optimal amount of resource of interactive applications based on knowledge that can be learned from online profiling, user interaction pattern, or machine learning techniques.

The proposed scheduling scheme assumes a simple stream receiving application where an advanced video decompression technology is not required at the receiving end (at the display wall). This allowed us to assume a fixed required bandwidth that can be simply calculated a priori for an application as we mentioned in Section 3. This simplification was based on the advent of high-speed optical networking which is one of the key components of the cyberinfrastructure. However, modern video compression/decompression technologies are also an important factor that enables users to stream high-resolution videos. The scheduling scheme can be improved by supporting a receiver application that employs modern decompression technologies. This requires further study on how to reliably estimate the optimal amount of resources for the receiver application that keeps changing its resource utilization (depending on the decompressing complexity of current data) even without user interactions.

Also in our experiments, we implemented the receiver application that has a single thread for receiving images that are streamed from the sender over the network. We plan to improve our scheduling scheme to support multiple concurrent threads for high-performance applications such as a scientific visualization that can render and stream images in parallel.

Currently, our work is focused on network streaming resources but a more complex receiver application might need to utilize different types of resources. Handling different types of resources will make the proposed scheduling scheme more general and enables the scheduler to support wider range of application.

References

[1] R. Ball, C. North, Analysis of user behavior on high-resolution tiled displays, human–computer interaction, in: INTERACT 2005, 2005, pp. 350–363.
[2] B. Yost, Y. Haciahmetoglu, C. North, Beyond visual acuity: the perceptual scalability of information visualizations for large displays, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2007, pp. 101–110.
[3] J. Leigh, L. Renambot, A. Johnson, B. Jeong, R. Jagodic, N. Schwarz, D. Svistula, R. Singh, J. Aguilera, X. Wang, V. Vishwanath, B. Lopez, D. Sandin, T. Peterka, J. Girado, R. Kooima, J. Ge, L. Long, A. Verlo, T.A. DeFanti, M. Brown, D. Cox, R. Patterson, P. Dorn, P. Wefel, S. Levy, J. Talandis, J. Reitzer, T. Prudhomme, T. Coffin, B. Davis, P. Wielinga, B. Stolk, G. Bum Koo, J. Kim, S. Han, J. Kim, B. Corrie, T. Zimmerman, P. Boulanger, M. Garcia, The global lambda visualization facility: an international ultra-high-definition wide-area visualization collaboratory, Future Gener. Comput. Syst. 22 (2006) 964–971.
[4] M. Czerwinski, G. Robertson, B. Meyers, G. Smith, D. Robbins, D. Tan, Large display research overview, in: CHI'06 Extended Abstracts on Human Factors in Computing Systems, 2006, pp. 69–74.
[5] D.S. Tan, D. Gergle, P. Scupelli, R. Pausch, With similar visual angles, larger displays improve spatial performance, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2003, pp. 217–224.
[6] M. Haller, J. Leitner, T. Seifried, J.R. Wallace, S.D. Scott, C. Richter, P. Brandl, A. Gokcezade, S. Hunter, 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, 2010, pp. 609–618.
[7] C. Andrews, A. Endert, C. North, Space to think: large high-resolution displays for sensemaking, in: Proceedings of the 28th International Conference on Human Factors in Computing Systems, 2010, pp. 55–64.
[8] C. Plaue, J. Stasko, 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, 2009, pp. 179–188.
[9] K. Doerr, F. Kuester, CGLX: a scalable, high-performance visualization framework for networked display environments, IEEE Trans. Vis. Comput. Graphics 17 (2011) 320–332.
[10] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P.D. Kirchner, J.T. Klosowski, Chromium: a stream-processing framework for interactive rendering on clusters, ACM Trans. Graph. 21 (2002) 693–702.
[11] S. Eilemann, M. Makhinya, R. Pajarola, Equalizer: a scalable parallel rendering framework, IEEE Trans. Vis. Comput. Graphics 15 (2009) 436–452.
[12] B. Jeong, L. Renambot, R. Jagodic, R. Singh, J. Aguilera, A. Johnson, J. Leigh, High-performance dynamic graphics streaming for scalable adaptive graphics environment, in: Supercomputing, 2006, SC'06. Proceedings of the ACM/IEEE SC 2006 Conference, 2006, p. 24.
[13] R. Jagodic, L. Renambot, A. Johnson, J. Leigh, S. Deshpande, Enabling multi-user interaction in large high-resolution distributed environments, Future Gener. Comput. Syst. 27 (2011) 914–923.
[14] L. Smarr, M. Brown, C. de Laat, Special section: OptIPlanet—the OptIPuter global collaboratory, Future Gener. Comput. Syst. 25 (2009) 109–113.
[15] T.A. DeFanti, J. Leigh, L. Renambot, B. Jeong, A. Verlo, L. Long, M. Brown, D.J. Sandin, V. Vishwanath, Q. Liu, M.J. Katz, P. Papadopoulos, J.P. Keefe, G.R. Hidley, G.L. Dawe, I. Kaufman, B. Glogowski, K.-U. Doerr, R. Singh, J. Girado, J.P. Schulze, F. Kuester, L. Smarr, The OptIPortal, a scalable visualization, storage, and computing interface device for the OptiPuter, Future Gener. Comput. Syst. 25 (2009) 114–123.
[16] S. Nam, B. Jeong, L. Renambot, A. Johnson, K. Gaither, J. Leigh, Remote visualization of large scale data for ultra-high resolution display environments, in: Proceedings of the 2009 Workshop on Ultrascale Visualization, 2009, pp. 42–44.
[17] A. Cedilnik, B. Geveci, K. Moreland, J. Ahrens, J. Favre, Remote large data visualization in the paraview framework, in: Proceedings of the Eurographics Parallel Graphics and Visualization, 2006, pp. 162–170.
[18] D.P. Bovet, M. Cesati, Understanding the Linux Kernel, third ed., O'Reilly Media, Inc., Sebastopol, CA, 2005.
[19] M.K. McKusick, G.V. Neville-Neil, Thread scheduling in FreeBSD 5.2, Queue 2 (2004) 58–64.
[20] M.E. Russinovich, D.A. Solomon, A. Ionescu, Windows Internals, fifth ed., 2009.
[21] R. Motwani, S. Phillips, E. Torng, Non-clairvoyant scheduling, in: Proceedings of the Fourth Annual ACM–SIAM Symposium on Discrete Algorithms, 1993, pp. 422–431.
[22] S. Evans, K. Clarke, D. Singleton, B. Smaalders, Optimizing Unix resource scheduling for user interaction, in: Proceedings of the USENIX Summer 1993 Technical Conference on Summer Technical Conference, vol. 1, 1993, pp. 205–218.
[23] Y. Etsion, D. Tsafrir, D.G. Feitelson, 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, 2004, pp. 110–115.
[24] H. Zheng, J. Nieh, RSIO: automatic user interaction detection and scheduling, in: Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, 2010, pp. 263–274.
[25] C.L. Liu, J.W. Layland, Scheduling algorithms for multiprogramming in a hard-real-time environment, J. ACM 20 (1973) 46–61.
[26] C.W. Mercer, S. Savage, H. Tokuda, Processor capacity reserves: operating system support for multimedia applications, in: International Conference on Multimedia Computing and Systems, 1994, pp. 90–99.
[27] H. Tokuda, T. Nakajima, P. Rao, Real-time mach: towards predictable real-time systems, in: USENIX Mach Symposium, 1990, pp. 73–82.
[28] D.K.Y. Yau, S.S. Lam, Adaptive rate-controlled scheduling for multimedia applications, IEEE/ACM Trans. Netw. 5 (1997) 475–488.
[29] M.B. Jones, D. Roşu, M.-C. Roşu, CPU reservations and time constraints: efficient, predictable scheduling of independent activities, SIGOPS Oper. Syst. Rev. 31 (1997) 198–211.
[30] B. Urgaonkar, P. Shenoy, T. Roscoe, Resource overbooking and application profiling in shared hosting platforms, in: Proceedings of the 5th Symposium on Operating Systems Design and Implementation, 2002, pp. 239–254.
[31] D.B. Golub, Operating system support for coexistence of real-time and conventional scheduling, in: School of Computer Science, Carnegie Mellon University, 1994.
[32] P. Goyal, X. Guo, H.M. Vin, A hierarchical CPU scheduler for multimedia operating systems, in: J. Kevin, Z. HongJiang (Eds.), Readings in Multimedia Computing and Networking, Morgan Kaufmann Publishers Inc., 2001, pp. 491–505.
[33] J.H. Anderson, J.M. Calandrino, U.C. Devi, Real-Time Scheduling on Multicore Platforms, in: Real-Time and Embedded Technology and Applications Symposium, 2006. Proceedings of the 12th IEEE, 2006, pp. 179–190.
[34] L. Zhang, Virtual clock: a new traffic control algorithm for packet switching networks, in: Proceedings of the ACM Symposium on Communications Architectures & Protocols, 1990, pp. 19–29.
[35] I. Stoica, H. Abdel-Wahab, K. Jeffay, S.K. Baruah, J.E. Gehrke, C.G. Plaxton, A proportional share resource allocation algorithm for real-time, time-shared systems, in: Real-Time Systems Symposium, 1996, 17th IEEE, 1996, pp. 288–299.
[36] I. Stoica, H. Abdel-Wahab, K. Jeffay, On the duality between resource reservation and proportional share resource allocation, in: Proc. of Multimedia Computing and Networking, 1997, pp. 207–214.
[37] K.J. Duda, D.R. Cheriton, 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, 1999, pp. 261–276.
[38] J. Nieh, M.S. Lam, A SMART scheduler for multimedia applications, ACM Trans. Comput. Syst. 21 (2003) 117–163.
[39] A. Chandra, M. Adler, P. Goyal, P. Shenoy, 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, vol. 4, 2000, p. 4.
[40] H. Zixia, A. Ahsan, A. Pooja, N. Klara, W. Wanmin, Towards the understanding of human perceptual quality in tele-immersive shared activity, in: Proceedings of the 3rd Multimedia Systems Conference, 2012, pp. 29–34.

[41] C. Kuan-Ta, T. Cheng-Chun, X. Wei-Cheng, OneClick: A Framework for Measuring Network Quality of Experience, INFOCOM 2009, IEEE, 2009, pp. 702–710.
[42] W. Wu, A. Arefin, R. Rivas, K. Nahrstedt, R. Sheppard, Z. Yang, Quality of experience in distributed interactive multimedia environments: toward a theoretical framework, in: Proceedings of the 17th ACM International Conference on Multimedia, 2009, pp. 481–490.
[43] J.Y.C. Chen, J.E. Thropp, Review of low frame rate effects on human performance, IEEE Trans. Syst. Man. Cybern. A 37 (2007) 1063–1076.
[44] M. Claypool, K. Claypool, Perspectives, frame rates and resolutions: it's all in the game, in: Proceedings of the 4th International Conference on Foundations of Digital Games, 2009, pp. 42–49.
[45] R.T. Apteker, J.A. Fisher, V.S. Kisimov, H. Neishlos, Video acceptability and frame rate, IEEE MultiMedia 2 (1995) 32–40.
[46] S.R. Gulliver, G. Ghinea, Changing frame rate, changing satisfaction? [multimedia quality of perception], in: IEEE International Conference on Multimedia and Expo, ICME'04, 2004, pp. 177–180.

Sungwon Nam received M.S. degree in Computer Science from the University of Southern California in 2005. He is currently pursuing a Ph.D. degree in Computer Science in the University of Illinois at Chicago. His research interest lies in the area of interactive visualization system and a collaborative environment.

Khairi Reda received his M.S. degree in Computer Science from the University of Illinois at Chicago, where he is currently pursuing a Ph.D. degree in Computer Science. His research interests are in data visualization and human–computer interaction. He works closely with domain scientists to develop novel, interactive visualizations that apply perceptual and cognitive techniques for the exploration of complex scientific data.

Luc Renambot received the Ph.D. degree from the University of Rennes-1, France, in 2000, conducting research on parallel rendering algorithms for illumination simulation. Holding a postdoctoral position at the Free University of Amsterdam, Amsterdam, The Netherlands, until 2002, he worked on bringing education and scientific visualization to virtual reality environments. Since 2003, he has been with the Electronic Visualization Laboratory, University of Illinois at Chicago, Chicago, first as a Postdoctoral Researcher and now as a Research Assistant Professor, where his research topics include high-resolution displays, computer graphics, parallel computing, and high-speed networking.

Andrew Johnson is an Associate Professor in the Department of Computer Science and a member of the Electronic Visualization Laboratory, University of Illinois at Chicago, Chicago. His research and teaching focus on interaction and collaboration using advanced visualization displays and the application of those displays to enhance discovery and learning.

Jason Leigh is a Professor of Computer Science and Director of the Electronic Visualization Laboratory and the Software Technologies Research Center at the University of Illinois at Chicago, Chicago. He is an internationally recognized pioneer in collaborative virtual reality. His prior projects and research for which he is best known include: the OptIPuter, GeoWall, CoreWall, LambdaVision, Tele-Immersion, and Reliable Blast UDP. His research for the past ten years focused on ultraresolution display-rich collaboration environments amplified by high-performance computing and networking. His current research focuses on human augmentics—the development of technologies for expanding the capabilities and characteristics of humans.