De-centralizing the VNC Model for Improved Performance on Wall-Sized, High-Resolution Tiled Displays

Daniel Stødle
daniels@cs.uit.no

John Markus Bjørndalen
jmb@cs.uit.no

Otto J. Anshus
otto@cs.uit.no

Abstract

This paper presents changes to the Virtual Network Computing (VNC) model to improve performance on wall-sized, high-resolution tiled displays. VNC does not fully utilize data distributed to the tiles, noticeably reducing interactive performance when panning images and moving windows. By de-centralizing the VNC model, the VNC viewers can exchange pixels amongst each other, improving performance. The VNC server changes from individually servicing viewer requests, to servicing the viewers once everyone has requested an update. The model is implemented, and its performance documented through experiments. When panning images, the number of pixels refreshed is increased three times or more, while reducing the server's bandwidth by 74% and CPU load by 35%. When moving windows, the number of pixels refreshed is increased by a factor of 1.8, while reducing the server's bandwidth by 68% and CPU load by 19.7%. The paper demonstrates how conceptually simple changes to VNC, while complex to realize, can yield significant performance improvements.

1 Introduction

Using high-resolution, tiled display walls for visualization and collaboration is becoming increasingly popular. The high resolution and large physical size of display walls make them useful for visualizing data from many domains. Users often need to run applications written for standard desktop environments on the display wall, such as the weather forecasting application shown in Figure 1. Virtual Network Computing (VNC) [1], a remote desktop solution, is one way of achieving this. It is usually used to share regular-sized desktops, but for display walls, VNC can also be used to create a very high-resolution desktop. In the latter case, the desktop is maintained by a VNC server, which transmits

Figure 1: Weather forecasting on a tiled display wall with 28 projectors behind the canvas, using VNC to provide the desktop environment.

This paper was presented at the NIK-2007 conference; see http://www.nik.no/.

tiles of the desktop to corresponding clients (VNC viewers) running on a display cluster. Due to VNC's centralized approach to rendering and distributing pixels, it does not scale well to large display walls. With typical display walls ranging in resolution from 10 to 100 megapixels [2, 3] and beyond, a single complete refresh requires sending between 38 MB to 380 MB in total to the viewers.

This paper presents De-centralized VNC (DVNC). DVNC modifies the VNC model, allowing viewers to exchange pixels when screen content moves, but is not otherwise modified. Cases where this happens include panning large images, moving windows on the desktop or scrolling in windows. Work is delegated to the viewers, letting the server focus on sending new pixels rather than resending already transmitted pixels to the viewers. When the viewers receive pixels from both the server and each other, the correct ordering of display updates becomes important in order to preserve consistency of the display.

DVNC was implemented by modifying an open-source version of VNC [4], and its performance measured by comparing it to the original on a tiled display wall with a total resolution of 7168x3072 pixels. As a result, we found interactive performance to be significantly better when panning images and moving windows. The main contribution is the modified VNC model, where viewers go from being passive receivers of pixels, to become active participants in distributing pixels.

2 Related work

There has been much work on improving the performance and utility of VNC [1], including new compression techniques [5, 6] and support for 3D acceleration [7]. This paper is not focused on these aspects of VNC. DVNC instead aims at improving performance when using VNC to create a desktop on tiled display walls, by delegating work to viewers. In THINC [8], performance is improved compared to VNC and other remote desktop solutions by efficiently encoding and transferring raw graphics operations generated by applications. THINC is focused on thin-client usage, and is currently not suitable for use in creating desktop environments for display walls as it can only export desktops with the same resolution as the computer it is running on has.

Microsoft Remote Desktop and the X Window System [9] (X11) are two other ways of accessing or creating desktops over the network. Both approaches use drawing operations ("draw line", "draw string", and so on) to achieve good performance. The former is limited to a maximum resolution of 4096x2048, and does not allow different regions to be displayed by different viewers. Xdmx [10] can be used to enable X11 application to run on a tiled display wall. Xdmx acts as a proxy to a set of X servers running on the display cluster. This differs from DVNC in that no data is exchanged between the different X servers on the display cluster to improve performance. DVNC uses a single X server to render into a virtual framebuffer, which is then distributed to the tiles using the VNC protocol.

SAGE [11] is a system for streaming high-resolution graphics from rendering or storage clusters to one or several display walls. Pixel data is received by "SAGE Receivers" and then displayed. While this can be used to display multiple VNC desktops at once, no pixel data is exchanged between the different SAGE Receivers.

3 Model and design

When VNC is used on standard displays, a single viewer typically has access to the pixels for the server's entire desktop. On a tiled display wall, each viewer runs on its own computer showing a small region of the server's desktop, as shown in Figure 2 (a) and (b). In the original VNC model, the server does all the work. The viewers do nothing except receive and display pixels. In DVNC, this model is modified by letting the viewers exchange pixels amongst each other for a certain class of update operations. The purpose of this is to reduce the server's load and improve end-user performance. The viewers go from being passive receivers to being active slaves in a master-slave relationship to the server. Figure 2 (c) illustrates this change.

VNC uses the Remote Framebuffer (RFB) protocol [12] to send display updates from the server to the viewers. Viewers request the area they are interested in from the server, which responds with update operations for that area. The RFB protocol uses three operations to update a region of the display: Image Rect, Fill Rect and Copy Rect. The Image Rect operation contains a rectangular set of pixels which is drawn by the viewer at the location indicated by the rectangle. The Fill Rect operation is used to fill a rectangle with a given color. The Image and Fill Rect operations offer no obvious ways for distributing network load.

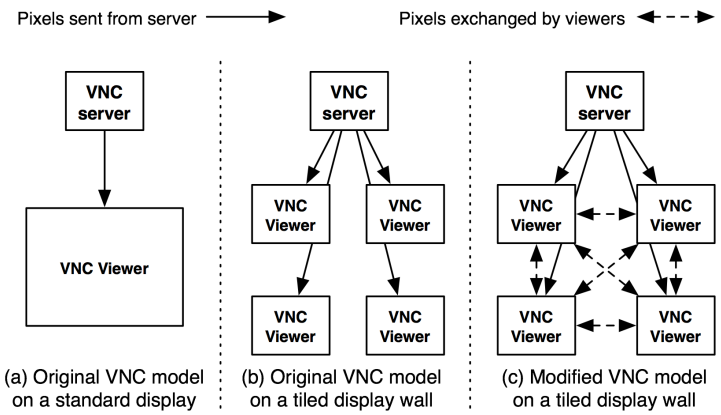

Figure 2: The original VNC model for (a) a standard display, (b) a tiled, 2x2 display wall. In the modified model (c), the viewers exchange pixels with each other in addition to receiving pixels from the server.

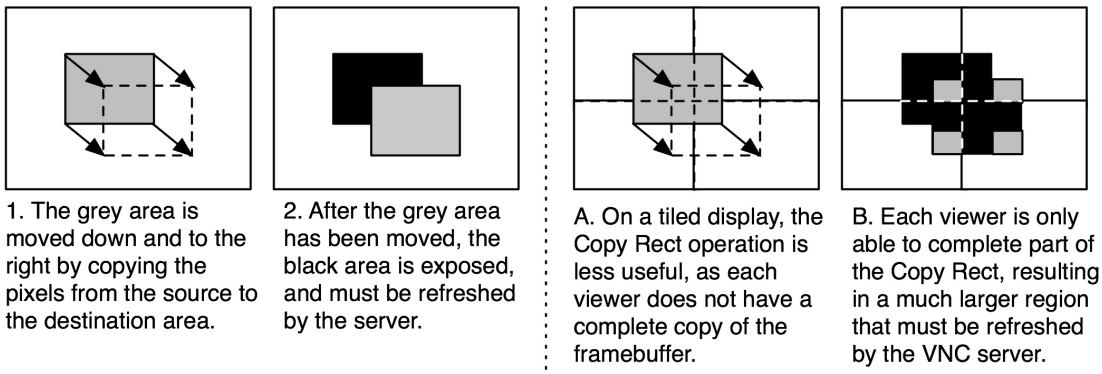

Figure 3: The Copy Rect operation as it is used for a single viewer for the entire remote desktop, and its behaviour when used with multiple viewers each showing a region of the server's desktop on a tiled, high-resolution display.

The Copy Rect operation is used whenever an area of the screen is moved, but the pixels inside the area remain unchanged, shown in Figure 3 (1) and (2). This is common when moving windows, scrolling in documents or panning images. Since a Copy Rect only takes 12 bytes to send regardless of the size of the area being updated, the Copy Rect

operation is important for reducing the server's bandwidth usage. To make the best use of it, the viewer must have access to all of the pixels being moved for the entire desktop. On a tiled display, this is not the case, as each viewer only has the pixels covering its own area of the display. Copy Rect operations that span the areas of more than one viewer force the server to split the operation, resulting in a larger set of exposed areas, shown in Figure 3 (A) and (B). This incurs additional load on the server, which DVNC alleviates by letting the viewers themselves exchange the necessary data. Figure 4 illustrates this.

In the de-centralized VNC model, the viewers receive updates not only from the server, but also from each other. This creates consistency issues, both for the viewer's own display, and for pixels sent by the viewer to other viewers. For instance, a viewer receiving one update from the server and a second update from a different viewer, needs to know which of the two updates to apply first in order to ensure a consistent display. In DVNC, the consistency issues are solved by imposing a total ordering on all updates sent by the server to the viewers, and by ensuring that all the viewers see the same Copy Rect operations.

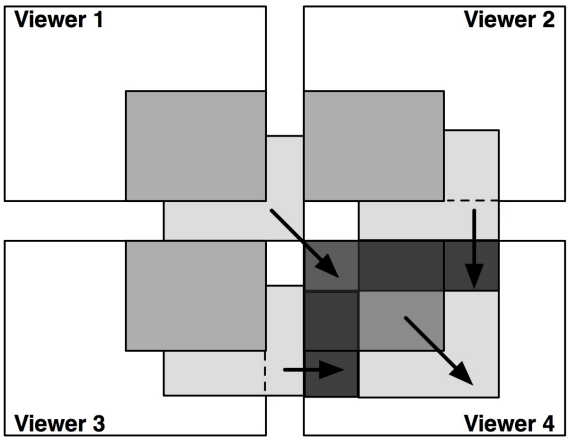

Figure 4: A Copy Rect operation spanning four viewers. The darkened, grey area moves down and to the right. Viewers 1, 2 and 3 transfer some of their pixels to viewer 4. (Other pixel transfers are not shown, such as from viewer 1 to viewer 2.) The arrows indicate direction of movement.

The viewers make independent decisions about where to send pixel data based on the Copy Rect operations they receive. Pixel data is always pushed to other viewers. To avoid circular dependencies between different viewers, the Copy Rect operation is split into two phases: A Copy Rect pre-phase, and a Copy Rect post-phase. During the pre-phase, the viewer determines which viewers it should send pixels to, copying and sending data as necessary. During the post-phase, a viewer applies updates from other viewers in the correct order.

4 Implementation

DVNC was implemented by modifying RealVNC's free VNC distribution (available under the GPL license), version 4. Both the VNC server and the VNC viewer required modifications. There are many implementations of VNC, including TightVNC, UltraVNC, and others. Since the modifications involve changing the model, they could also have been implemented by modifying a different VNC implementation.

VNC server modifications

The VNC server was modified to ensure that all connected viewers see the same Copy Rect operations in the same order. Instead of accumulating updates for each viewer, the server accumulates the same set of updates for all viewers, sending them once all the viewers have requested an update. The drawback to this approach is that the server can provide updates no faster than the slowest viewer. When updates are sent, the Copy Rect operation's rectangle is no longer clipped to the area which the viewer requests from the server, but sent regardless of whether the Copy Rect actually intersects with the viewer's

area. The server continues to clip Fill and Image Rect operations to the area requested by the viewer. A 4-byte, logical timestamp was added to the RFB protocol's framebuffer start message. The logical timestamp is incremented once for each group of update operations, and is used by the viewers to match updates received from other viewers to the correct Copy Rect operation. Finally, the server was modified to measure its load during the various experiments.

VNC viewer modifications

The VNC viewer was modified to receive framebuffer updates from other viewers. A separate thread is responsible for sending and receiving pixel data to and from other viewers. This thread also handles the logic necessary to determine which pixels should be sent and received, as well as the order in which updates are applied. Also, both the original and modified viewers were changed to record various statistics used for the experiments.

To better overlap communication with computation, incoming update operations from the server are queued. If the operation to be queued is a Copy Rect, its pre-phase is executed before queueing it (in some cases, execution of the pre-phase may be delayed to ensure consistency). The pre-phase copies data and sends it to other viewers, increasing the chance that other viewers will have the data they need when they begin executing the Copy Rect's post-phase.

When the server signals that it is done sending updates, all the queued operations are applied by the viewer. Applying a Copy Rect operation is done by executing its post-phase. During the post-phase, the viewer scans its list of updates received from other viewers, matching them to the current Copy Rect using the VNC server timestamp and other data contained by the operation. If the viewer hasn't received all the necessary data from other viewers, it will block waiting for the remaining data to arrive.

The queueing strategy introduces a queueing overhead not present in the original implementation. To minimize this overhead, the modified viewer avoids queueing when possible. If the rectangle covered by the incoming operation does not overlap with the rectangles of any queued operations, the operation can be applied immediately.

Determining where a viewer sends its pixels for a given Copy Rect operation is done by examining the data given by each Copy Rect operation. A Copy Rect operation consists of a source rectangle R=(x, y, width, height) and a delta point (dx, dy). The delta point indicates where the pixels identified by the source rectangle should be moved, yielding a destination rectangle. The viewer intersects the source rectangle with its own area. If the intersection is non-empty, the destination rectangle is computed by offsetting the clipped source rectangle by the operation's delta point and intersecting the result with the viewer's area. If the source and destination rectangles have different sizes (indicating that part of the destination rectangle falls outside the viewer's area), the viewer will transmit some of its data to other viewers.

When the viewer starts up, it is given the area of the VNC desktop that it should display as part of its arguments. The viewer then connects to the server and to the all other viewers by means of a multicast discovery mechanism. When a connection to another viewer is established, the viewers exchange a handshake, before they can exchange framebuffer updates. The handshake consists of five long integers: A magic number followed by the area the viewer covers. All future messages consist of a four-byte field containing the length of the message, followed by the actual message itself. These messages consist of the VNC server timestamp, the rectangle and delta point from the Copy Rect operation, followed by the pixels for the update. The pixels are currently not compressed.

5 Experiments

The performance of the original VNC and modified DVNC implementations is measured using three metrics: Total number of pixels refreshed, total number of bytes sent from the server to the viewers, and the server's CPU load. A high pixel refresh count is better than a low refresh count, as more pixels updated means better interactive performance. The DVNC implementation is also expected to reduce bandwidth used by the server, and reduce the server's CPU load. This is because the modified model is based on distributing load from the server to the viewers.

Hardware and software setup

The hardware used was (i) a display cluster with 28 nodes (Intel Pentium 4 EM64T, 3.2 GHz, 2 GB RAM, HyperThreading enabled, running the Rocks cluster distribution 4.0) connected to 28 projectors (1024x768, arranged in a 7x4 matrix), (ii) switched, gigabit Ethernet, (iii) a dual Intel Xeon 3.8 GHz with 8 GB RAM, and (iv) another Pentium 4 (same hardware as the nodes in the display cluster). The Xeon and the last Pentium 4 were used to run the server, and ran RedHat Enterprise Linux 4. The image viewer used was "xloadimage" by Jim Frost. The event generator for the control experiments used the XTestExtension to post input events to the server, and was custom-made for these experiments. The VNC distribution was RealVNC version 4 [4], exporting a 16-bit desktop.

Server and viewer instrumentation

The original and modified servers were instrumented to record their CPU load over the duration of an experiment, recording both time spent at user level, and time spent on behalf of the servers at kernel level. The servers recorded 10 samples per second, sending performance data to a second computer on the same local network. The additional network traffic generated by sending performance data is negligible at less than 500 bytes per second.

The original and modified viewers were instrumented to record the statistics outlined in Table 1. Each viewer makes its own measurements. At the end of each experiment, the number of pixels refreshed and number of bytes exchanged is summed. The queueing overhead's global maximum and minimum values are determined, and the global queueing overhead average is calculated by averaging the averages from each viewer. The total number of bytes sent between the viewers was also recorded, but these data have not been used to characterize performance in this paper.

Name	Description
Total Pixels	Total number of pixels refreshed by this viewer.
Server Bytes	Total number of bytes received by this viewer from the server.
Viewer-to-viewer Bytes	Total number of bytes received by this viewer from other viewers.
Queuing Overhead	Minimum, maximum and average overhead caused by queueing incoming operations.

Table 1: Statistics gathered from the viewers.

Experiments and methodology

Two sets of trace experiments and a set of control experiments were conducted. The trace experiments aim at measuring the performance for a user interacting with the desktop. In particular, the answers to the following four questions were of interest: (i) How many

more pixels can the DVNC implementation refresh compared to VNC? (ii) How much bandwidth does DVNC save? (iii) How does the DVNC changes affect the server's load? (iv) How big is the queuing overhead? The trace experiments play back two recorded user traces, where a user either pans an image or moves a window (see Table 2). In the first set of trace experiments, the server ran on the Xeon, and in the second set, the server ran on the Pentium 4.

Trace	Description
Image pan	A user pans an image sized at 9372x9372 pixels. The visible portion of the image covers almost the entire display wall, the rest of which is covered by the image viewer's window decorations. The trace lasts for 255 seconds.
Window move	A window sized at 2592x1944 pixels is moved around on screen. The trace lasts for 145 seconds.

Table 2: The traces used for measuring performance.

The control experiments have two purposes: (i) Get an objective view of the system's performance, and (ii) measure the maximum performance gain in a situation where the server's possibility for using Copy Rect operations is near maximized. The image from the Image pan trace is moved vertically up and down in a controlled manner. The rate at which movement occurs is varied for each experiment, ranging from one to fifty times per second, with each movement scrolling the picture 8 pixels up or down. An event generator is used to move the image at the constant rate defined by each experiment, with each experiment lasting 30 seconds.

Where the trace experiments measure the system's performance in a setting similar to real-world use, the control experiments allow for external repeatability. Before running either trace or control experiments, the server was restarted, and its desktop configured to match the experiment's starting point (open windows and window positions on the desktop). Then the viewers were restarted, and the experiment was conducted, before performance data was gathered.

A null-benchmark measured the overhead incurred by the changes to the VNC protocol. The server displayed a static image, and the number of bytes required to refresh the viewers was measured. The original sent a total of 85688.97 KB, while the modified sent 85707.69 KB - an overhead of 0.02%.

Trace results

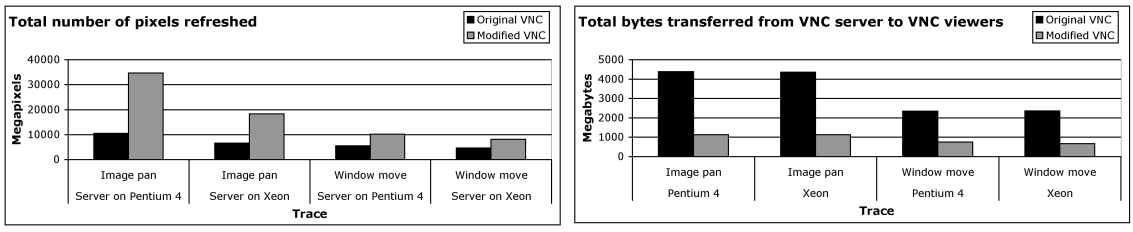

Figure 5: Left: Total number of pixels refreshed for each trace by the original and modified VNC viewers. Right: Total number of bytes sent by the server to the viewers.

Figure 5 shows the total number of pixels refreshed by the original and modified viewers as well as bytes sent from the server to the viewers for each trace. With the server on the Pentium 4, the modified implementation refreshes 34.6 gigapixels (GPx) for the Image pan trace, 3.29 times more than the original's 10.5 GPx. For the Window move

trace, the modified implementation refreshes 1.83 as many pixels. The number of bytes sent is reduced by 74% for the Image pan trace, and by 68% for the Window move trace. On the Xeon, number of pixels refreshed increases from 6.5 to 18.2 GPx and 4.6 to 8.1 GPx for the two traces respectively. Interestingly, the Pentium 4 is able to refresh almost twice as many pixels as the Xeon for the Image pan trace. The amount of data transferred is approximately the same regardless of where the server runs.

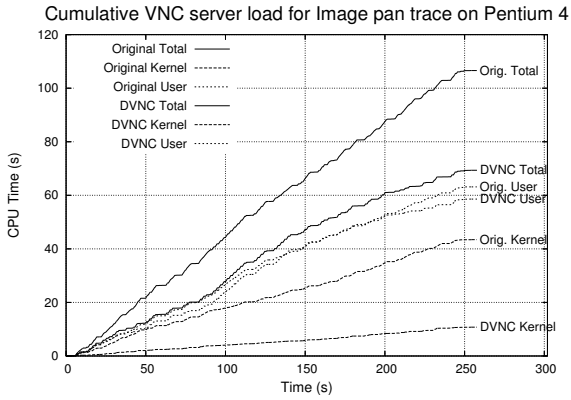

Figure 6: Cumulative server CPU load for the Image pan trace on Pentium 4, with total, user and kernel level load for both implementations.

Figure 6 shows the original and modified servers' cumulative CPU load, measured in seconds, when running on the Pentium 4 for the Image pan trace. The X axis shows the running time of the trace, and the Y axis shows the CPU time consumed by the server. The DVNC server's load is reduced by 35% compared to the original VNC server (from 106.6 to 69.3 CPU seconds). The biggest reduction happens at kernel level, where the load is reduced by 75%, while the difference in user level load is only 4%. The reduction in kernel level load correlates well with the reduction in bandwidth used by the DVNC server. For the window move trace, the reduction in CPU load is 19.7% (from 64.8 to 52.0 CPU seconds). The server load on the Xeon has similar characteristics.

Table 3 shows the modified implementation's queuing overhead. The maximum queuing overhead is 0.56 seconds, which means that an update operation received by one of the viewers was queued for a little over half a second before it was actually drawn. The average queuing overhead is between 0.008 and 0.011 seconds, and the minimum overhead is 0.000 seconds.

Trace	**Min**	**Avg**	**Max**
Image pan *(P4)*	0.000 s	0.009 s	0.546 s
Window move *(P4)*	0.000 s	0.008 s	0.467 s
Image pan *(Xeon)*	0.000 s	0.011 s	0.582 s
Window move *(Xeon)*	0.000 s	0.011 s	0.504 s

Table 3: The queueing overhead, measured in seconds, for the traces on the Pentium 4 and the Xeon.

Control experiment results

Figure 7 shows the number of pixels refreshed by the viewers for the control experiment. The measured values are compared to a target number of pixels that should have been refreshed if sufficient resources to avoid all bottlenecks were available. The target value is calculated by measuring the number of pixels refreshed when scrolling the image vertically by 8 pixels, and multiplying that number with the duration of each experiment and rate at which the image is moved.

The number of pixels refreshed increases linearly with the event generation rate. At first, both implementations closely follow the target refresh count. The original implementation reaches its maximum at an event rate of 26, while the modified implementation keeps tracking the target up to 40 events per second. The original's performance goes down by 57.8% when the event generation rate is increased from 26 to 28. The DVNC performance goes down by only 6.6% when increasing the event generation rate from 40 to 45. At an event rate of 50, DVNC refreshes 11.9 times as

many pixels as the original.

Figure 8 shows the server's total, kernel and user level load in percent for both implementations. Initially, the CPU load increases linearly for the implementations, with the original's load increasing almost twice as fast as the modified's load. At the peak in load, close to 100%, the event rate for the original and modified server is respectively 26 and 40. This is also the rate at which the two implementations peak in number of pixels refreshed.

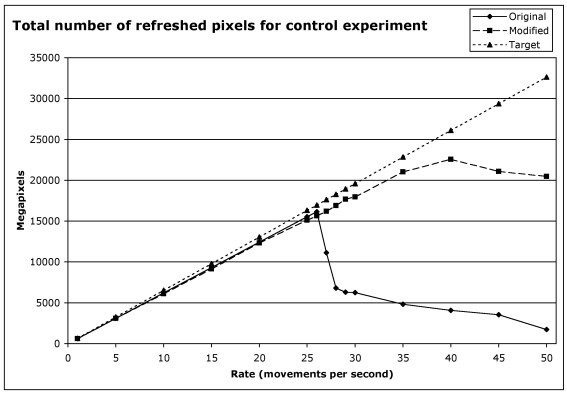

Figure 7: Total number of pixels refreshed for the control experiment for the two implementations, as well as the target refresh count. Event generation rates range from 1 to 50.

Figure 9 shows the total number of bytes transferred from the server to the viewers. The number of bytes transferred increases linearly with the event rate, with a slower growth for the modified implementation. The original implementation peaks at 1135 MB, while the modified implementation peaks at 375 MB. This corresponds to a bandwidth use of 37.8 MB/s and 12.5 MB/s, respectively, neither of which is close to the maximum transfer rate of gigabit Ethernet at about 90 MB/s. Interestingly, the bandwidth used by the original implementation continues to climb even after having peaked both in CPU load and number of pixels refreshed.

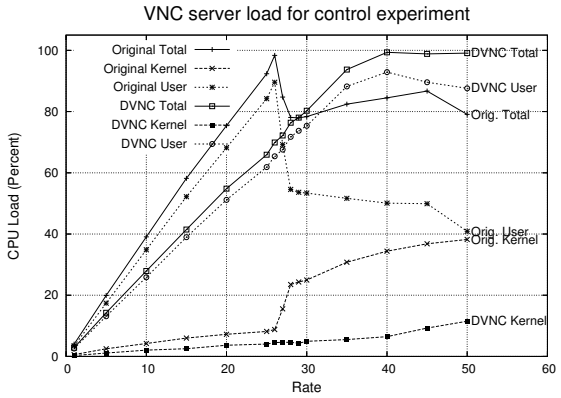

Figure 8: CPU load for the VNC server, showing total, kernel, and user level load for both the original and modified implementations in the control experiment.

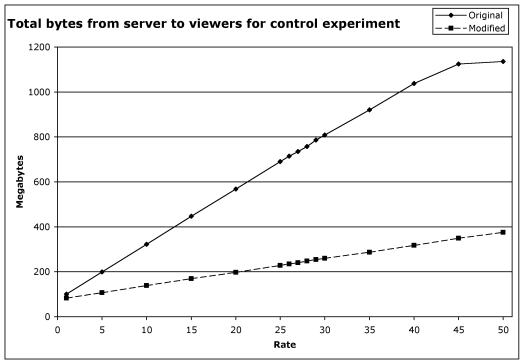

Figure 9: Total bytes sent from the servers for the control experiment.

6 Discussion

The results from the trace experiments show that the DVNC implementation can refresh more than three times as many pixels compared to the original. The control experiment documents that DVNC can outperform the original by a factor of up to 11.9. For the case where the server sends no Copy Rect operations - and hence no gain can be expected from delegating work to the viewers - DVNC adds very little overhead; only 0.02% in the null-benchmark. DVNC provides no performance benefit for content that is updated using

operations other than Copy Rect - typically video, animated or otherwise “fresh” content. DVNC provides a significant performance increase for certain operations that the server is able to translate into Copy Rect operations.

Trace experiments

The server spends less CPU time at kernel level since it sends less data, leaving more resources for the server and other applications. The server’s user level load is not reduced as much, since the server provides viewers with more frequent updates, while sending less data.

The maximum queueing overhead was half a second, and can be observed as occasional stutters during playback of the traces. Even though there is some queueing overhead associated with keeping each viewer consistent, overall performance is still much better than the original implementation.

The staircase effect in Figure 6 is caused by periods of lower user activity. When the user is not moving the image or the window, fewer updates take place and the server experiences low load. Typically, this occurs when the user repositions the cursor to drag the image or move the window. This effect is not present in graphs depicting the CPU load for the control experiments (not included in this paper).

Control experiments

DVNC’s performance compares even more favorably to the original in the control experiments, than it did in the trace experiments. The reason for this is that the pay-off from each Copy Rect generated in the control experiments is greater than it is in the trace experiments. In the trace experiments, many Copy Rects move diagonally. Diagonal movements cause the areas covered to be smaller, meaning that larger areas must be refreshed by the server. New pixels must be sent by the server to refresh not only the top or bottom edge, but also the right or left edge of a given area. Diagonal movements also cause more complicated dependencies between the different viewers when they exchange pixels. A vertical or horizontal Copy Rect operation only requires that a viewer sends its pixels to one other viewer, while a diagonal Copy Rect can require a viewer to send pixels to three different viewers.

The original implementation’s sudden drop in pixel refresh count (Figure 7) is not caused by lack of network bandwidth, as the bandwidth used continues to increase even after the drop in refreshed pixels (Figure 9). The drop is caused by the server having to work harder to keep its own framebuffer updated, which delays updates to the viewers. The delay makes each viewer accumulate a larger dirty region, requiring more bytes to refresh. This behaviour also explains why the original starts spending more time at kernel level when the drop in performance occurs, as the kernel is heavily involved in the communication.

Server on Pentium 4 and Xeon

The performance measured by the trace experiments on the Pentium 4 and on the Xeon were not as expected. The Pentium 4, with its older CPU architecture, performed better than the newer Xeon. The server implements Copy Rect by moving memory from one location to a different location in the server’s framebuffer. To investigate whether memory bus speeds were the issue, the two computers’ processor-memory bandwidth was measured using CacheBench [13]. The sustained read/modify/write bandwidth to

memory for the Pentium 4 was 3.78 GB/s, while the Xeon only managed 2.16 GB/s. This is a factor of 1.75, which correlates well with the difference in refreshed pixels, 34 GPx vs. 18 GPx, a factor of 1.88.

Lessons learned

The performance improvements achieved by DVNC is made possible by changing the model at a number of different levels. From a model where the server does all work and the viewers are passive receivers, the new model makes the viewers partially serve each other, off-loading the server. The viewers, which previously needed no knowledge about other viewers, now need to know about every other viewer in order to exchange pixels with them. Each viewer makes its own decisions about where to send pixels, as opposed to having the server handle this task.

Discovering that the server's memory bandwidth is a bottleneck was surprising, given that the pixels moments later must be moved over a "slow" gigabit Ethernet. This is becase the server may have to move up to 80 MB of data before sending a Copy Rect operation describing the movement, while the A Copy Rect operation itself only requires 12 bytes to transfer.

7 Conclusion

This paper has presented a modification to the VNC model that improves performance when VNC is used to create the desktop environment for tiled display walls. The Decentralized VNC (DVNC) system increases performance for tasks like navigating large images and moving windows on the desktop. The main principle employed is to let the VNC viewers exchange data amongst each other, freeing the VNC server from re-sending already distributed pixel data.

The DVNC model has been implemented by modifying an open-source VNC implementation, and its performance evaluated. A tiled 7x4 display wall with a total resolution of 7168x3072 pixels was used for the experiments. The system's performance was measured through several user trace and control experiments, and compared to VNC without modifications. The results show that end-user performance was significantly improved. For panning large images, DVNC could refresh three to twelve times more pixels on the display wall compared to the original implementation. These improvements are expected to carry over to other cases where screen content moves, but otherwise remains unchanged, such as scrolling in documents.

The performance improvements are a result of distributing work between the server and viewers. This lets server-side processing overlap with viewer-side pixel distribution. In addition, the bandwidth required for the server to keep the viewers updated is reduced. Consequently, the server can spend more cycles keeping its framebuffer updated, as well as leaving more cycles for other applications.

8 Acknowledgements

The authors wish Lars A. Bongo and Espen S. Johnsen. This work has been supported by the Norwegian Research Council, projects No. 159936/V30, SHARE - A Distributed Shared Virtual Desktop for Simple, Scalable and Robust Resource Sharing across Computer, Storage and Display Devices, and No. 155550/420 - Display Wall with Compute Cluster.

References

[1] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. Virtual Network Computing. *IEEE Internet Computing*, 2(1):33–38, 1998.

[2] Kai Li, Han Chen, Yuqun Chen, Douglas W. Clark, Perry Cook, Stefanos Damianakis, Georg Essl, Adam Finkelstein, Thomas Funkhouser, Timothy Housel, Allison Klein, Zhiyan Liu, Emil Praun, Rudrajit Samanta, Ben Shedd, Jaswinder Pal Singh, George Tzanetakis, and Jiannan Zheng. Building and Using A Scalable Display Wall System. *IEEE Comput. Graph. Appl.*, 20(4):29–37, 2000.

[3] Bram Stolk and Paul Wielinga. Building a 100 Mpixel graphics device for the OptIPuter. *Future Gener. Comput. Syst.*, 22(8):972–975, 2006.

[4] RealVNC, Ltd. VNC for Unix 4.0. http://www.realvnc.com/.

[5] Lars Ailo Bongo, Grant Wallace, Tore Larsen, Kai Li, and Olga Troyanskaya. Systems support for remote visualization of genomics applications over wide area networks. In *Proc. of GCCB'06. LNBI 4360*, 2006.

[6] Tony Lin, Pengwei Hao, Chao Xu, and Ju-Fu Feng. Hybrid image coding for real-time computer screen video transmission. Januar 2004. Visual Communications and Image Processing (VCIP) 2004, part of the IS&T/SPIE Symposium on Electronic Imaging 2004.

[7] dcommander. VirtualGL. http://www.virtualgl.org.

[8] Ricardo A. Baratto, Leonard N. Kim, and Jason Nieh. THINC: a virtual display architecture for thin-client computing. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 277–290, New York, NY, USA, 2005. ACM Press.

[9] Robert W. Scheifler and Jim Gettys. The X window system. *ACM Trans. Graph.*, 5(2):79–109, 1986.

[10] R. E. Faith and K. E. Martin. Xdmx: Distributed, multi-head X. http://dmx.sourceforge.net/.

[11] Byungil Jeong, Luc Renambot, Ratko Jagodic, Rajvikram Singh, Julieta Aguilera, Andrew Johnson, and Jason Leigh. High-Performance Dynamic Graphics Streaming for Scalable Adaptive Graphics Environment. SuperComputing 2006, 11.-17. November 2006.

[12] Tristan Richardson. The RFB Protocol, version 3.8.

[13] Philip J. Mucci. Low-level characterization benchmarks. Available from http://icl.cs.utk.edu/projects/llcbench/index.html.