Experiencias al Implementar un Display Distribuido para el Servidor X Window

Experiences at the Implementation of a Distributed Display for the XWindow Server

Ana Isabel Rodríguez, Jesús Alberto Verduzco, Nicandro Farias, Juan García, Noel García
Instituto Tecnológico de Colima, Av. Tecnológico No. 1 Colima, México. C.P. 28976
kidoki42@hotmail.com, javrtesis@yahoo.fr, nmendoza@ucol.mx, juan_garcia@ucol.mx, garcian@ucol.mx

Resumen

La necesidad de disponer de superficies de visualización gráfica de mayores dimensiones y resolución que aquellas proporcionadas por el monitor de la PC se hace evidente al utilizar aplicaciones que generan cantidades significativas de datos gráficos como: la visualización científica, los entornos de la realidad virtual, el diseño en ingeniería, el análisis de gráficas en finanzas, etc. Este documento describe las experiencias obtenidas al implementar un display distribuido para el servidor X Window, que permite la distribución de las gráficas de una aplicación hacia un conjunto de nodos los cuales cooperan para visualizar una imagen única, de grandes dimensiones y de alta resolución.

Abstract

The need to have surfaces of graphical visualization of greater dimensions and resolution than those provided by the monitor of the PC becomes evident with applications that generate significant quantities of graphical information, like: scientific visualization, virtual reality environments, engineering design, graphics for financial analysis, etc. This document describes the experiences obtained by implementing a distributed display for the X Window Server that allows the distribution of an application graphics results towards a set of nodes which cooperate to display a unique image of big dimensions and high resolution.

1. Introducción

1.1 Visualización de Datos

El monitor es el dispositivo normalmente utilizado para mostrar información al usuario de una computadora. Sin embargo, la tecnología del monitor ha evolucionado lentamente en comparación con otros componentes de la PC. Los dispositivos de la PC como las memorias, el procesador y los discos han experimentado un desarrollo constante y acelerado.

La resolución y dimensiones del monitor constituyen un obstáculo para cierto tipo de aplicaciones que tienen como característica principal la generación de importantes cantidades de datos gráficos. La visualización científica de datos, los entornos de la realidad virtual, el diseño en ingeniería y el análisis de gráficas en finanzas, son ejemplos de aplicaciones que requieren superficies de visualización gráfica de mayores dimensiones que aquellas proporcionadas por el monitor. Ejecutándose en un sistema con reducida potencia gráfica, por ejemplo el monitor de la PC, una aplicación de este tipo no puede explotar toda su capacidad en la generación de dicho entorno gráfico. Como consecuencia, la visualización de gráficas complejas y el proceso de interacción con el usuario son fuertemente perturbados. El interés por aumentar las dimensiones y resolución de la superficie de visualización ha estado presente en estos años recientes. Las soluciones actuales utilizan una arquitectura centralizada o una distribuida. El enfoque centralizado consiste en dotar a un sólo nodo de múltiples tarjetas gráficas y con ello alimentar más de un monitor. Como ventaja se dispone de una alternativa de bajo costo y de fácil implementación. Sin embargo este esquema sufre de algunos inconvenientes. Un solo nodo efectúa el proceso de renderización, lo que implica baja eficiencia. Un problema mayor es el número limitado de puertos PCI (del inglés Peripheral Component Interconnect), AGP (del inglés Advances Graphics Port) o PCI-Express disponibles en la PC. Una solución más atractiva es distribuir el proceso gráfico entre un grupo de nodos (llamados nodos de renderización) interconectados por una red de transmisión de datos y distribuir las imágenes y las primitivas utilizando un

protocolo adecuado. En este modelo, los problemas principales son el consumo excesivo del ancho de banda de la red de transmisión de datos y la sincronización de las actividades de los nodos de renderización. A pesar de estos inconvenientes, actualmente los proyectos en desarrollo emplean como solución la distribución del proceso de renderización.

La mayor parte de los esfuerzos han sido dedicados al desarrollo de soluciones para permitir la ejecución de aplicaciones que generan escenarios 3D utilizando muro de imágenes [14, 15,16].

Pocos esfuerzos han sido dirigidos a desarrollar soluciones para permitir la ejecución de los clásicos entornos 2D en muros de imágenes. Dos proyectos, VNCwall [17] y Xdmx [18] proponen cada uno su propio método de solución.

VNCwall es una extensión del software llamado VNC (Virtual Network Computing) [19]. VNC permite mostrar las gráficas de una aplicación en una máquina remota. Para este fin, la aplicación envía comandos gráficos a un servidor VNC que renderiza las gráficas en un conjunto de localidades de memoria llamado *shadow frame buffer*. Posteriormente un protocolo orientado a la transmisión de píxeles, envía el conjunto de píxeles para su visualizacion a un VNCviewer que se ejecuta en una máquina remota. VNCwall extiende este principio, renderizando la imagen, dividiéndola en tiles y distribuyendola a un conjunto de nodos de un muro de imágenes. El proceso de renderización es concentrado en un solo nodo (el del servidor VNC) y la cantidad de datos (píxeles) que circulan por la red es importante por lo que este esquema sufre de pobre desempeño.

Por otro lado, Xdmx esta basado en el sistema X Window y dos extensiones Xnest y Xinerama. Un servidor (front end) aparece para una aplicación como un clásico servidor X Window, cuya función es ocultar los nodos del muro de imágenes (backend nodes). Aunque es un sistema completamente funcional y en constante mejoramiento, su desempeño depende de Xinerama. En Xinerama muchas de las operaciones gráficas son realizadas mediante copiar y pegar. Esto genera problemas de rendimiento con aplicaciones que generan gráficas intensivamente.

El objetivo de nuestro proyecto, es desarrollar una solución que soporte la distribución de la renderización utilizando dispositivos de bajo costo como las PC tradicionales, las redes de transmisión de datos y software de libre distribución como Linux y X Windows. Nuestra solución difiere de otros trabajos, en que utiliza un enfoque diferente para procesar y distribuir las primitivas gráficas que aquel empleado por Xdmx y VNCwall. Nuestro software es ligero y presenta buen rendimiento.

Este documento está organizado de la siguiente manera, en la sección dos describimos muro de imágenes, el sistema de ventanas X Window y el concepto de servidor proxy. En la sección tres describimos nuestra solución. En la sección cuatro analizamos los resultados del desempeño obtenidos al aplicar diferentes pruebas. Finalmente en la sección cinco establecemos nuestras conclusiones y se dan algunas pistas para la continuación de este trabajo.

2. Trabajos Relacionados

Esta sección introduce los conceptos generales utilizados a lo largo de este documento.

2.1. Muro de Imágenes

Actualmente es posible construir un sistema de cómputo de gran capacidad y de un costo moderado asociando un conjunto de computadoras personales PC y comunicándolas por medio de una red de transmisión de datos de alto desempeño. Este sistema conocido como Cluster dispone de numerosos recursos distribuidos como los procesadores, las memorias, los discos y las tarjetas gráficas. Una explotación simple y eficaz de estos recursos requiere de un software que los administre como un todo, ocultando al usuario su carácter distribuido. El objetivo es de poder utilizar un Cluster como una estación de trabajo de gran capacidad en almacenamiento, en procesamiento y en visualización gráfica. Los sistemas de archivos como NFSP (del inglés Network File System Parallel) permiten globalizar los recursos de almacenamiento. *Software* como Mosix y Condor ofrecen una imagen única de los recursos de CPU y memoria. Sin embargo existen pocas soluciones que ofrezcan una vista global del conjunto de recursos gráficos disponibles en los nodos de un Cluster. Para satisfacer estas necesidades se ha construido el llamado Muro de Imágenes (por su significado en inglés *Display Wall).* Un Muro de Imágenes es un sistema capaz de visualizar imágenes en alta resolución sobre grandes superficies [1]. La imagen global es generada sobre una o varias superficies por un conjunto de videoproyectores, los cuales son alimentados por las salidas gráficas de los nodos que constituyen un Cluster. En la figura 1 se muestra un Muro de Imágenes compuesto por cuatro superficies independientes (nombradas *tile*). Cada nodo del Cluster es responsable de generar la imagen correspondiente a cada *tile* y la red de transmisión de datos traslada primitivas gráficas y comandos de sincronización entre ellos. En el Cluster existen uno o más nodos que permiten la interacción del usuario con las aplicaciones, ya sea utilizando el teclado y el ratón o por otros medios como el reconocimiento de gestos, el rastreo de la

posición del usuario, etc. Actualmente existen proyectos en funcionamiento, como Infinite Wall [2], The Office of the future [3], The Information Mural [4], The Power Wall [5]. Tradicionalmente para aplicaciones de visualización gráfica de datos se han utilizado como elementos fuente de cálculo a supercomputadoras de la clase SGI Onyx. La rápida evolución e incremento del desempeño y tendencia al bajo costo de los componentes de las PC y las redes de transmisión de datos, elementos esenciales de un Cluster, lo convierten en una alternativa viable para aplicaciones de procesamiento intensivo de datos. La utilización de Cluster como fuente de cálculo de sistemas de visualización gráfica es cada vez más común y en algunas implementaciones se han logrado obtener desempeños similares a aquellos de las supercomputadoras [6].

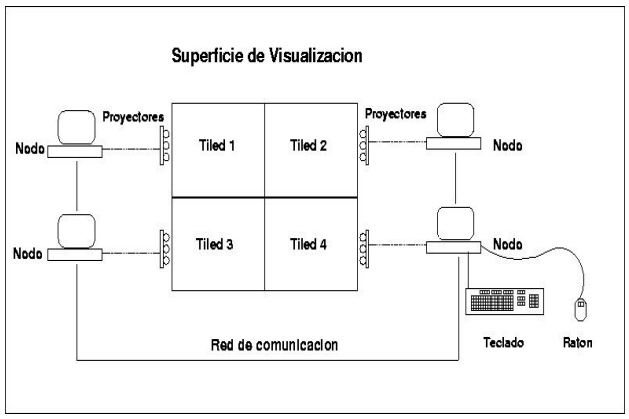

Fig. 1. Arquitectura de un Muro de Imágenes.

2.2. El sistema de ventanas X Window

El sistema de ventanas X Windows (comúnmente llamado servidor X) es un servidor de ventanas que construye el entorno gráfico que las aplicaciones despliegan en el monitor y gestiona la interacción del usuario por medio del control del teclado y el ratón. El Consorcio X [7] mantiene el código base del servidor X y una gran comunidad de desarrolladores trabajan en la implementación de rutinas con funcionalidades específicas para soportar configuraciones especiales en *software* y *hardware*. Funcionalmente el servidor X utiliza el modelo cliente-servidor. El servidor actúa como un intermediario entre el cliente, los recursos gráficos y los dispositivos de entrada/salida de datos (básicamente teclado y ratón) de la computadora en la que está en ejecución. Las tareas del servidor son controlar la terminal y sus periféricos y el de construir y visualizar las gráficas y los textos. La construcción del entorno gráfico la efectúa el servidor al procesar (proceso llamado renderización) las *primitivas* (mensajes X11) enviadas por el cliente. Las interacciones del usuario vía el ratón y el teclado son recibidas en forma de *eventos* por el servidor y enviadas hacia el cliente quien las procesa y reenvía el resultado al servidor. La aplicación del usuario representa la parte cliente quien envía primitivas hacia el servidor que contienen *comandos gráficos*. Para este propósito el cliente hace invocaciones a funciones de la librería Xlib [8], la cual contiene un grupo de funciones que ocultan los detalles de la construcción, encapsulamiento y transporte de los mensajes intercambiados entre el cliente y el servidor.

La comunicación entre el cliente y servidor es estandarizada por medio del protocolo X11[9]. Esta especificación establece un mecanismo por medio del cual un cliente puede comunicar la información necesaria para utilizar un sistema de ventanas a distancia sobre un canal bidireccional. El protocolo X define 3 tipos de mensajes:

- *Requests (llamadas aquí peticiones).* Es el único tipo de mensajes enviados del cliente al servidor. Contienen *comandos gráficos* que le indican al servidor la construcción y despliegue de gráficas en el monitor.
- *Eventos*. Describen las acciones del usuario sobre el teclado y el ratón.
- *Errores*. Indican excepciones del servidor durante el procesamiento de una *primitiva gráfica*.
- *Replys (*llamadas aquí *respuestas).* Son mensajes que le indican al cliente el éxito en el procesamiento de *comandos gráficos*

Las características gráficas de un servidor X están determinadas por el hardware específico sobre el cual se ejecuta y están resumidas en lo que comúnmente se conoce como *visual*. Una explicación más detallada sobre *visual* puede encontrarse en [10].

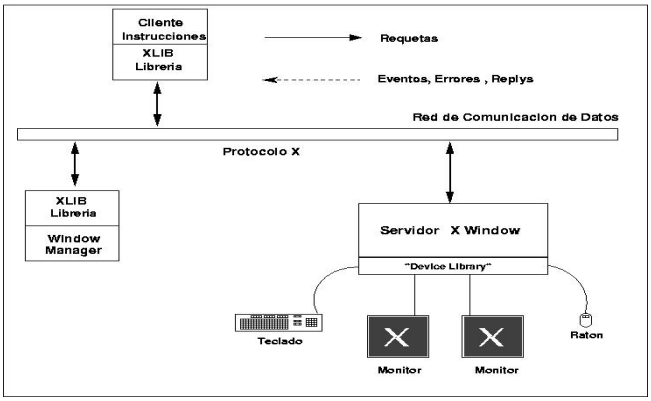

Fig. 2. Arquitectura del sistema de ventanas X Window.

Es común en el vocabulario del servidor X utilizar el término *display* para referirse a una instancia del servidor X en ejecución, un monitor, un teclado y un ratón. En la figura 2 se ilustra la arquitectura del sistema X Window.

El servidor X dispone de la propiedad de *clipping* que consiste en eliminar a nivel de renderización las partes de las gráficas que no son visibles en el monitor. Por ejemplo, si desplazamos una ventana hasta los límites del monitor, la mitad de la ventana será ocultada por efecto de las acciones de *clipping*.

Una de las características del servidor X que aprovechamos en nuestro proyecto, consiste en que el servidor X puede administrar más de un monitor de manera independiente.

2.3. Servidor proxy

Partiendo del modelo de funcionamiento del servidor X, un proxy puede ser insertado entre el cliente y el servidor. Los mensajes entre cliente y servidor pueden interceptarse, modificarse, eliminarse o redirigirse para un servidor X específico.

Los proxys han sido utilizados para agregar nuevas funcionalidades a las aplicaciones X. La ventaja principal de un proxy es que no exige la modificación y recompilación del código de los clientes X. Sin embargo, la interposición de un proxy entre cliente y servidor tiene un cierto número de efectos sobre el desempeño del sistema. El problema principal es que produce un retardo adicional en el tratamiento de cada mensaje intercambiado entre cliente y servidor. En ciertos casos, por ejemplo, cuando numerosos clientes activos son atendidos por un proxy ó si un cliente produce una gran cantidad de peticiones y de interacción con el usuario, el proxy puede convertirse en un *cuello de botella* afectando el rendimiento del sistema. La idea de un proxy para el servidor X no es nueva pues existen antecedentes en [11] e implementaciones como XMX [12].

3. El prototipo proXy

En esta sección nosotros describimos el trabajo desarrollado necesario para distribuir el display de un servidor X Window y visualizar las gráficas de una aplicación en los nodos que ejecutan una instancia del servidor X.

Los mensajes producidos por los clientes X necesitan ser modificados y distribuidos entre el grupo de servidores. Similarmente los mensajes de retorno, producto de la interacción del usuario deben de ser recuperados y dirigidos hacia la aplicación, como lo muestra la figura 3.

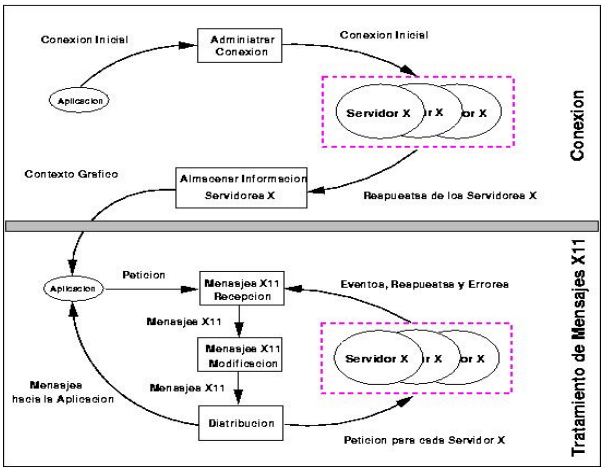

Fig. 3. Flujo de datos en el servidor proXy.

Analizando la funcionalidad y la disposición de los componentes de nuestra arquitectura, identificamos de manera específica dos flujos de datos:

Flujo de renderización: Se genera al momento en el que el cliente envía peticiones al proXy, quien las recibe, efectúa su tratamiento y las distribuye hacia los nodos de renderización que efectúan la renderización y visualización de las gráficas resultantes.

Flujo de interacción: Las acciones que el usuario ejecuta sobre el teclado y el ratón del servidor de interacción producen los *eventos* que serán enviados hacia el proXy que los recolecta y posteriormente los encamina hacia el cliente. El mismo procedimiento siguen los dos tipos de mensajes restantes, es decir los *errores* y las *respuestas* cuando sean generados los servidores de renderización.

En la figura 3 mostramos la arquitectura propuesta para nuestro sistema, la cual está conformada por los siguientes componentes básicos:

Nodo de proxy: Aloja una instancia en ejecución del proxy. A este nodo los clientes deben imperativamente conectarse, utilizando para ello el *socket* en el cual escucha el proxy.

Nodos de renderización: Reciben las *peticiones* enviadas por el cliente por medio del proxy y efectúan su renderización. Cada nodo efectúa la renderización de la fracción de la imagen que le corresponde visualizar. Con este procedimiento el conjunto de monitores o de videoproyectores, ordenados bajo un arreglo particular, presentan una sola superficie de visualización en la que se despliegan las imágenes de los clientes.

Nodo de interacción: Habilita los dispositivos de entrada de datos, teclado y ratón, mediante los cuales el usuario puede interactuar con la aplicación. Este nodo

también efectúa la renderización de *peticiones* y la visualización de gráficas.

3.1 Display virtual

La primera tarea de nuestro prototipo (al que nosotros bautizamos como proXy), es efectuada al momento de su lanzamiento, consiste en definir la configuración del *display virtual*. El usuario proporciona en la línea de comando la lista de nodos de renderización y la disposición que guardaran en el *display virtual*. El proXy envía *la petición* de apertura de display (XOpenDisplay) a cada servidor de renderización con la finalidad de recuperar sus características gráficas, particularmente las dimensiones del display. Con esta información, el proXy determina la resolución del *display virtual* (e.g., 4 servidores de renderización A,B,C y D cuyos displays son de *1280x960* píxeles y la disposición forma un arreglo de *2x2*, formarán un *display virtual* de *2560X1920* píxeles, como el que se muestra en la figura 4. Al momento en que un cliente X se conecta por primera vez al proXy, el proXy le envía las características del *display virtual*. Con esta información el cliente cree disponer de una superficie de visualización de mayores dimensiones que aquella del monitor de la PC en la cual se ejecuta.

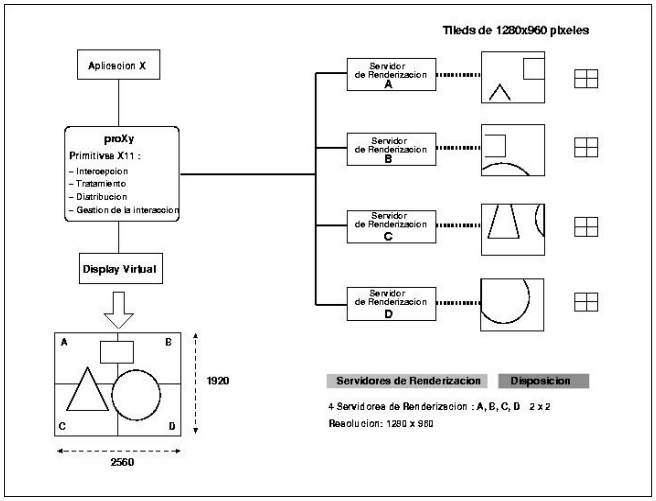

Fig. 4. Estructura del Display Virtual en proXy1

3.2. El flujo de renderización

Como lo expresamos en la sección 3, el *flujo de renderización* consiste en la modificación (tratamiento) y distribución de las *peticiones* recibidas por el proxy.

Nosotros desarrollamos dos estrategias para el tratamiento y la distribución de las *peticiones*. La primera estrategia es aplicada por el prototipo bautizado como proXy1 y consiste en la intercepción y adaptación de las coordenadas que transportan las *peticiones* con el fin de adaptarlas al contexto del *display virtual*. Esta solución se expresa por la formula siguiente:

$$X(u,v) = Xvirtual - U * X$$
$$Y(u,v) = Yvirtual - V * Y$$

Donde X, Y identifican la resolución del *display virtual* *(u,v)* la posición del servidor de renderización numerada a partir de (0,0) , (X*virtual*, Y*virtual*) las coordenadas de un punto en el *display virtual* y (X*(u,v)*, Y*(u,v)*) las coordenadas correspondientes con respecto al servidor de renderización *(u,v).* Gracias a las acciones de la función *clipping* los servidores de renderización procesan y visualizaran únicamente las partes de las gráficas que corresponden a la sección del display virtual que individualmente cada uno controla. La figura 4 muestra este procedimiento.

La segunda solución es utilizada por el prototipo llamado proXy2, esta consiste en términos generales de descentralizar las funciones de la adaptación de las *peticiones* hacia los servidores de renderización. Para este fin, proXy2 construye una ventana sobre el *display* de cada servidor de renderización. Las características especiales de esta ventana son que dispone de los atributos de *ventana raíz* y dimensiones similares a la del *display virtual.* Esta ventana es posicionada en cada servidor de renderización con el intervalo de coordenadas del *display virtual* que cada *servidor de renderización* es responsable de visualizar, el resto de la ventana creada en el servidor es eliminada por la función *clipping.* La figura 5 muestra este procedimiento.

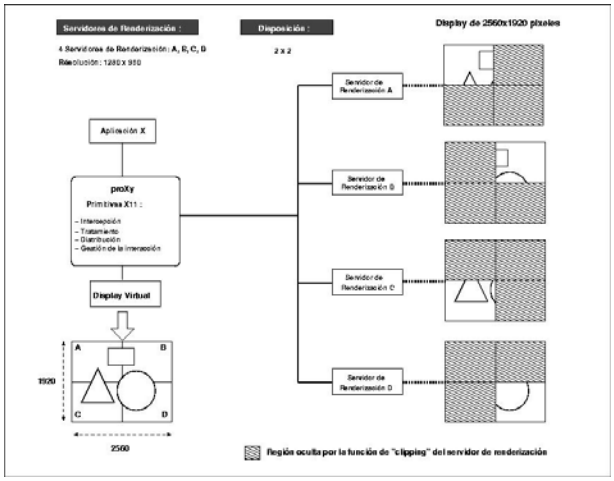

Fig. 5. Estructura del Display Virtual en proXy2

3.3. El flujo de interacción

Para recuperar las acciones del usuario sobre el ratón y el teclado, el proXy intercepta los mensajes

provenientes del servidor de interacción. El servidor X dispone del mecanismo de *focus* para determinar la ventana hacia la cual los resultados de la renderización son dirigidos. La asignación del *focus* a una ventana es determinada por la posición del cursor del ratón.

Mientras que el *focus* permanezca confinado al *tile* del servidor de interacción los eventos del teclado y del ratón son tratados localmente. La situación cambia cuando el usuario arrastra el cursor del ratón al *tile* vecino. Entonces los eventos generados en el servidor de interacción deben ser dirigidos hacia el servidor de renderización que posee el *focus* en ese momento. Nosotros analizamos el programa X2X que permite controlar con el teclado y el ratón un display remoto. X2X utiliza la extensión XTEXT presente en el servidor X Window que permite enviar seudoeventos hacia un display remoto. Nuestra solución esta basada en este código.

3.4. Resultados

Nuestro prototipo actualmente permite ejecutar aplicaciones del sistema X Window que despliegan gráficas en el muro de imágenes sin necesidad de modificaciones en el código fuente. Con respecto al flujo de interacción, nuestro sistema permite soportar un solo cliente activo a la vez, por lo que futuros desarrollos estarán encaminados a trabajar en este aspecto. La figura 6 muestra un ejemplo de la visualización de las gráficas del paquete Xfig del sistema operativo Linux.

Fig. 6. Ejemplo de funcionamiento del prototipo proXy

4. Desempeño

Para evaluar el desempeño de las dos estrategias implementadas, nosotros utilizamos el software *x11perf* [13]. Las pruebas de desempeño fueron realizadas utilizando 8 PC equipadas con un display de *1280x960* píxeles como nodos de renderización. En tanto que servidor de proXy utilizamos una PC con características similares a los nodos de renderización. Nosotros aplicamos los test: *XCreateWindow* que construye ventanas de *600x600 píxeles*, *XMoveWindow* que desplaza ventanas de *600x600 píxeles* y *XPutImage XY 500x500* que imprime un rectángulo de *600x600* píxeles.

Nosotros seleccionamos estos tres test, considerando que su visualización sobre el *display virtual* consume una cantidad significativa de tiempo de comunicación y de tiempo de tratamiento de las *peticiones* por parte del proXy. Decidimos en cada ejecución de los tests, aumentar el tamaño del *display virtual*, porque esto implica el incremento en el tiempo de tratamiento de cada *petición* por parte del proXy y la difusión de una cantidad mayor de mensajes hacia los *nodos de renderización* incrementando por consecuencia, el tiempo de comunicación.

Como se pueda apreciar en las gráficas de las figuras 7, 8 y 9 la solución proXy2 es netamente superior a la versión proXy1.

Esto demuestra que al distribuir el proceso del tratamiento de las *peticiones* hacia los nodos de renderización tiene efectos positivos en el desempeño del proceso del flujo de renderización.

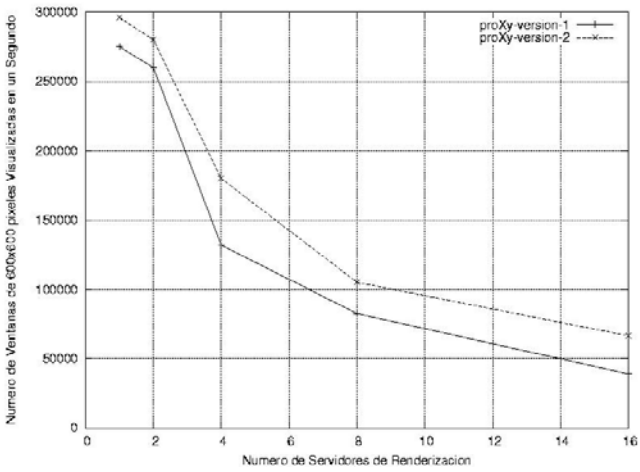

Fig. 7. Gráfica de resultado del test *XCreateWindow*

5. Conclusiones

En este documento mostramos el desarrollo de un servidor proxy que recibe mensajes de clientes del sistema X Window y las distribuye hacia un conjunto de servidores X, los cuales cooperan para construir la imagen del cliente, obteniendo como resultado gráficas de mayores dimensiones y mayor resolución.

Para distribuir el display del servidor X Window, nosotros utilizamos dos estrategias. La primera consiste en adaptar las *primitivas gráficas* al contexto del *display virtual*, esta tarea es realizada en el prototipo llamado proXy1. La segunda estrategia es implementada en el prototipo proXy2 y consiste en realizar el proceso de adaptación en los nodos de renderización. Esta estrategia presentó un mejor desempeño con las pruebas de rendimiento realizadas.

Con esta implementación el usuario se beneficia al disponer de una herramienta que despliega las gráficas de las aplicaciones del sistema X Window sobre grandes superficies, aumentando la cantidad de información que puede ser percibida y mejorando el proceso de interacción con el usuario, sobretodo cuando se trabaja con aplicaciones que requieren de mayor potencia gráfica que la que un monitor puede proporcionar.

Trabajos posteriores serán encaminados a desarrollar un manejador de ventanas que soporte la interacción simultánea de múltiples usuarios, provistos de ratón y teclado.

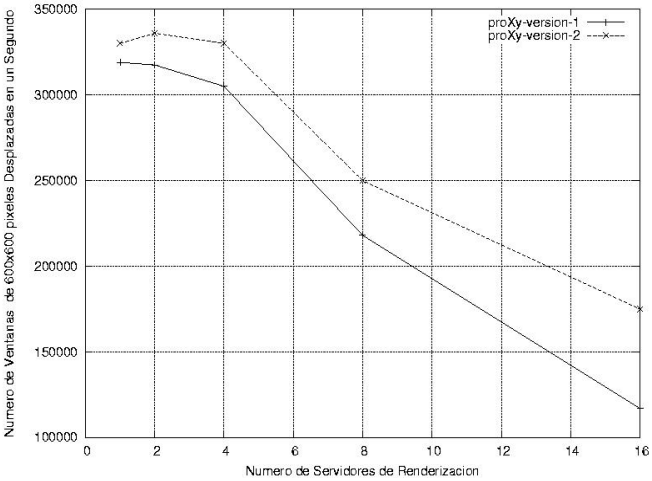

Fig. 8. Gráfica de resultado del test *XMoveWindow*

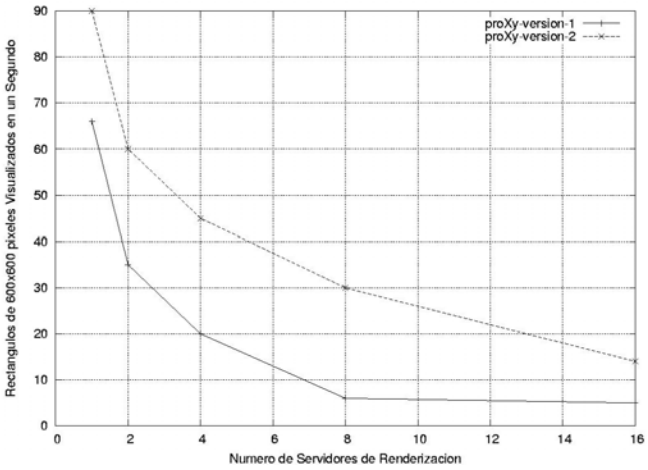

Fig. 9. Gráfica de resultado del test *XPutImage*

6. Referencias

[1] K. Li, H. Chen, Y. Chen, D. W. Clark, P. Cook, S. Damianakis, G. Essl, A. Finkelstein, T. Funkhouser, Allison Klein, Z. Liu, E. Praun, R. Samanta, B. Shedd, J. Pal Singh, G, Tzanetakis, and J. Zheng. Early experiences and challenges in building an d using a scalable display wall system. *IEEE Computer Graphics and Application*, 20(4):671-680, 2000.

[2] C. Cruz-Neira, D. J. Sandin, and T. A. Defanti. Surround/screen projection-based virtual reality: The design and implementation of the cave. *Computer Graphics* (*SIGGRAPH 93 Proceedings*), 135-142, August 1993.

[3] R.Raska, G. Welch, M. Cutts, A. Lake, L. Stesin, and H. Fuchs. The office of the future: A unified approach to image-based modeling. *In SIGGRAPH 98*, pages 179-188, July 1998.

[4] G. Humephrey and Narran. A distributed graphics system for large tiled displays. *In IEEE Visualization 99*, 1999.

[5] P. Woodward et al. The Power Wall. *University of Minnesota*, pagina web: http://www.lcse.umn.edu/research/powerwall/overview.htm

[6] D. B. Maxwell. Linux cluster power tour-wall 3-d display. *WWW.LINUXJOURNAL.COM*, pages 92-94, December 2002.

[7] Consortium x. page http://www.x.org/XOrg.html.

[8] J. Gettys and R. W. Schiefler. Xlib c languages x interface. *X Consortium Standard*, *X version 11, Release 6.4*, page xc/doc/hardcopy/X11/xlib.PS.gz.

[9] R. W. Schiefler. X window system protocol. *X Consortium Standard*, *X version 11, Release 6.4*, page xc/doc/hardcopy/XProtocol/proto.PS.gz.

[10] A. Ñye. *Xlib Programming Manual for Version 11*, volumen 1. 1993.

[11] O. Jones. Multidisplay software in x: A survey of architectures in the x resource. *O'Reilly & Associates*,

Inc., 6:97-113, 1993.

[12] J. Basic. Xmx an x protocol multiplexor. *Brown Universoty*, page http://www.cs.brown.edu/software/xmx/.

[13] J. McCormack, P. Karlton, S. Angebranndt, C. Kent, K. Packard, and G. Gill. X11perf – x11 server performance test program. http://www.xfree86.org/4.2.0/x11perf.1.html.

[14] J. Allard, V. Gouranton, L. Lecointre, E. Melin, and B. Raffin. Netjuggler : running vrjuggler with multiple displays on a commodity component cluster. Pages 275-276, March 2002.

[15] P. Augerat, C. Goudeseune, H. Kaczmarski, B. Raffin, B. Shaeffer, L. Soares, and M. K. Zuffo. Commodity cluster for immersive projection environments. Siggraph 2002 course, July 2002.

[16] G. Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett, and P. Hanrahan. Wiregl : a scalable graphics system for clusters. *In Proceedings of SIGGRAPH 2001*.

[17] Software display wall in a box. Página web: http://www.ncsa.uiuc.edu/TechFocus /Deployment/DBox/doc/vnc.html

[18] Distributed multihead x. Página web: http://sourceforge.net/projects/dmx

[19] T. Richardson, Q. Stafford-Fraser. Virtual network computing . January/February 1998.

[20] Xinerama extension to the x protocol. Página web: http://sourceforge.net/projects/xinerama