Using Multi-threading and Server Update Pushing to Improve the Performance of VNC for a Wall-Sized Tiled Display Wall

Yong Liu, John Markus Bjørndalen, and Otto J. Anshus

Department of Computer Science, University of Tromsø
{yongliu,jmb,otto}@cs.uit.no

Abstract. Display walls are wall-sized, high-resolution displays, typically built using several computers, each driving a projector or an LCD. The VNC (Virtual Network Computer) model is a simple way of creating desktops large enough for display walls by using a centralized virtual frame buffer. However, performance suffers significantly when the resolution increases due to the centralized server locating and compressing updates for the display computers. Another problem is that the display computers request and receive updates independently, resulting in an inconsistent view. TiledVNC is developed to better adapt VNC to a display wall and improve performance over an existing implementation, TightVNC. The changes include multi-threading, a server push update protocol, and pushing updates for the same frame to all viewers. To evaluate our system, we play two videos on our 22 megapixel display wall. Compared to TightVNC, TiledVNC increases the frame rate with up to 46% for a 6.75 megapixel video.

Keywords: TiledVNC, TightVNC, display wall, VNC, performance, high resolution.

1 Introduction

Display walls [14] are large, wall-sized, high-resolution displays constructed by tiling a number of smaller LCD displays or projectors driven by several computers. While a single display typically support a few megapixels, all the displays taken together add up to typically from twenty to several hundreds of megapixels. This allows for having a lot of data and information available simultaneously. Many windows from several applications can be displayed, or the whole display wall can be dedicated to high-resolution visualizations. When viewed at a distance, overview and coarser structures become visible; while viewed up close, users can read the text and see finer details. The size of the display wall lets a single user walk along the wall to view different parts of the data being displayed, and it lets multiple users have space enough either to work using different regions of the display wall or together in a collaborative manner. The size in combination with a high pixel count gives a display wall many more usage areas than a standard projector projected at a distance onto a large screen to create a large image.

P. Mueller, J.-N. Cao, and C.-L. Wang (Eds.): Infoscale 2009, LNICST 18, pp. 306–321, 2009.

By having a standard desktop expanded to fill a display wall, the users will be able to use unmodified legacy applications while taking advantage of the large size and pixel count. However, standard operating systems don't support having several PCs and displays and coordinating them into a display wall with a correspondingly large desktop. A simple approach to create a large enough desktop is to use the VNC (Virtual Network Computer) model [11]. A VNC server maintains a virtual desktop with dimensions matching a display wall. The applications run on the same computer as the VNC server, but they only see what they believe to be the normal X server. Each display wall computer runs a VNC viewer requesting updates from the server corresponding to its tile of the virtual desktop, and physically displays it. We have used this approach for several years, and it is also used in the NCSA Display Wall In a Box (http://www.ncsa.uiuc.edu/Projects/DWiB/).

VNC transfers updates as compressed pixel data. The advantage is that this is portable across platforms and simplifies implementation of the viewers compared to solutions that forward graphics API calls to the clients. The main drawback is overhead. Applications render into the server's virtual frame buffer, and the VNC server needs to keep track of dirty regions[1] in the virtual frame buffer. Large dirty regions result in large updates. These must be compressed and then transmitted to the viewers. However, the VNC server uses no graphics hardware support to do this. Consequently, encoding is done by the CPU with the frame buffer for the virtual desktop in the computer's random access memory. If no compression is used, the pixel copying and processing is reduced, but now the network can become a bottleneck.

Another problem with using VNC for a display wall is that the tiles update independently of each other, whenever the individual VNC viewers request an update. Because of this, the tiles can sometimes display content from different frames making text and images display inconsistently across the tiles.

Other approaches to using a display wall may give better performance, but have drawbacks. Linking an application running on some computer with a modified graphical library that transparently will redirect output to the display wall can improve the performance in comparison with VNC [6]. This is because the library transmits graphics commands instead of the much larger amount of pixel data to the display computers. The drawback is that the dependence on specific libraries limits the number of applications which can be used without changes.

Modifying applications to run fully or partially on the computers that drive the individual displays may provide the best performance, but requires access to the source code of the application. This may be too complex and time-consuming depending on the application. The individual copies of the application need to be synchronized, data may need to be exchanged, and user input and other I/O have to be solved on an application-by-application basis.

We believe that using VNC is a simple and flexible approach to providing users with a familiar desktop environment where standard applications can run

[1] Regions that have been modified since the last update sent to one or more viewers covering that region.

unmodified. For static documents, like windows which rarely are updated and moved around, a low frame rate is not a significant problem. For dynamic documents like videos, animations and complex web sites, the frame rate becomes low enough to frustrate the user. The performance and the consistency of each frame across the tiles should be improved to better support a wider range of display wall usage scenarios.

2 Related Work

Distributed Multihead X, XDMX [4], makes it possible to create a high-resolution display wall desktop. Client applications connect to a front-end X server, which transparently transmits X commands from its clients to a set of back-end X servers running on the computers of the display wall. No data compression is used. In XDMX the back-end servers are meant to receive only their part of the data. This is very important for dynamic documents like videos, web sites with animation and similar to reduce network traffic and achieve good frame rates. We have observed a high startup time to have a static document like a high-resolution 25-megapixel image displayed. However, after the initial startup time, panning the image around is very smooth.

MultiStream[9] is a system using freely available video encoders to create lossy videos on-the-fly of a desktop, and then stream the videos to a display wall or PCs for play-back using common media players. The system is transparent to the applications. However, when used to create a video of the large desktop of a display wall, capturing and encoding the pixels are CPU intensive, and the performance suffers.

In the Chromium approach [6] a set of individual tiles of a display wall are made to appear as a single display to the applications. Display output from applications using OpenGL is transparently redirected to the display wall. The advantages include that applications don't have to be changed, and the model is flexible in that it can be used both for single process and multiple process parallel applications. A limiting factor is that the applications must use OpenGL. We have also experienced performance limitations when using Chromium on a game with more than four tiles [12].

SAGE [7] is a flexible and scalable software approach to sending display output from applications to a display wall. However, applications are required to be rewritten with the SAGE application interface library. Xinerama [15] is an extension to the X window system using multiple graphical devices to create one large virtual display. However, it requires the graphical devices to coexist on a single computer. These approaches are designed to let an application display to a display wall. They are not suitable as a way to do a high resolution display wall desktop.

3 Applying VNC to a Display Wall

A display wall is comprised of several computers and projectors (see Figure 1). The projectors are physically aligned into a matrix, and (rear-) projected onto a

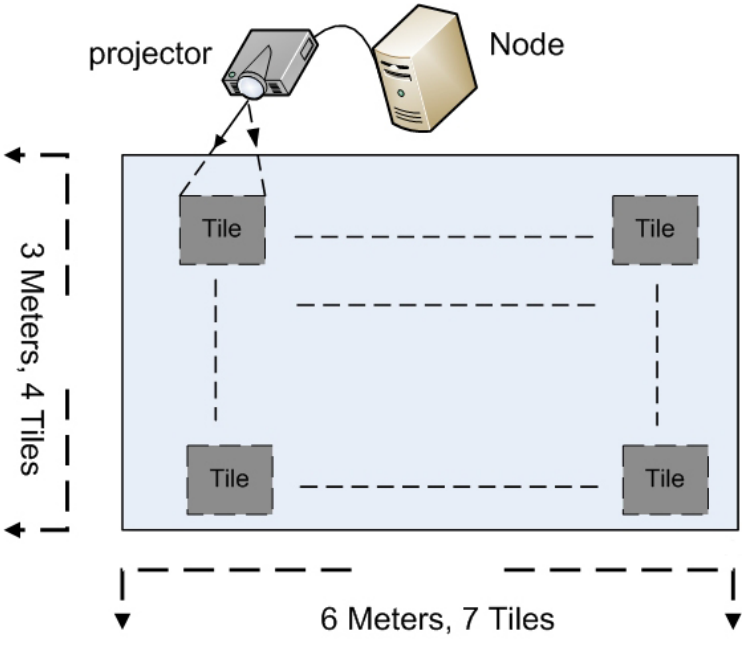

Fig. 1. The Architecture of Display Wall. Each projector is plugged into a computer that displays one tile of the desktop.

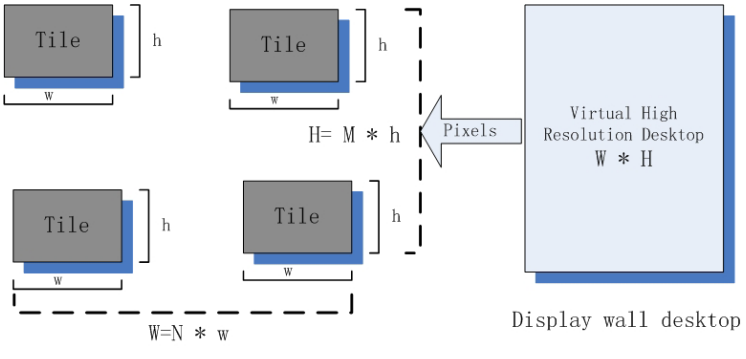

Fig. 2. The Architecture of Display Wall Desktop

screen. The VNC (Virtual Network Computer) system is a simple way of creating a desktop large enough to fill a display wall. See Figure 2. A VNC server runs on a computer together with the applications displaying to the desktop.

The VNC server handles a virtual desktop large enough to fill the display wall. Each display computer runs a VNC viewer requesting desktop updates from the server for the area of the desktop corresponding to its tile, and displays it.

When an application needs to produce output to the desktop, it uses normal X library calls to request the X server to output pixels. This call is, transparently to the application, sent to the VNC server which updates the virtual desktop correspondingly. To get updates, a VNC viewer polls the VNC server. Based on its position in the projector matrix, each viewer requests the corresponding pixel area from the VNC server. The server then, on demand, checks for updates in the desktop area covered by the requesting viewer. If there are updates, the server encodes and transmits them to the VNC viewer.

4 Methodology

To document what the problems are when using VNC to drive a desktop for a display wall, we initially did performance measurements on an existing implementation of the VNC model, TightVNC [13]. This was done on a single core computer with hyper-threading (more below). We then used the insight from these experiments to develop TiledVNC. Because one of the modifications was to change the server from a single-threaded to a multi-threaded model, we used a quad-core computer to let the multi-threading have better effect than it would have on a single-core computer. We therefore did a complete new set of experiments both on TightVNC and TiledVNC to allow us to compare them directly and see how the modifications impacted performance and why.

For the experiments we use a display wall with 28 computers, each with a projector. The network is a gigabit switched Ethernet. The screen is 6 meters wide and 3 meters high for a total area of $18m^2$. Onto this screen we rear-project in a 7x4 matrix. Each projector has 1024x768 pixels giving in total 7168x3072 pixels, or almost 22 megapixels.

We use the TightVNC 1.3.9 implementation of the VNC server. The display nodes run either the RealVNC [1] viewer or a VNC viewer we have implemented. We found the RealVNC viewer to give TightVNC better performance than the TightVNC viewer and decided to use it for the initial experiments. We use our own implementation of the VNC viewer on both TightVNC and TiledVNC for the second set of experiments. Our viewer gave TightVNC better performance than the RealVNC viewer. Using the same viewer for TightVNC and TiledVNC in the second set of experiments was done to better remove the viewer as a factor. Another reason for using our own viewer is that it is customized for the display wall domain, removing all keyboard and mouse handling. Our own viewer use the SDL [2] cross-platform multimedia library.

The display wall computers run 32-bit Rocks 4.1 Linux on a cluster of Dell Precision WS 370s, each with 2GB RAM and a 3.2 GHz Intel P4 Prescott CPU.

For the initial set of experiments on TightVNC the VNC server runs on a computer identical to the display node computers running the viewers, and also with Rocks 4.1 Linux. For the second set of experiments on TightVNC and TiledVNC we used a quad-core 3.8 GHz Intel Xeon CPU and 8 G RAM memory. The operating system is 64-bit Redhat Linux RHEL 5. While the TightVNC server could only use one thread, we configured the TiledVNC server to use four threads to handle updates.

To emulate updates to a high-resolution desktop we use two videos of 3 and 6.75 megapixel. We play the videos using a media player, one at a time, at the computer running the VNC server, and display them to the VNC virtual desktop. The VNC viewers request updates from the VNC server as often as they can manage, and display them physically to the display wall tiles.

We made the two videos using FFmpeg [5], and at the resolution 2048x1536 (R1) and 3072x2304 (R2). The videos are encoded as MPEG4 at 30 FPS, each video lasting 120 seconds for a total of 3600 frames. The VNC server compression ratio using hextile encoding on the videos is about 5.

The R1 video covers exactly four (two by two) tiles and its area is $2.6m^2$. This area is representative for the area used by several typical streaming applications on the display wall. At nine tiles (three by three) the R2 video covers an even larger area of the display wall.

We use ffplay [5] as the video media player. The media player runs at the same computer as the VNC server, displaying into the virtual desktop. It plays images one by one, and does not drop images because of timeouts. This is important, as the computer and VNC server cannot keep up with the 30FPS framerate, slowing down ffplay and extending the time spent on playing back the video.

For each experiment we define a virtual frame buffer (a desktop) at the VNC server matching the configuration of the video we play back, and at a color depth of 32 bits. By doing this we get each video to exactly cover the whole VNC virtual desktop, avoiding complicating issues arising from having the video play back partially on more than four (or nine) tiles of the display wall.

In [10] a method is described using slow-motion benchmarking to measure thin-client performance. Following the same ideas to benchmarking, we use two videos. While these videos played back well on all the computers we had available, a 22 megapixel video did not, making it impractical to use it in the experiments. However, the lower resolution videos do reflect interesting collaboration settings where full desktop updates are rare. One such setting is collaboration using many static documents and a few videos and web sites with animations.

We have three performance metrics:

FPS: "Frames per second" as seen at the display wall or at the VNC server. This is the number of updates received by the VNC viewers or by the VNC server per second. While we measure and calculate the FPS per tile, we only report on the average FPS over all four or nine tiles. Higher is better. The FPS for the server is computed as the number of video frames (3600) divided by the time it takes ffplay to actually play the video (which will be longer than 120 seconds).

UFP: "Updated frames percent": The percentage of all video frames played back by the media player which is received by the VNC viewers. First, we count the accumulated average number of updates sent to the VNC viewers until a video finishes. We then, because each video is 3600 frames long, divide the accumulated average of frame updates by 3600 to compute the UPS. This is possible since ffplay does not drop frames, and will draw every frame on the desktop. UFP is 100% when each update is seen by all viewers, and is less than 100% when the

application updates the VNC virtual frame buffer faster than the viewers get updates. The higher the UFP, the less updates are missed by the VNC viewers. Higher is better.

"Time": This is the time we measured it took for ffplay to play back into the virtual frame buffer the 120 seconds long video. Closer to 120 is better.

5 Problems When Using VNC for a Display Wall

To document the performance behavior encountered when applying VNC to a high-resolution desktop for a display wall, we did a set of experiments measuring the performance of TightVNC.

5.1 Performance Results for TightVNC

The results are shown in Table 1. The media player, ffplay, at the VNC server computer attempts to play back the videos at 30 FPS. The measurements show that even for the lowest resolution 3 megapixel video (R1) only 8.1 FPS is received by the display wall viewers.

Table 1. Performance of Display Wall Desktop. The numbers show the Frames Per Second (FPS) and Updated Frames Percent (UFP) for two resolutions: 2048x1536 (R1) and 3072x2304 (R2).

Video Resolution	2048x1536 (R1)	3072x2304 (R2)
FPS	8.1	4.3
UFP	67%	99.5%
Time	297	836

We observe that the media player has enough resources to play back into the virtual frame buffer, 33% more frames are displayed at the VNC server than what the VNC server sends to the VNC viewers.

We believe that this slowdown is in part due to the application updates being delayed in the VNC server while the VNC server is busy handling the VNC clients. TightVNC is single-threaded, and is not able to handle XLib requests while compressing and sending data to VNC viewers. Another factor is that the media player and the VNC server run on the same host, competing for resources.

In the case of the higher resolution 6.75 megapixels video (R2), the FPS decreases to 4.3 while at the same time almost all of the frames written to the virtual frame buffer get sent to the VNC viewers.

In practical terms, a user viewing the videos on a display wall will experience that the 120 seconds long videos need 297 and 836 seconds, respectively, to play back.

5.2 Discussion and Conclusion

The network bandwidth could have been a bottleneck in these experiments because the 6.75 megapixel video has frames with 27MB per frame. However, pixel compression reduces the network traffic, and with a compression ratio of about five we can achieve about 20 FPS on a network with 1000Mb/sec bandwidth (which is about what a gigabit Ethernet can support). However, to find out if pixels can be compressed, each pixel must be compared at least once. Using the CPU and memory (instead of a graphics card) adds to the latency of X operations. We observe that VNC is slow when large areas of the desktop needs to be updated. Modified pixels must be encoded to reduce the network bandwidth consumed when transferring updates to the viewers, but this encoding turns into a bottleneck for large display walls. The single-threaded implementation of the server contributes to the scalability issues as it cannot benefit from newer multi-core CPUs. We conclude that a multi-threaded model is needed to benefit from multi and many-core architectures.

The media player attempts to play back at 30 FPS, but, for the R2 video, only 4.3 FPS is actually written to the virtual frame buffer. The VNC viewers have enough resources to queue up new update requests before the media player is finished updating a frame. The VNC server has enough resources to send all of these updates to the viewers.

If a viewer requests an update when a dirty region is registered for that viewer, the server will immediately compute and return an update. If there is no update available, the server will place the viewer on a waiting queue. The first XLib update that modifies a region corresponding to the viewer will start a timeout timer for the viewer. At timeout, a set number of milliseconds, the server will compute an update packet and send it to the client. This allows the VNC server to aggregate multiple small updates before sending updates to the viewer. However, this does not work well for the large frame updates generated by the R2 video. For large updates the time of compressing and initiating transferring it to a viewer is more than the default update timeout delay of 40ms. After an update is sent, the timers for the other waiting viewers have expired, and the server has to send the next of these updates immediately.

A high UFP is good if the server and viewers manage to provide this without slowing down the application too much, but our measurements show that the media player is slowed down severely. As a result, the 2-minute (3600 frames) R2 video plays in 14 minutes, when displayed on the R2 desktop.

We have observed that the desktop does not keep the display wall consistent. The tiles in the display wall are updated independently of each other because the VNC server handles update requests from a viewer without considering recent or pending updates for the other viewers. This can result in some tiles having older output than other tiles, and happens more frequently when the load of the server increases. We conclude that VNC model is not well suited when used to support a desktop for a display wall, which have a number of tiles that we want to keep consistent, and that a solution for this problem is needed.

Please see [8] for more details about the performance problems encountered when using VNC for a display wall desktop and on how we improved the performance of TightVNC without changing the VNC model and update protocol.

6 Improving TightVNC

To adapt VNC to a display wall and improve performance, we did several changes, and used the TightVNC implementation of VNC as a starting point for the resulting TiledVNC system.

To utilize multi-core architectures, we change the VNC server from being single-threaded to multi-threaded.

To control the load on both the server and client side we change the update protocol of the VNC server from a viewer pull to a server push model. The server initially waits until all viewers connect to it, and then the server periodically pushes updates to the viewers.

To reduce the period when tiles show content from different frames, we only push updates for the same frame to all viewers. This was simplified by the server push model.

We also remove the mouse and keyboard input handler from all viewers because in the case of a display wall using VNC all input devices can be connected directly to the VNC server computer instead of to the viewers. We do not report further here on the detailed effect of this change.

6.1 From Single-Threading to Multi-threading

Figure 3 illustrates the architecture of the TiledVNC multi-threaded VNC server. The idea is to have a set of threads waiting on a timer. When a timeout happens, the threads get a task from a pool of tasks. Each task indicates an area of the virtual frame buffer corresponding to a tile of the display wall. A thread scans the frame buffer for dirty regions that needs to be updated, and if there are any, it will compress the data and send it to the corresponding viewer. Then the thread goes to the pool of tasks and gets a new area to check. A counter is used to keep track of the number of remaining tasks. When all tasks are done, the threads block for the next timeout.

We implemented the server in C, using the Pthread [3] thread package. We use a Pthread monitor condition variable to have threads wait.

The period between timeouts, the delay, lets the server control its load, and impact the performance.

Different numbers of threads can be used. However, in this paper we only report on using four threads, that is, one thread per core of the computer running the VNC server.

6.2 From Viewer Pull to Server Push

A VNC server usually uses a viewer pull model. The server waits for requests for updates from the VNC viewers, and services them as soon as they arrive.

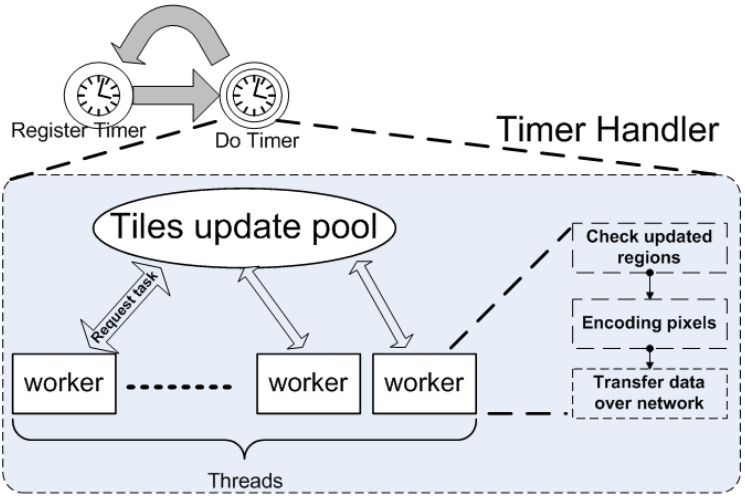

Fig. 3. Multi-thread Display Wall Desktop

If the server finds an update it is sent as soon as possible. This approach lets the viewers request an update only when they are ready, and the server only has to send updates, and thereby creating network traffic and using bandwidth, when there are viewers ready to receive. However, by just requesting an update, a viewer enforces the server to look for updates and encode them as soon as possible.

If we analyze some interesting permutations of resolution, network bandwidth, and number of viewers we find:

Low desktop resolution, low network bandwidth, few viewers: Even when a high level of compression is used, the low resolution and small number of viewers do not give the server a too high load, and the media player (in our case) gets more CPU resources. The server can keep up with the requests from the viewers. The result is a relatively good frame rate.

High desktop resolution, low network bandwidth, many viewers: A high level of compression is needed, and this together with the high number of viewers give the server a high load, and the media player gets less CPU resources. The server cannot keep up with the requests from the viewers. The low bandwidth will further delay the receiving of updates. The result is a very low frame rate.

High desktop resolution, high network bandwidth, many viewers: A high level of compression is needed, and this together with the high number of viewers give the server a high load, and the media player gets less CPU resources. The server cannot keep up with the requests from the viewers. However, the high bandwidth will mean that the network is not the bottleneck, and will not reduce the frame rate further. The result is still a low frame rate.

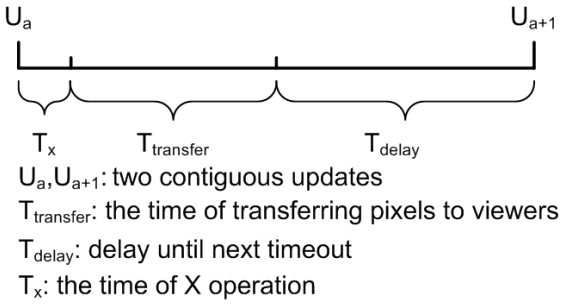

Fig. 4. Time Distribution

The viewer pull based VNC model works better for usage domains with a standard sized PC desktops with just one or a few viewers, than it does for a high-resolution display wall desktop with potentially many more viewers. A high-resolution VNC server with multiple clients spends more time on encoding pixels than transferring the pixels over the network. The many viewers also force the server to service them as soon as possible. Consequently, the server has little influence over its own work load. The performance of the VNC server is impacted by the frequency of updates.

The time ΔT between two contiguous updates is the sum of T_x, $T_{transfer}$ and T_{delay}, as shown in Figure 4.

T_x and $T_{transfer}$ are insignificant for the standard PC desktop resolution case. When the size of the desktop increases, T_x and $T_{transfer}$ grow larger. T_x in particular can become large in the high resolution. T_{delay} is the duration between updates. During this time the media player application may also get more CPU resources because the server may wait idle for new viewer requests or for the timeout. Reducing the update frequency will reduce the server load. This can be achieved by increasing T_{delay}.

We have changed the viewer pull model to a server push model to let the server have better control over its work load and the update rate. The assumption is that the viewers always will have more resources available than they need to display the updates sent from the server.

6.3 Consistency between Tiles

The viewer pulls update protocol used by VNC, when used in a tiled display wall, results in a possibility for viewers displaying content from different frames. This is because each viewer requests and gets updates independently of each other. The server push update protocol makes it simple to reduce the period when tiles show content from different frames by simply pushing updates from the same frame to all viewers before the next frame is pushed out. Even if viewers receive the updates at slightly different times, the updates are from the same frame. This works well as long as the viewers have resources enough to receive

and display all updates. This is a realistic assumption for a display wall with many viewers, and a heavily loaded server.

7 Comparing TightVNC vs. TiledVNC

To document the impact of changing VNC in the way described above, and to be able to do comparisons, we did a set of experiments measuring the performance of TightVNC and TiledVNC on the same platform and using our VNC viewer.

7.1 Performance Results

The result from the experiments is shown in Figures 5, 6 and 7.

The X-axis is the delay between timeouts, T_{delay} in millisecond. The Y-axis is the frame rate (FPS) or the updated frames percent (UFP).

The results show that the best frame rate as seen by the **viewers** for the 3 megapixels video was achieved when updates were done every 30ms. In this case TiledVNC increased the frame rate to 14 FPS, or about 34% more than TightVNC. About 80% of the updates done at the server side were displayed at the display wall.

The best frame rate as seen by the viewers for the 6.75 megapixels video was achieved when updates were done more rarely, at 70ms. In this case TiledVNC increased the frame rate to 6 FPS, or about 46% more than TightVNC. About 100% of the updates done at the server side were displayed at the display wall.

TiledVNC is performing better than TightVNC in all cases, but when the delay between updates increases, the frame rate decreases for both systems to between 3-6 FPS.

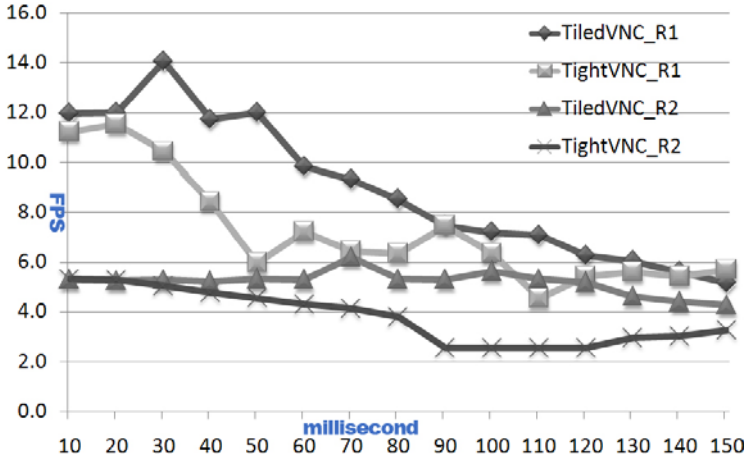

Fig. 5. FPS at the Display Wall

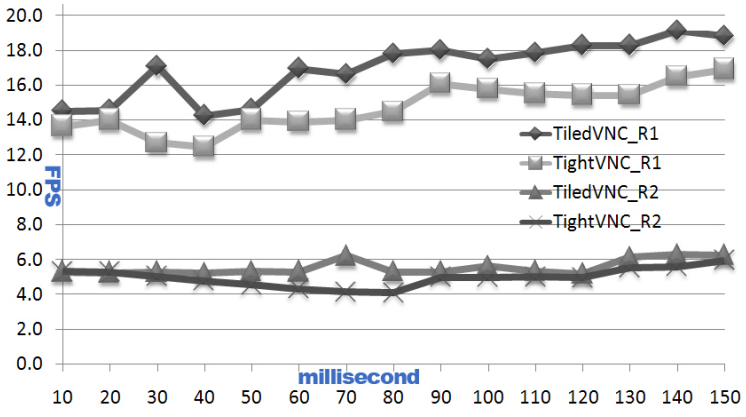

Fig. 6. FPS at the VNC Server

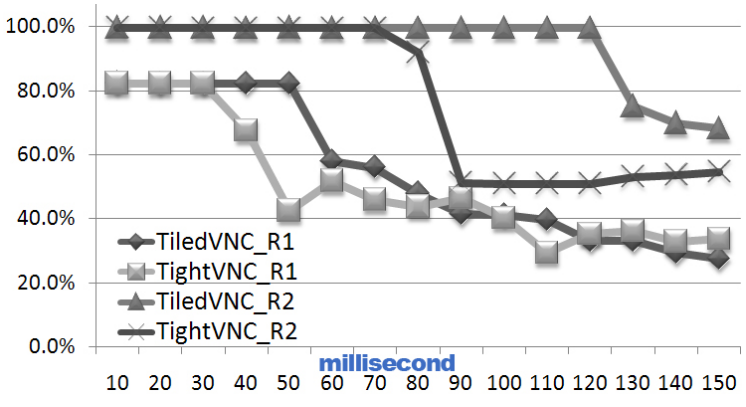

Fig. 7. UFP

The results show that the best frame rate at the **server** for the 3 megapixels video was achieved when updates were done every 140ms. In this case TiledVNC achieved a frame rate of 19 FPS, about 15% better than TightVNC.

The best frame rate at the server for the 6.75 megapixels video was achieved when updates were done at 70ms and above 130ms. In this case TiledVNC increased the frame rate to about 6 FPS, but this is only slightly better than TightVNC.

TiledVNC is performing better than TightVNC in all cases, but less so than as seen by the viewers. At the server, when the delay between updates increases, the frame rate increases for both systems when playing the R1 video from about 14-15 to 17-19 FPS. However, for the R2 video, the frame rate stays at around 5-6 FPS almost independently of delay time. Playing back R2 and running the server seems to saturate the computer used. In this case a faster computer should give better performance.

Table 2. Time to Play Back a 120 Seconds Long Video

T_{delay}	30 ms	70 ms
TiledVNC_R1	211	216
TiledVNC_R2	679	579
TightVNC_R1	283	257
TightVNC_R2	712	869

From Figure 7, it can be seen that TightVNC begins earlier to skip frames than TiledVNC. This is because TightVNC uses a single core, while TiledVNC benefits from using all four cores. However, both systems have the same UFP at delays below 30 and 70 ms respectively for R1 and R2.

In practical terms, a user viewing the videos on a display wall will experience that the 120 seconds long videos takes longer to play back. TightVNC need for the best cases 257 and 712 seconds, respectively, to play back R1 and R2. For TiledVNC the videos take for the best cases 211 and 579 seconds to play back.

Comparing the results for TightVNC in Table 2 with Table 1, the performance is improved because of the more powerful computer used in the last set of experiments, and because using 30ms or 70ms give better performance than the default 40ms used in the first set of experiments.

7.2 Discussion

Although the performance of TiledVNC is better than TightVNC, it still suffers from low FPS. One main reason is that updating the virtual frame buffer has no hardware (GPU) support. However, we still benefit from multiple cores. The compression rate of the Hextile encoding used by TightVNC and TiledVNC is about 5. When $T_{delay} = 30ms$, TiledVNC sends about 34 MB/sec and 29 MB/sec into the network when playing back respectively the 3 and 6.75 megapixels videos. The gigabit Ethernet can support about 100MB/sec. With more cores the multi-threaded TiledVNC should be able to push more updates into the network. For the 3 megapixel video, we estimate that using eight cores may give us about 19 FPS. This is about 45MB/sec, and still less than what a gigabit Ethernet will support. This is interesting because it indicates that there are performance benefits for TiledVNC to be expected from more cores, and it is simpler to update to a computer with more cores than to update the whole network.

The TiledVNC server used only four threads to scan for and compress updates. For future experiments we will document the effect of using more threads.

8 Conclusion

VNC is a simple way of supporting a high-resolution desktop for a tiled display wall. It will allow legacy applications to run as before, but displaying to a much larger desktop than a standard sized PC desktop.

We have done a set of experiments on TightVNC to identify the performance behavior and bottlenecks when using VNC to support a high-resolution desktop for a tiled display wall. Based on the results we changed VNC from single to multi-threaded model, and changed the update protocol from a viewer pull model to a server push model. We also achieved to keep the viewer content more consistent across frames. We then did a set of new experiments on the new TiledVNC system to document the impact on performance coming from these modifications. We used a 3 megapixel and a 6.75 megapixel video to emulate frequent updates to a display wall.

The new TiledVNC has better performance than TightVNC in all cases. Best case frame rates are improved about 34% and 46% when playing demanding videos at 3 and 6.75 megapixels. Even though the best frame rate at 14 FPS is about half of the 30 FPS the videos were made to play at, this will be enough in many situations where the updates to the display wall desktop are much less frequent than the frequent updates generated by a video covering the whole desktop.

The main bottleneck is the load on the computer running the VNC server and media player. Looking for updates in the virtual frame buffer and compressing them before sending the data to the viewers saturate the resources. However, we expect the multi-threaded TiledVNC to track well the progress of multi-core architectures, and also be able to benefit from further performance boost from a graphical processing unit (GPU). We are working on GPU support for VNC.

Acknowledgements. This work has been supported by the NFR funded project IKT 2010 159936: SHARE - A Distributed Virtual Desktop for Simple, Scalable, and Robust Resource Sharing across Computer, Storage, and Display Devices.

References

1. RealVNC, `http://www.realvnc.com/`
2. Simple DirectMedia Layer, `http://www.libsdl.org/`
3. Butenhof, D.R.: Programming with POSIX threads. Addison-Wesley Longman Publishing Co., Inc., Boston (1997)
4. Faith, R.E., Martin, K.E.: Xdmx: distributed multi-head x, `http://dmx.sourceforge.net/`
5. FFmpeg, `http://ffmpeg.mplayerhq.hu/`
6. Humphreys, G., Houston, M., Ng, R., Frank, R., Ahern, S., Kirchner, P.D., Klosowski, J.T.: Chromium: a stream-processing framework for interactive rendering on clusters. ACM Trans. Graph. 21(3), 693–702 (2002); 566639
7. Jeong, B., Renambot, L., Jagodic, R., Singh, R., Aguilera, J., Johnson, A., Leigh, J.: High-performance dynamic graphics streaming for scalable adaptive graphics environment (2006); 1188568 108
8. Liu, Y., Anshus, O.J.: Improving the performance of vnc for high-resolution display walls. In: CTS 2009: The 2009 International Symposium on Collaborative Technologies and Systems (2009)

9. Liu, Y., Anshus, O.J., Ha, P.H., Larsen, T., Bjørndalen, J.M.: Multistream a cross-platform display sharing system using multiple video streams. In: 28th International Conference on Distributed Computing Systems Workshops, ICDCS 2008., pp. 90–95 (2008)
10. Nieh, J., Yang, S.J., Novik, N.: Measuring thin-client performance using slow-motion benchmarking. ACM Trans. Comput. Syst. 21(1), 87–115 (2003)
11. Richardson, T., Stafford-Fraser, Q., Wood, K.R., Hopper, A.: Virtual network computing. IEEE Internet Computing 2(1), 33–38 (1998); 613221
12. Stødle, D., Hagen, T.-M.S., Bjørndalen, J.M., Anshus, O.J.: Touch-free multi-user gaming on wall-sized, high-resolution tiled displays. In: PerGames 2007: Proceedings of the 4th International Symposium on Pervasive Gaming Applications, pp. 75–83 (2007)
13. TightVNC, `http://www.tightvnc.com/`
14. Wallace, G., Anshus, O.J., Bi, P., Chen, H., Chen, Y., Clark, D., Cook, P., Finkelstein, A., Funkhouser, T., Gupta, A., Hibbs, M., Li, K., Liu, Z., Samanta, R., Sukthankar, R., Troyanskaya, O.: Tools and applications for large-scale display walls. IEEE Computer Graphics and Applications 25(4), 24–33 (2005)
15. Xinerama, `http://sourceforge.net/projects/xinerama/`