COLEGIO DE CIENCIA Y TECNOLOGÍA

LICENCIATURA EN INGENIERÍA EN SISTEMAS ELECTRÓNICOS Y DE TELECOMUNICACIONES

Diseño e implementación de un sistema de cómputo distribuido y paralelo utilizando software libre

TRABAJO RECEPCIONAL

PARA OBTENER EL TÍTULO DE LICENCIADO EN

INGENIERÍA EN SISTEMAS ELECTRÓNICOS Y DE TELECOMUNICACIONES

PRESENTA

FERNANDO OCTAVO SALINAS

Director del trabajo recepcional

M. en C. Joel Yazbek Buendía Gómez

México, D.F. octubre de 2015

UACM
Universidad Autónoma de la Ciudad de México
Nada humano me es ajeno

CA Coordinación Académica

SISTEMA BIBLIOTECARIO DE INFORMACIÓN Y DOCUMENTACIÓN

UNIVERSIDAD AUTÓNOMA DE LA CIUDAD DE MÉXICO
COORDINACIÓN ACADÉMICA

RESTRICCIONES DE USO PARA LAS TESIS DIGITALES

DERECHOS RESERVADOS©

Resumen

En este documento se presentan el diseño y la construcción de un sistema de cómputo distribuido y paralelo utilizando software libre, para el laboratorio de redes de computadoras del plantel San Lorenzo Tezonco de la Universidad Autónoma de la Ciudad de México.

El sistema de cómputo distribuido se compone de dos clúster, xexelo0 y xexelo1, configurados para realizar computación de alto desempeño; una zona de administración remota, desde donde se controlan todos los equipos de cómputo y comunicaciones, y un sistema de seguridad perimetral.

Los dos clúster siguen una arquitectura cliente-servidor, ésto es, cada clúster posee un nodo coordinador, también llamado maestro o frontend, el cual reparte las tareas en paralelo a los demás nodos esclavos, gestiona todos los procesos relativos a la interfaz de paso de mensajes y es el único equipo capaz de dar acceso a Internet.

Las computadoras, o nodos, del clúster xexelo0 emplean procesadores Intel Xeon a 2.10 GHz y los nodos del clúster xexelo1 procesadores Intel Core 2 Duo a 3.00 GHz. Además, la tecnología de red para la conexión de los equipos de cómputo y comunicaciones es Ethernet a 1 Gbps y 10 Gbps.

Empleando el sistema distribuido se consiguió disminuir el tiempo, de 134.605 minutos a 7.203 minutos, en la solución del producto de dos matrices cuadradas de 9000 x 9000 (números enteros), la cual implica realizar un billón cuatrocientos cincuenta y siete mil novecientos diecinueve millones de operaciones (1,457,919,000,000). Además, se realizó la prueba de Linpack en el clúster xexelo1, obteniendo un poder de cómputo de 150.9 Gflops.

Agradecimientos

A Dios por permitirme disfrutar de experiencias buenas y malas, por concederme situaciones de errores y aprendizajes y por situarme en el camino del estudio y el conocimiento.

A mi madre S. Leticia Salinas Hernández por todo su amor, apoyo, sabiduría, alegría y ejemplo de vida, muchas gracias por respaldarme. A mi padre Ricardo Octavo Coli por su amor, consejos, entusiasmo y ejemplo de disciplina, muchas gracias por alentarme. A los dos los amo, respeto y admiro profundamente. A mi hermano Víctor por ser mi amigo incondicional, por escucharme, por su ejemplo, por compartir sus conocimientos conmigo, te admiro y te quiero mucho. A mi hermano Ricardo por transmitirme su ánimo e influenciar mi interés por las tecnologías de la información, te quiero y respeto mucho. A todos mis tíos y primos, en especial a: Patricia, Carlos, Angélica, Pedro, Alejandra, Alberto, Antonio y Estela. Gracias por su presencia, los quiero inmensamente.

A mi maestro y director de trabajo recepcional Joel Yazbek Buendía Gómez, gracias por su confianza, sus enseñanzas y por permitirme colaborar con usted. A mis distinguidos maestros de carrera y lectores: Magali Cortez Vázquez, J. Ignacio Castillo Velázquez, Ricardo Galindo Reyes y José Luis Quiroz Fabián. Gracias a todos por sus comentarios y observaciones constructivas. Los admiro, respeto y estimo.

A mi novia y colega Ana Lilia Hernández Abonza por su cariño y motivación, eres maravillosa. A sus padres Leodegario Hernández Portilla e Hilda A. Abonza Chávez, los aprecio mucho y los considero extraordinarias personas.

A mis compañeros, amigos y colegas de la carrera de ISET, en especial a J. Armando Díaz Ramírez, Isabel Muñoz Martínez, Laura Velazquez Cortés, Sonia Luna Tlacomulco, Yeny de Jesus Tiburcio y María de la Paz Cervantes Eugenio. Son excelentes personas, gracias por apoyarme en su debido momento.

A la Universidad Autónoma de la Ciudad de México por permitirme cursar mis estudios superiores en sus aulas y por el apoyo otorgado para la impresión y empastado de este trabajo recepcional. También, un especial reconocimiento a la Secretaría de Ciencia, Tecnología e Innovación por favorecer el proyecto descrito en esta obra.

Contenido

Lista de Figuras

Lista de Tablas

Capítulo 1
Introducción

1.1. Sistemas distribuidos, clústeres de computadoras y su importancia en el mundo moderno

Muchos de los problemas que se atienden en la actualidad tienen la característica de exigir repetitivas operaciones sobre grandes cantidades de datos, por ejemplo, evaluaciones numéricas para resolver problemas científicos e ingenieriles, tratamiento digital de imágenes, modelado de estructuras atómicas, pronóstico del tiempo, búsquedas y relaciones de información sobre extensas bases de datos, entre otros. Desarrollar herramientas que permitan resolver éstos dilemas es crucial para tomar decisiones adecuadas, ya que de éstas dependen actividades como por ejemplo: la programación de horarios y rutas de barcos y aviones para viajes comerciales y turísticos, planificar las cosechas anuales de ciertos alimentos, informar a la población sobre próximas lluvias o posibles fenómenos naturales, por mencionar algunos [1].

Las computadoras son la herramienta que hoy utilizamos para obtener, procesar, almacenar y relacionar información. Son el resultado de varios años de estudio de diversos grupos de investigación motivados por responder, en principio, cuestiones matemáticas. Inherentes a las computadoras, están las redes que las interconectan y el software que las hace funcionar. En conjunto representan uno de los sistemas más complejos y funcionales que se han creado.

Las primeras computadoras capaces de realizar cuantiosas operaciones, en cortos lapsos de tiempo, fueron las llamadas supercomputadoras. Equipos fabricados especialmente para ejecutar unas cuantas instrucciones sobre grandes cantidades de datos. Desde su liberación a mediados de los años cuarenta, y hasta la fecha, han evolucionado en su arquitectura de hardware y software [2].

Para poner en perspectiva la necesidad de poder y velocidad de procesamiento, se considera el siguiente ejemplo referente al pronóstico del clima. Se propone encerrar toda la atmósfera terrestre en $5X10^8$ células (figura 1.1). Para calcular las condiciones existentes en cada célula, por ejemplo, temperatura, presión, humedad, velocidad y dirección del viento, entre otras, se usan complejas ecuaciones matemáticas. Supongamos que para calcular las condiciones de cada célula se requieren ejecutar 200 operaciones de punto flotante (tipo de operación

necesaria si los números que intervienen tienen parte fraccionaria o están elevados a una potencia). En un sólo instante de tiempo se necesitarían realizar 10^{11} operaciones. Si se tuviera que pronosticar el clima de siete días, usando intervalos de un minuto se precisarían más de 10^4 pasos de tiempo, equivalente a 10^{15} operaciones de punto flotante. Una computadora con capacidad de 100 Mflops tardaría más de 100 días en completar los cálculos. Para lograr lo mismo en cinco minutos se tendría que disponer de una computadora operando a 3.4 Tflops [3].

Figura 1.1: Atmósfera terrestre dividida en células.

El ejemplo anterior manifiesta que la demanda de velocidad de cómputo está relacionada con el tipo de datos y la cantidad de operaciones que las computadoras deben realizar, además de la eficiencia de los algoritmos para realizar las operaciones [4].

Para atender la demanda de procesamiento, organizaciones del sector industrial, de estudios superiores, de investigación y de gobierno, se ocuparon de mejorar las características de las computadoras, es decir, la tecnología de los microprocesadores, de los módulos de memoria RAM, de los dispositivos de almacenamiento y de las placas de circuitos (donde los elementos anteriores se interconectan). Casi a la par, las redes de computadoras también evolucionaron para lograr transmitir más datos en menos tiempo a distancias cada vez más amplias.

En la búsqueda de realizar más operaciones en menos tiempo y ampliar la atención a más usuarios, las redes de: sensores, dispositivos móviles, estaciones de trabajo, servidores, entre otras, se han acoplado, mediante software, para interactuar y dar la apariencia de ser un único sistema coherente. Así, el conjunto de las subredes mencionadas forman los llamados sistemas distribuidos.

Actividades como: retirar efectivo de un cajero automático, coordinar el tráfico de vehículos a través de los semáforos, observar un espectáculo vía remota, consultar el acervo de una biblioteca a través de la red escolar, llenar algún formulario en el portal web de la secretaría de finanzas, comprar o reservar boletos de viaje, son algunos de los servicios que los sistemas distribuidos ofrecen.

Por otra parte, los sistemas distribuidos se pueden clasificar en clústeres y en grids [5]. Se hablará de éstos en la sección 2.1.1.

1.2. Planteamiento del problema

Para brindar los servicios mencionados en la sección anterior, se debe disponer de una plataforma de hardware y software estable y funcional. En la actualidad hay compañías tecnológicas como Amazon, HP, IBM, Cray Inc, SGI, Dell, entre otras, que ofrecen estructuras informáticas con plataformas de hardware y software especializadas y adaptadas para satisfacer las necesidades de sus diferentes clientes: empresas de gobierno, empresas privadas, universidades y centros de investigación. Comúnmente las empresas privadas y las secretarias de gobierno (por cuestiones de seguridad) son las que prefieren adquirir los productos y servicios con licencia de software propietario. Por otro lado, parte de la comunidad técnica y científica optan por personalizar sus propios clústeres usando software de código abierto [8].

De esta manera, el presupuesto, los requerimientos de seguridad y los conocimientos para mantener un sistema distribuido son las principales razones para optar por una solución privativa o crear una solución con software libre.

Debido a lo anterior, en este documento se expone un marco de referencia para diseñar e implementar un sistema de cómputo distribuido con software libre. Además se describe cómo utilizarlo y se reportan los resultados obtenidos al ejecutar un programa paralelizado sobre él.

1.2.1. Objetivo General

Diseñar e implementar un sistema de cómputo distribuido y paralelo para establecer diferentes configuraciones de software.

1.2.2. Objetivos Particulares

Diseñar e implementar el sistema de cómputo distribuido y paralelo de acuerdo a un presupuesto económico, a las condiciones físicas y a los recursos de hardware del laboratorio B404 del plantel San Lorenzo Tezonco de la UACM.

Configurar el sistema de cómputo distribuido y paralelo para realizar cómputo de alto desempeño.

1.3. Alcances y Limitaciones

Este documento trata sobre el diseño e implentación de un sistema de cómputo distribuido y paralelo utilizando software libre, el cual comprende una red de datos basada en los estándares ANSI/TIA/EIA-568-C.1 y 606-A, un área de administración remota, dos clústeres de computadoras para cómputo de alto desempeño y un sistema de seguridad perimetral.

Ambos clústeres contemplan únicamente un servidor central, llamado frontend, para asignar a los nodos esclavos, a través del estándar MPI, los intervalos de datos comprendidos en los problemas paralelizados y compilados en lenguaje C.

En cuanto a las pruebas de funcionalidad, las operaciones comprendidas fueron de granularidad fina.

1.4. Contenido

El resto del documento se divide de la siguiente forma. En el capítulo 2 se tratan los conceptos teóricos de los sistemas de cómputo distribuidos, sus propiedades y su clasificación; la definición, las propiedades y los tipos de clústeres de computadoras; la clasificación de las computadoras paralelas y los tipos de acceso a la memoria principal; el sistema operativo linux, OpenMPI como middleware y los límites del paralelismo. En el capítulo 3 se describe el diseño e implementación del sistema de cómputo distribuido para el laboratorio de redes de computadoras B404 del plantel San Lorenzo Tezonco de la UACM. En el capítulo 4 se explica cómo ejecutar tareas en paralelo en los clústeres; se presentan los resultados de evaluar una matriz cuadrada de 9000 x 9000 en xexelo0 y xexelo1 y también los valores obtenidos en la prueba de referencia Linpack en xexelo1. Finalmente, en el capítulo 5 se establecen las conclusiones y el trabajo a futuro.

Capítulo 2
Marco Teórico

2.1. Sistemas de cómputo distribuidos

Se da comienzo a este capítulo con dos definiciones de lo que es un sistema de cómputo distribuido. La primera dice que ***un sistema distribuido es una colección de computadoras independientes que, a la percepción de sus usuarios, aparentan ser un único sistema coherente*** [5]. La segunda indica que ***es aquel en el que los elementos de hardware y software, localizados en computadoras conectadas en red, comunican y organizan sus acciones exclusivamente con el paso de mensajes*** [6]. En las dos definiciones no se especifica el tipo de computadoras ni tampoco se asume alguna tecnología de conexión entre éstas. De tal forma que éstas definiciones engloban una variedad de sistemas digitales y tecnologías de red.

Ejemplos de sistemas distribuidos son: la world wide web, la internet, las intranets, la computación móvil y ubicua y los diferentes tipos de clústeres (de éstos últimos se hablará a lo largo de éste trabajo).

Los sistemas distribuidos poseen las siguientes características:

- Heterogeneidad: Inclusión de diversidad tecnológica de redes, hardware, lenguajes de programación, sistemas operativos, y que el sistema de cómputo distribuido esté exento de problemas de compatibilidad.

- Extensibilidad: Propiedad que permite añadir nuevas características y servicios al sistema distribuido de forma dinámica, especialmente los servicios de compartición de recursos.

- Seguridad: El manejo de la información debe ser confidencial, inaccesible para usuarios no autorizados; íntegra, no debe ser corrompida o alterada y disponible, los usuarios autorizados siempre tendrán acceso a ella.

- Escalabilidad: Habilidad para funcionar en diferentes proporciones, es decir, el sistema se comporta estable pese a que se incremente significativamente el número de usuarios y

recursos.

- Tratamiento de fallos: Capacidad de detectar fallos, enmascararlos, tolerarlos, recuperarse de ellos y brindar redundancia.

- Concurrencia: Procesar simultáneamente múltiples solicitudes de diversos clientes sin provocar errores, durante y al final del procedimiento.

- Transparencia: La manera en la que se realizan las tareas debe ser imperceptible para el usuario. Éste debe asumir al sistema como una sola entidad coherente.

2.1.1. Clasificación de los sistemas distribuidos

Como se mencionó en la introducción, los sistemas distribuidos se pueden clasificar en clústeres y en grids. En los primeros, el hardware primordial consiste en grupos de estaciones de trabajo o pc's similares, estrechamente conectadas a través de una red de alta velocidad y que poseen el mismo sistema operativo. En los grids se manejan colecciones de subsistemas con alto grado de heterogeneidad, diferente hardware, diferentes sistemas operativos, distintas tecnologías de red, distintas políticas de seguridad, entre otras.

A partir de aquí se hablará más de los clúster, y específicamente de los que emplean computadoras de escritorio y servidores.

2.2. Clústeres de computadoras

Un clúster de computadoras es una consolidación de software y hardware configurados para resolver tareas computacionales de forma paralela y en red [5]. Para mantener uniformidad en el poder de cómputo, sus elementos comparten características similares. Sin embargo, también existen clústeres heterogéneos.

El primer clúster de computadoras fue desarrollado en 1960 por IBM. Grandes mainframes conectados para proporcionar una más rentable forma de paralelismo comercial. Sin embargo, los clústeres de computadoras adquirieron mayor impulso hasta la convergencia de tres tendencias: los microprocesadores de altas prestaciones, las redes de alta velocidad y las herramientas estándar para la computación distribuida de alto rendimiento [7]. Así, el primer clúster de mayor popularidad fue el llamado Beowulf (figura 2.1), creado en el Centro Goddard de Vuelos Espaciales de la NASA, en 1994. El cual utilizó las primeras versiones del sistema operativo Linux y una PVM (*Parallel Virtual Machine* 'Máquina Virtual Paralela') sobre dieciséis computadoras personales Intel base 80486 de 100 MHz conectadas por una LAN Ethernet dual de 10Mbps [8].

PVM es una biblioteca de funciones enlazables que permite ejecutar rutinas en computado-

ras separadas, pero interconectadas, y así intercambiar datos y coordinar sus operaciones. El desarrollo de PVM fue un hito de la aplicación práctica del modelo de paso de mensajes (desarrollado por el Laboratorio Nacional Oak Ridge, la Universidad Emory y la Universidad de Tennessee). Posterior a PVM, el estándar de paso de mensajes MPI (*Message Passing Interface* 'Interfaz de Paso de Mensajes') fue definido. Éste estándar es el empleado en este proyecto y se hablará de él en la sección 2.7.5.

Figura 2.1: Primer clúster Beowulf [9].

2.2.1. Clasificación de los clústeres de computadoras

Existen dos clasificaciones principales para los clústeres de computadoras, los de alta disponibilidad y los de computación de alto rendimiento [10].

Los clústeres de alto rendimiento se utilizan para realizar grandes cantidades de cálculos matemáticos y por lo tanto requieren alto poder de procesamiento. Se utilizan comúnmente en centros de investigación y son los que se tratan en éste proyecto.

Los clústeres de alta disponibilidad. Se concentran en mantener disponibles las aplicaciones de los servidores, bases de datos, aplicaciones de negocio, sitios de comercio electrónico, entre otros. Se distinguen por tener: redundancia en software y hardware, fiabilidad, tolerancia a fallos, facilidad de mantenimiento, entre otras características.

Por otro lado, la forma de conectar los componentes físicos de los dos tipos de clústeres es similar. Así, una misma plataforma de hardware se puede configurar, mediante software, para cambiar su comportamiento. En este proyecto, la plataforma de hardware se configuró para aproximar su comportamiento al de un clúster de alto rendimiento.

2.2.2. Modelo de capas de un clúster

Los clústeres se componen de cuatro elementos principales. Los primeros dos son de hardware, los nodos de trabajo y la red que los interconecta; y los siguientes dos son de software, el grupo de herramientas usadas para desarrollar programas de aplicación paralela y el ambiente de software para administrar los recursos del clúster [8]. En la figura 2.2 se muestran que las capas de hardware y sistema operativo están en diferentes computadoras, sin embargo las capas de middleware y de aplicaciones distribuidas hacen que los equipos de cómputo se comporten como una única entidad computacional.

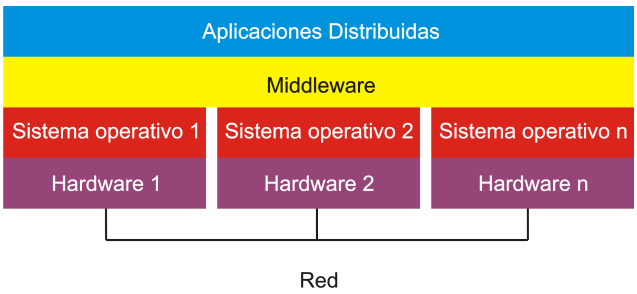


Figura 2.2: Modelo de capas de un clúster [6].

2.2.3. Tecnologías de interconexión de los clústeres

Comúnmente, la interconexión de los elementos de los clústeres se realiza a través de Infiniband, 10G, Gigabit Ethernet, entre otras [11]. Cabe señalar que una red de computadoras, por sí misma, no es considerada un sistema distribuido. La diferencia está en el software, básicamente en el sistema operativo y en la capa de middleware. Así que, es aceptable decir que un sistema distribuido es un sistema de software establecido sobre una red de computadoras [12].

Por otra parte la capa de hardware de la figura 2.2 se enfoca más en el tipo de computadoras que se utilizan en los clústeres, las cuales se revisan en la siguiente sección.
En cuanto a los tópicos inherentes a la capa de middleware y a la capa de aplicaciones distribuidas, se verán en la sección 2.2.6.4. y 2.2.7.

2.2.4. Capa de hardware de un clúster

Uno de los objetivos de los clústeres, es que el conjunto de los elementos que lo forman se asuma como una única entidad funcional, sin embargo, actualmente cada elemento de cómputo posee más de una unidad de procesamiento, por lo que ya se le puede considerar como una computadora paralela y como una única entidad funcional. Así que, a partir de éste pensamiento se pueden clasificar estas entidades de cómputo como sigue.

2.2.5. Tipos de computadoras paralelas

Existen varios criterios para la clasificación de las computadoras paralelas. Entre los más populares están: según el grado de simultaneidad en el manejo de instrucciones y datos (Taxonomía de Flynn); la estructura de las unidades de procesamiento y el acceso a la memoria principal, esto es, múltiples unidades de procesamiento compartiendo el acceso a la memoria RAM, múltiples computadoras conectadas en red accediendo a sus respectivos módulos de memoria, o la combinación de las dos anteriores. También está el criterio del nivel de granularidad, el cual se basa en el reconocimiento de subtareas o instrucciones en un programa, que pueden ser ejecutadas en paralelo sobre un sistema multiprocesador [13].

2.2.5.1. Taxonomía de Flynn

Esta clasificación comprende cuatro tipos de computadoras . El primer tipo es el SISD (*Single Instruction Stream - Single Data Stream* 'Flujo Único de Instrucciones - Flujo Único de Datos'), el cual ejecuta una instrucción por unidad de tiempo sobre un elemento de dato (figura 2.3). El segundo tipo es el SIMD (*Single Instruction Stream - Multiple Data Stream* 'Flujo Único de Instrucciones - Flujo Múltiple de Datos'), es un sistema de tipo síncrono, donde los procesadores, que no están en modo ocioso, ejecutan la misma instrucción sobre diferentes datos (figura 2.4). El tercer tipo es el MISD (*Multiple Instruction Stream - Single Data Stream* 'Flujo Múltiple de Instrucciones - Flujo Único de Datos'), el cual realiza múltiples instrucciones sobre los mismos datos (figura 2.5) y el cuarto tipo es el MIMD (*Multiple Instruction Stream - Multiple Data Stream* 'Flujo Múltiple de Instrucciones - Flujo Múltiple de Datos') el cual es un sistema multiprocesador de propósito general, el cuál comprende una única cadena de instrucciones por cada procesador y cada instrucción opera sobre diferentes datos (figura 2.6).

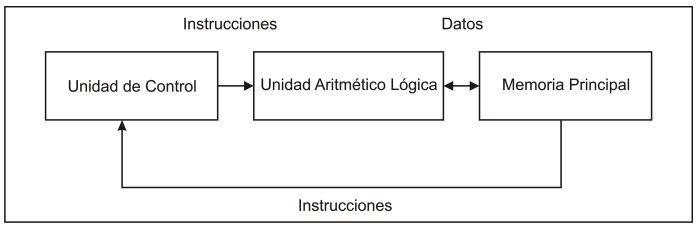


Figura 2.3: (SISD). Flujo Único de Instrucciones - Flujo Único de Datos.

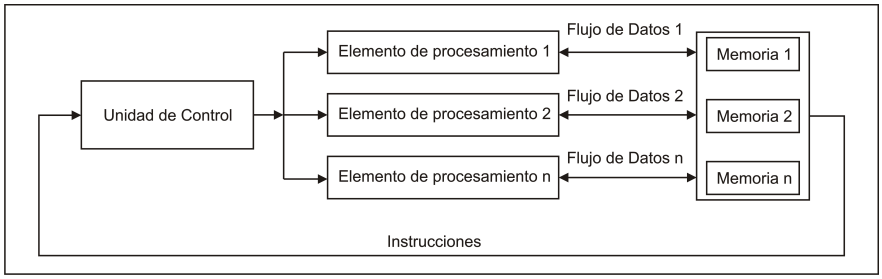


Figura 2.4: (SIMD). Flujo Único de Instrucciones - Flujo Múltiple de Datos.

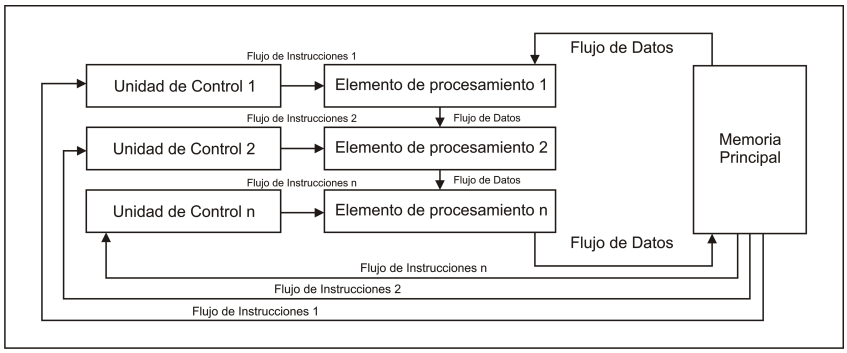


Figura 2.5: (MISD). Flujo Múltiple de Instrucciones - Flujo Único de Datos.

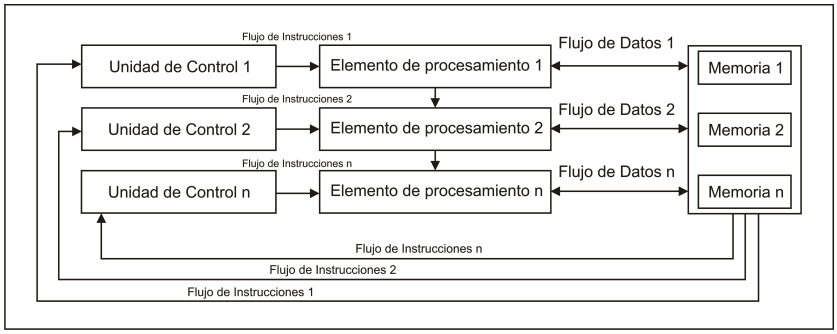


Figura 2.6: (MIMD). Flujo Múltiple de Instrucciones - Flujo Múltiple de Datos.

2.2.5.2. Estructura de las unidades de procesamiento

En esta clasificación, existen básicamente tres tipos de computadoras paralelas. La primera es una única computadora con múltiples unidades de procesamiento (figura 2.7). La segunda es un conjunto de computadoras conectadas a través una red (figura 2.8) y la tercera es la combinación de los dos primeras (figura 2.9).
De acuerdo a la cercanía de las unidades de procesamiento, se dice que los procesadores del

primer tipo de computadora están fuertemente acoplados y los procesadores del segundo y tercer tipo de computadoras están débilmente acoplados.

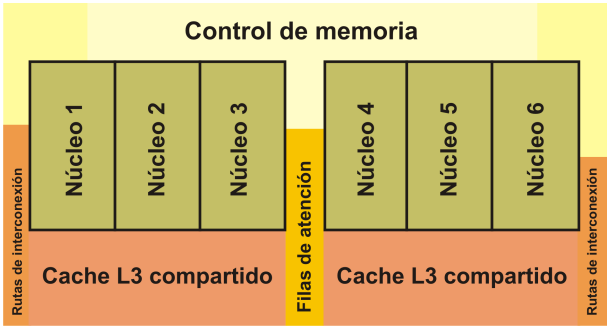


Figura 2.7: Computadora paralela tipo multinúcleo [14].

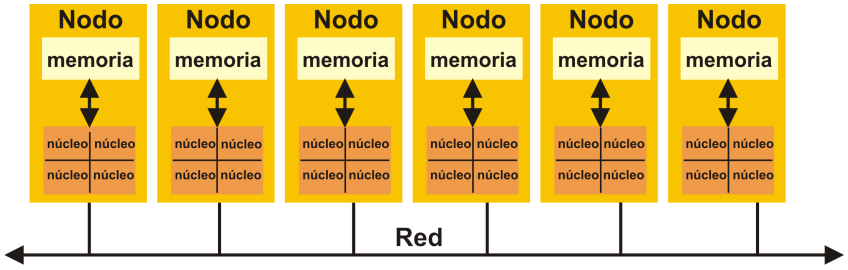


Figura 2.8: Computadora paralela tipo multicomputador [1].

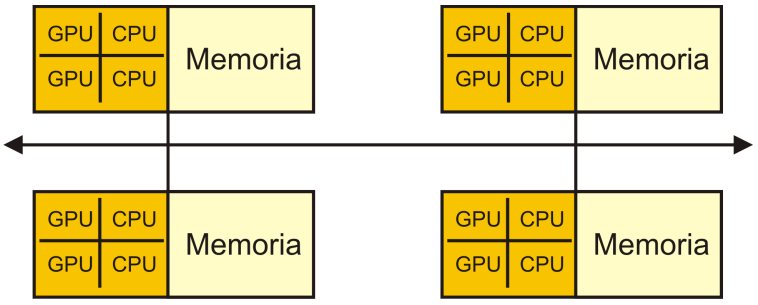


Figura 2.9: Computadora paralela híbrida [15].

2.2.5.3. Acceso a la memoria principal

Existen tres tipos de acceso a la memoria de las computadoras paralelas. El primero es el de *memoria compartida*, en el cual todos los procesadores tienen acceso a las mismas direcciones de memoria, esto quiere decir que si un procesador hace cambios a una localidad de memoria, será visible para los demás procesadores. La memoria compartida se clasifica como UMA (*Uniform Memory Access* 'Acceso Uniforme a Memoria') y NUMA (*Non Uniform Memory*

Access 'Acceso No Uniforme a Memoria'). La diferencia entre UMA y NUMA esque en la primera todos los procesadores pueden acceder a la memoria principal a la misma velocidad y de manera uniforme. En la configuración NUMA el acceso a localidades específicas de memoria no es igual para todos los procesadores.

La ventaja del acceso a memoria compartida es que la distribución de datos entre tareas es más rápida que en la arquitectura de memoria distribuida. Además de que el espacio global de direcciones proporciona una perspectiva de programación amigable.
La principal desventaja es que la escalabilidad es prácticamente nula entre memoria y unidades de procesamiento.

El segundo tipo de acceso es el de *memoria distribuida* en la cual se requiere una red de comunicación para conectar la memoria de cada procesador. No existe un espacio global de direcciones que compartan éstos y por lo tanto cualquier cambio a una localidad de memoria no afecta a las de los otros procesadores. Es responsabilidad del programador definir explícitamente la sincronización entre tareas, es decir, cómo y cuándo los procesadores tienen acceso a los datos de las otras memorias [1]. Las ventajas de éste modelo son:

- La memoria y el número de procesadores son escalables (crecen proporcionalmente).
- Cada procesador accede rápidamente a su propia memoria sin interferir con los demás.
- Se pueden utilizar equipos fuera de soporte (procesadores y redes).

El tercer tipo de acceso es la combinación de las dos anteriores. Las computadoras más grandes y veloces, en la actualidad, lo emplean y sus principales ventajas son la escalabilidad y su gran utilización, sin embargo la programación se vuelve más compleja debido a que el programador tiene que describir el paso de mensajes y la memoria de compartida.

De acuerdo a la taxonomía de Flynn, los clústeres xexelo0 y xexelo1 son computadoras paralelas tipo SIMD, debido a que ejecutan las mismas instrucciones sobre una gran cantidad de datos. Por otro lado, el acceso a la memoria de ambos clústers es de tipo no uniforme, porque no se estableció alguna configuración para que cada unidad de procesamiento de xexelo0 y xexelo1 tuviera acceso a diferentes módulos de memoria ubicados en diferentes nodos.

2.2.6. Capa de sistema operativo y middleware

Uno de los objetivos de las redes de computadoras es la compartición de recursos, habitualmente impresoras, escáners, fax, lectores de CD's, archivos de datos, archivos de programas, entre otras. En el caso de los clústers de alto desempeño, el recurso principal es el poder de procesamiento, que invariablemente compromete a las unidades de procesamiento, espacio de memoria RAM y almacenamiento. El sistema operativo es el que se encarga de los recursos físicos de las computadoras y los provee mediante una interfaz de llamadas al sistema [6].

Los sistemas operativos permiten su uso a través de dos interfaces: mediante una la línea

de comandos, sujeta a una sintaxis, y a través de una interfaz gráfica (ventanas, botones, puntero, por mencionar algunos). Para el cómputo de alto desempeño se prefieren los sistemas operativos sin entorno de escritorio, debido al consumo de memoria RAM y procesador que éste representa.

Dicho lo anterior, el sistema operativo es el que atiende las invocaciones de uso de recursos para el middleware. Estas dos capas están muy relacionadas y la elección de ambas depende del rendimiento que su combinación ofrezca.

Existen una variedad de sistemas operativos, orientados a servidores para clústers, de código propietario (tanto de Linux como de Windows) y de código abierto (particularmente Linux). Sin embargo, las distribuciones de Linux son dominantes en el ámbito de cómputación de alto rendimiento. Basta revisar la lista de las quinientas computadoras de HPC (*High Performance Computing* 'Computación de Alto Desempeño') más poderosas del mundo en la página electrónica *http://www.top500.org/* para notar que el 98.8 % utilizan distribuciones Linux.

En este proyecto se utilizó la combinación (sistema operativo - middleware) Linux - MPI y a continuación se hablará de éstos.

2.2.6.1. Linux

Linux es el sistema operativo de código abierto desarrollado por Linus Torvalds en 1991 y está basado en Unix. A partir de su liberación, desarrolladores de todo el mundo han mejorado, extendido y modificado el código fuente, cuidando la inexistencia de características inútiles, de código redundante o de una mala programación.

2.2.6.2. ¿Por qué usar Linux en los clústers?

La razón más importante para usar linux en los clústers es por su flexibilidad. Gracias a que es de código abierto puede ser modificado, reorganizado y ajustado para cualquier tarea. Otra razón es porque existen muchos recursos en la web, foros, manuales, blogs, sitios oficiales, wikis, entre otros, donde se resuelven muchas cuestiones de usuarios principiantes y de usuarios avanzados. Además de ésto, soporta muchos tipos de procesadores. Alpha, PowerPC, IA32, IA64, entre muchos otros [8].

2.2.6.3. Kernel Linux

El núcleo o kernel, es un programa cuya característica principal es que su código se ejecuta con privilegios completos de acceso a los recursos físicos de una computadora. Su función es la de ser una interfaz para el hardware y facilitar un ambiente de administración de procesos y memoria [6]. De hecho, el término Linux se refiere, de forma más precisa, al uso de Unix como núcleo.

Comúnmente, muchas distribuciones de Linux compilan sus núcleos para la primer generación de procesadores Pentium, asegurando su funcionamiento en una gran cantidad de máquinas. No obstante, para un clúster de alto rendimiento, la compilación del kernel puede significar una importante optimización (debido a la explotación de las instrucciones más avanzadas de la CPU).
En este proyecto no se realizó una compilación personalizada del kernel respecto a la arquitectura de los procesadores, sin embargo, la distribución de Linux CentOS 6.6 soporta arquitecturas de 32 y 64 bits, adecuado para sustentar la siguiente capa del modelo de sistema distribuido, la capa de middleware.

2.2.6.4. Capa de middleware

El middleware se refiere a los procesos de un conjunto de computadoras, que se relacionan entre sí para generar medios de comunicación y de recursos compartidos en aplicaciones distribuidas. Ocultan la heterogeneidad y proporcionan un modelo de programación [6]. Los servicios del middleware existen entre la capa de aplicación, la capa de sistema operativo y los servicios de red. Ésta tecnología evolucionó durante los años noventa para proveer interoperabilidad a las arquitecturas cliente-servidor.

2.2.6.5. La especificación MPI y OpenMPI

MPI es una especificación para los desarrolladores y para los usuarios de las bibliotecas de paso de mensajes. Existen varios proyectos que utilizan la especificación MPI (MPICH, MPICH2, MVAPICH, MVAPICH2, MS-MPI, entre otros). *OpenMPI* es una implementación de código abierto, de uso libre, de libre redistribución y de alto rendimiento. Además representa la unión de otros proyectos como FT-MPI, LA-MPI, LAM/MPI y PACX-MPI y recibe soporte de una comunidad muy amplia y de distintos sectores (consorcio de académicos, investigadores y socios de la industria de la computación) [16].

MPI se concentra en un modelo de programación paralela de paso de mensajes. Los datos son trasladados desde el espacio de dirección de un proceso hacia el espacio de dirección de otro proceso, mediante operaciones cooperativas dentro de cada proceso.

Algunas de las características de OpenMPI: soporta de redes heterogéneas; la misma biblioteca soporta distintas tecnologías de red; utiliza compiladores C, Fortran y Java; se adecua a sistemas operativos de 32 y 64 bits; es portable y soporta ejecución de trabajos para modelos SIMD y MIMD, entre otras características.

OpenMPI requiere de los compiladores c, c++ y fortran. Éstos deben instalarse en todos los nodos del clúster para que OpenMPI los utilice de acuerdo a la tabla 2.1. Los compiladores son un tema muy extenso y no se contemplan en este trabajo.

Tabla 2.1: Compiladores OpenMPI.

Lenguaje	Nombre del compilador en OpenMPI
C	mpicc
C++	mpiCC, mpicxx, o mpic++
Fortran	mpifort, mpif77 y mpif90

En los siguientes párrafos se hablará sobre los problemas paralelizables y su ejecución en un clúster.

2.2.7. Computo paralelo y capa de aplicaciones distribuidas

Hablar de clústeres de alto rendimiento implica hablar del cómputo paralelo y una forma adecuada de entenderlo es comparándolo con el cómputo serial.

Tradicionalmente el software se escribía para hacer una sola cosa a la vez. Un problema grande era descompuesto en conjuntos de instrucciones discretas, éstas eran ejecutadas en secuencia (una a una) mediante un único procesador y en una unidad de tiempo por instrucción (figura 2.10). Por otro lado, el cómputo paralelo es el uso simultáneo de múltiples recursos de cómputo para resolver un problema. Así, una tarea grande es dividida en tareas más pequeñas y éstas, a su vez, subdivididas en series de instrucciones y ejecutadas de forma concurrente (figura 2.11).

La idea de utilizar varios procesadores trabajando juntos no es nada nuevo, desde 1958 hasta 1996, personas como S. Gill, Charles Holland, John Horton Conway o Michael J. Flynn escribieron acerca del cómputo paralelo y sus implicaciones [3].

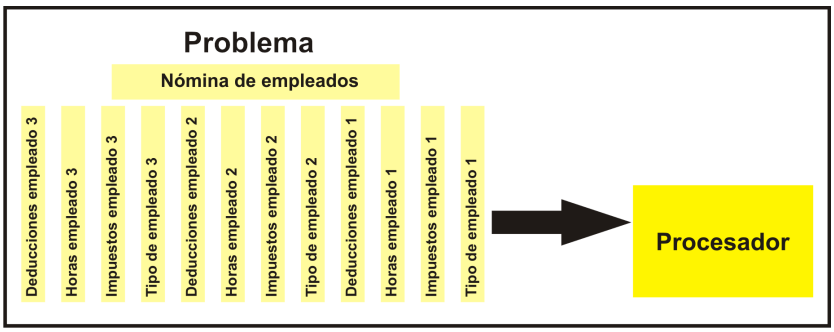


Figura 2.10: Procesamiento serial [1].

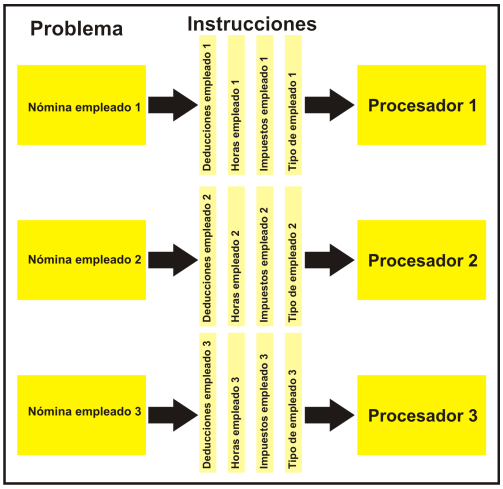


Figura 2.11: Procesamiento paralelo [1].

Cabe señalar que existen problemas que no pueden separarse en sub-problemas individuales y son llamados inherentemente secuenciales o seriales. En estos problemas, cada etapa depende de la entrada presente y del estado anterior. La simulación de un brazo en movimiento es un ejemplo de problema serial. Así que, para relacionar el estímulo del músculo (la entrada) y el movimiento del brazo (salida) se utilizan ecuaciones diferenciales, éstas se evalúan en el tiempo mediante un integrador y se ejecutan en pequeños pasos para generar una simulación de todo el movimiento. En consecuencia, esta tarea no se puede realizar en paralelo [17].

En cambio, para las tareas que sí pueden ser paralelizadas, se pueden elegir dos tipos de ejecución, la explícita y la implícita.

El método explícito es determinado por el programador, ya sea añadiendo mensajes con MPI o con hilos mediante POSIX (*Portable Operating System Interface - UniX* 'Interfaz de Sistema Operativo Portable'). Sin embargo, la implementación y depuración no es fácil. Para evitar ésto, se utilizan lenguajes de programación tales como C y Fortran, además de las bibliotecas de MPI para implementar las aplicaciones.

El método implícito es utilizado cuando se provee información acerca de la concurrencia de la aplicación. Entonces las directivas especificas del compilador o las herramientas de paralelización, analizan el programa y detectan automáticamente el paralelismo, así deciden cómo ejecutar la parte concurrente del programa.

2.2.7.1. Límites de la paralelización

Utilizar más de un procesador para realizar una tarea y completarla en el menor tiempo posible es la encomienda del cómputo paralelo. Sin embargo, en la práctica, disponer de varios procesadores, lejos de ayudar a realizar las cosas más rápido, incrementa el tiempo de respuesta dada la dificultad de paralelizar algunas tareas.

En 1967, Eugene Myron Amdahl estableció su famosa Ley de Amdahl, la cual propone la cantidad de veces que puede ser más rápida la solución de una tarea (***speedup***) de acuerdo a la fracción de código ***P*** que puede ser paralelizada. En la ecuación 2.1 se observa la ley de Amdahl.

$$speedup = \frac{1}{1 - P} \tag{2.1}$$

En 2.1 se advierte que, si nada del código puede ser paralelizado (P = 0), entonces no existe un incremento en la rapidez de solución (speedup = 1).

Por otro lado, si todo el código es paralelizable (P = 1), entonces el incremento en la velocidad de solución es muy grande (teóricamente infinita).

Si el cincuenta por ciento del código es paralelizable, entonces el speedup es igual a dos y por lo tanto el código se ejecutará dos veces más rápido.

En términos del número de procesadores, la ley de Amdahl se plantea de acuerdo a la ecuación 2.2.

$$speedup = \frac{1}{\frac{P}{N} + S} \tag{2.2}$$

Donde ***P*** es la parte de código paralelizada, ***N*** el número de procesadores y ***S*** la parte de código no paralelizada (serial).

En la figura 2.12 se muestra que el speedup con la ecuación 2.1 no es muy alto a pesar de que el 90 % del código del programa es paralelizable. En la figura 2.13 se observa que el incremento en la rapidez de ejecución tiene un límite, independientemente de la cantidad de procesadores que se utilicen, de acuerdo a la ecuación 2.2.

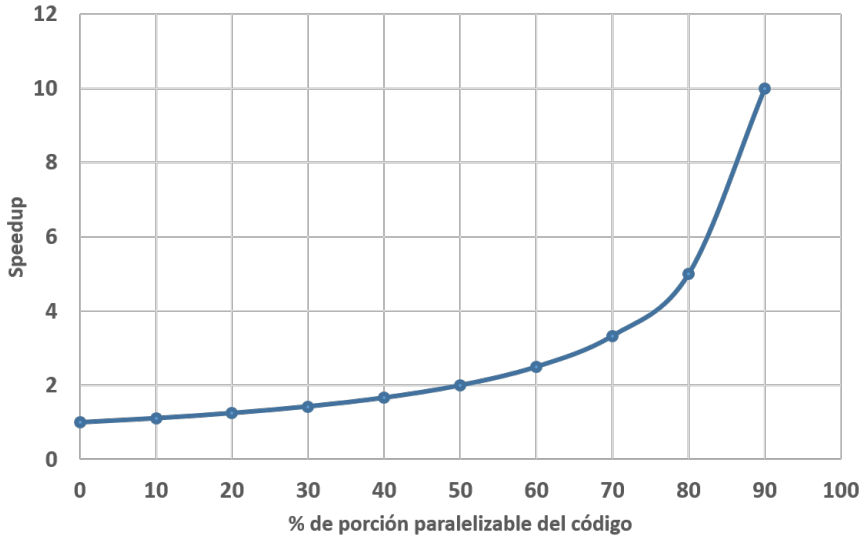


Figura 2.12: Speedup versus porcentaje de la porción paralela de código [18].

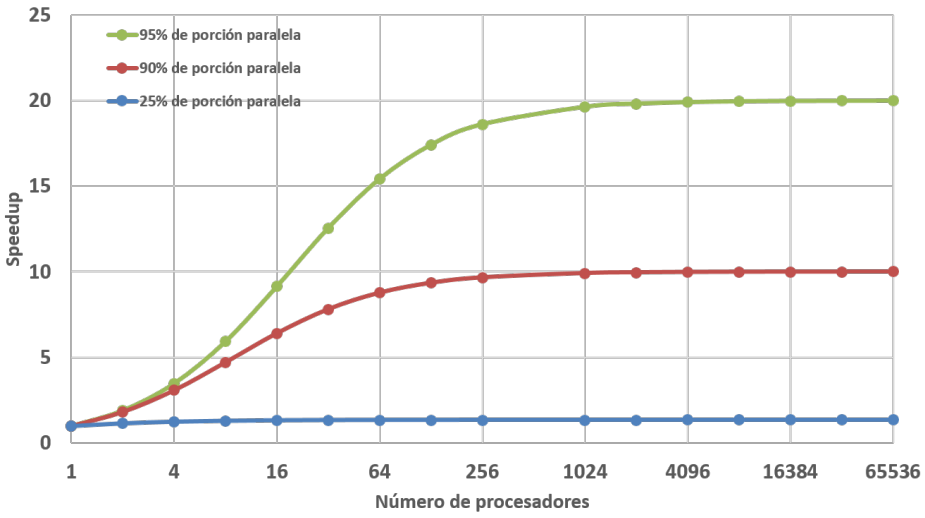


Figura 2.13: Speedup versus número de procesadores [19].

El factor de speedup también se puede calcular como se observa en la ecuación 2.3 [3].

$$speedup = \frac{Ts}{Tp} \tag{2.3}$$

Donde ***Ts*** es el tiempo en el que un solo procesador ejecuta un programa escrito en forma serial y ***Tp*** es el tiempo en el que dos o más procesadores ejecutan un programa escrito en paralelo. Por simplicidad y mejor aproximación a los valores obtenidos, se utilizó la ecuación 2.3 para representar el speedup de los clústeres.

En la práctica, es posible observar casos en donde el uso de más nodos (procesadores o núcleos de procesamiento) aumenta el tiempo de solución de un problema. Los posibles motivos son que la cantidad de datos a procesar sea pequeña y se puedan evaluar más rápido con una sola computadora; que los procesadores repitan operaciones sobre los mismos datos, relacionado a una mala asignación de intervalos de datos y por lo tanto a una mala programación. También se puede deber a que se pierda más tiempo en los procesos de envío, recepción, retorno y reintegración de datos, los cuales implican procesos inherentes al paso de mensajes.

Hasta aquí se ha presentado parte de la teoría que sustenta a los sistemas distribuidos y en concreto a los clústeres de alto rendimiento. En el siguiente capítulo se abordará todo el proceso de construcción del sistema de cómputo distribuido y paralelo del laboratorio de redes de computadoras B404 de la Universidad Autónoma de la Ciudad de México del plantel San Lorenzo Tezonco.

Capítulo 3
Diseño e implementación del sistema de cómputo distribuido y paralelo

En este capítulo se describen todas las actividades referentes al diseño e implementación del sistema de cómputo distribuido y paralelo con software libre. Se detallan todos los componentes de hardware empleados, su distribución, montaje y conexión. Además se explica el tipo y configuración de software utilizado para los clústeres xexelo0, xexelo1 y para la zona de administración remota. También se indican las herramientas de monitoreo de los servidores de xexelo0, de los switches Extreme Networks y Netgear.

El diseño del sistema distribuido obedece a las necesidades y recursos (económicos y de hardware) del laboratorio de redes de computadoras B404 de la UACM San Lorenzo Tezonco, esto es, implantar una infraestructura estable y adaptable para realizar cómputo distribuido y paralelo con software libre.

3.1. Elementos del sistema de cómputo distribuido y paralelo

El sistema de cómputo planeado consta de dos clústeres de computadoras y una zona de administración remota. El primer clúster, xexelo0, se compone de cuatro servidores multiprocesador y multinúcleo de arquitectura homogénea (en hardware y software) y se comunican a 10 Gbps. El segundo clúster, llamado xexelo1, se conforma de diez computadoras binúcleo de arquitectura homogénea (en hardware y software) y se comunican a 1 Gbps. En cada clúster existe una computadora que actúa como frontend, o nodo coordinador, y a través de ésta se distribuyen y ejecutan las tareas en paralelo. Además, el software para el sistema operativo y el middleware de ambos clústeres es libre. Por otra parte, la zona de administración remota se compone de dos equipos binúcleo de arquitectura de hardware y software semejante. En cuanto al cableado estructurado se siguió el estándar EIA/TIA-568-C.1 para edificios comerciales. En la figura 3.1 se muestra la topología del sistema distribuido.

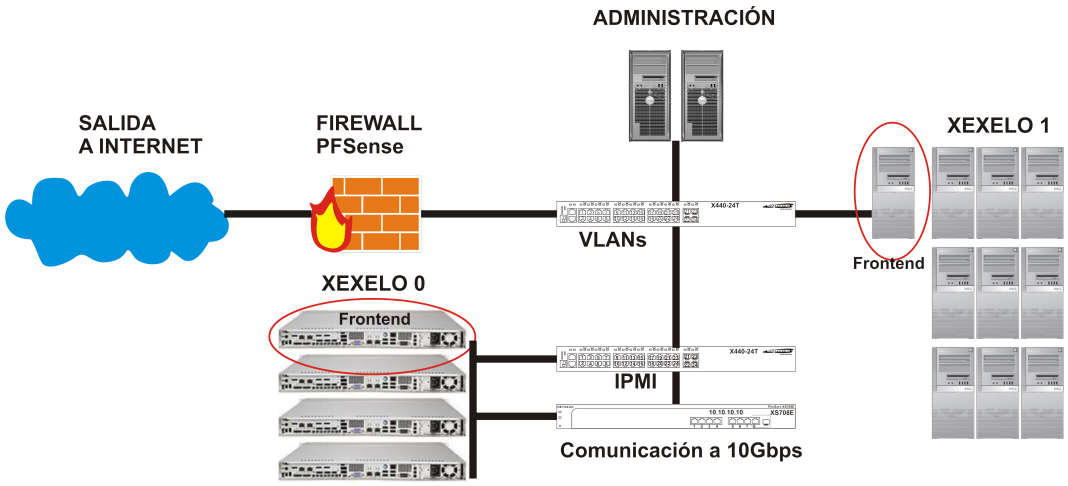


Figura 3.1: Topología del sistema distribuido.

El presupuesto empleado en la adquisición de ciertos recursos de hardware, fue suministrado por la Secretaría de Ciencia, Tecnología e Innovación (SECITI) y por la Universidad Autónoma de la Ciudad de México a través del Programa Operativo Anual. La elección de todos los componentes obedeció a factores como: el espacio de trabajo, el presupuesto, los equipos existentes de cómputo y comunicaciones, y en el caso de los servidores Supermicro, que el tipo de procesador fuera multinúcleo y que la cantidad de memoria RAM y almacenamiento fueran superiores a los de los equipos Optiplex 960.

En la tabla 3.1, se listan todos los componentes físicos empleados en el proyecto. En ésta, se aprecia que los primeros elementos son de fijación, los siguientes son de conexión, enseguida los de comunicación y por último los de cómputo.

Tabla 3.1: Hardware usado en el proyecto.

Cantidad	Equipo	Marca y modelo	Características
1	Rack con patch panel	Hubell Nextframe	-45 unidades de rack -2 organizadores verticales -1 organizador horizontal -48 puertos Ethernet cat 6 en patch panel
1	Anaquel	Genérico	-5 niveles
1	Gabinete	SBE-TECH	-20 unidades de rack -Puertas laterales desmontables -Puerta frontal y posterior con llave
1	Escritorio	Genérico	75cm x 150cm
22 [m]	Canaleta metálica tipo rejilla	Genérica	-2.5"de diámetro interno -Tipo U -Con sujetadores para superficie plana
130 [m]	Cable UTP	-Hubbell -Condumex -Intellinet	-Cat 6, 23 awg -Cat 6, 23 awg -Cat 6, 24 awg
5 [m]	Fibra óptica	033402FT	-62.5/125um
3	Face Plate	Genérico	-4 Salidas para jacks hembra RJ45 cat 6
1	Face Plate	Genérico	-2 Salidas para jacks hembra RJ45 cat 6
58	Plug RJ45	Genérico	-Cat 5e
1	Switch	3Com4060	-6 puertos 10/100/1000BASE-T(13x - 18x) -12 puertos 1000BASE-SX -6 puertos convertidores de interfaz Gigabit
1	Switch	Netgear ProSafe XS708E	-8 puertos 10GBASE-T
2	Switch	Extreme Networks X440-24T	-24 puertos 10/100/1000BASE-T -4 puertos 100/1000BASE-X (SFP) -1 puerto 10/100BASE-T (administración) -1 puerto Serial RJ45 (consola)
1	Firewall	Genérico	-PfSense
2	Computadora Minitorre	Dell Optiplex GX-620	-Intel Pentium 4 y Celeron D -2GB de RAM -80GB de capacidad en disco duro -Tarjeta de red a 1Gbps
4	Servidores	Supermicro X9DRD-iF/LF	-2 Procesadores Intel Xeon E5-2620 a 2.10GHz (12 núcleos con hyper-threding) -32GB de RAM -1TB de capacidad en disco duro -Tarjeta de red a 10Gbps
10	Computadora Minitorre	Dell Optiplex 960	-1 Procesador Intel Core 2 Duo a 2.99GHz -2/4 GB de memoria RAM -80/160/500 GB de capacidad en disco duro -Tarjeta de red a 1Gbps

Después de establecer las características del proyecto, éste se divide en dos etapas, construcción y configuración.

3.2. Etapa de construcción

En esta fase se planteó la distribución de los elementos de soporte; el tendido y terminado de cables; el etiquetado y verificación de los enlaces permanentes y la fijación y conexión de los equipos de cómputo y comunicaciones.

3.2.1. Distribución de los elementos de soporte

Se establecieron cuatro zonas para el sistema distribuido. La zona del rack de distribución, donde se concentra la mayor parte de las conexiones de red; la zona de xexelo0, donde un gabinete soporta a los servidores supermicro; la zona de xexelo1, donde un anaquel sostiene a las computadoras Optiplex 960 y la zona de administración remota, desde donde, a través de los equipos Optiplex GX620, se controlan y verifican los clústeres y los switches (figura 3.2).

Los elementos de soporte se acomodaron de tal forma que quedaran próximos a las tomas de corriente eléctrica y que no afectaran a otros objetos ya establecidos (puertas, mesas y lockers). Además, únicamente se situaron cajas para los nodos de red en la zonas de administración y de xexelo1.

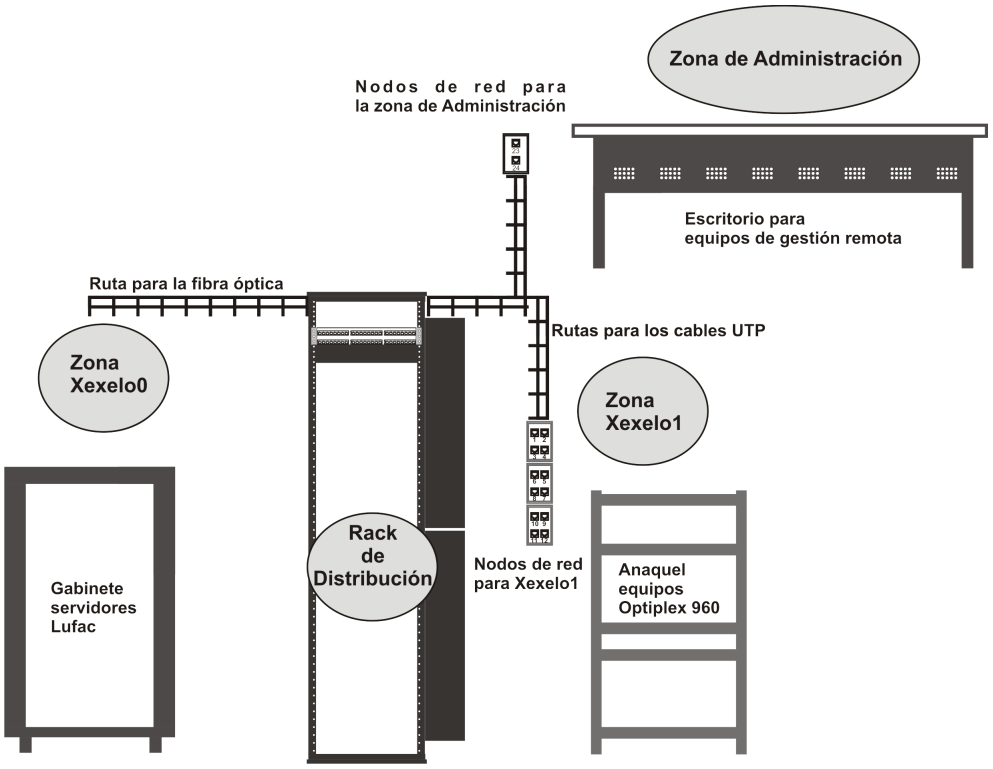


Figura 3.2: Zonas del sistema distribuido.

3.2.2. Tendido y terminación de cables

Para el cableado estructurado se consideró una topología de estrella jerárquica, además, que la distancia entre éste y las rutas de cableado eléctrico fuera de cincuenta centímetros o sus cruces a 90°. Para el peinado de los cables de red se evitó el mayor número de curvas, que éstas no fueran en ángulo recto y que la longitud del cable no excediera los treinta metros (considerando un excedente para futuros cambios). En cuanto a la agrupación de los cables se utilizó velcro (para no dañarlos) y para la terminación de éstos se desentorcharon los cuatro pares de alambre únicamente media vuelta en el extremo del panel de parcheo y cero desentorche del lado de los jacks (figura 3.3).

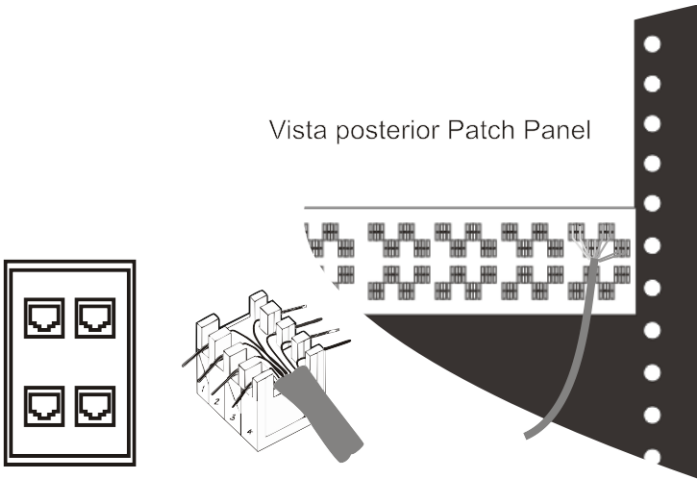


Figura 3.3: Terminación de los enlaces permanentes en el panel de parcheo y en los jacks hembra RJ45.

3.2.3. Etiquetado y verificación de enlaces permanentes

Para identificar cada área del sistema se siguió el estándar de infraestructura de telecomunicaciones para centros de datos EIA/TIA-606-A (figura 3.4). Además se realizó satisfactoriamente la prueba de rendimiento (o Throughput) a los enlaces permanentes del panel A del RACK 4 MDA (*Main Distribution Area* 'Área de Distribución Principal'). Ésta prueba consiste en enviar, desde una fuente hasta un destino, a través de cada enlace permanente, tramas de diferentes tamaños. Éstas tramas son de: 64, 128, 256, 512, 1024, 1280, 1518 y 9600 bytes y la tolerancia de errores o pérdida de tramas se establece en 0 %. El equipo que actuó como fuente de bytes, fue el tester ethernet JDSU HST-3000 y el equipo utilizado como destino fue el tester SmartClass ethernet de JDSU (figura 3.5). Los patch cords empleados están certificados para este tipo de sondeos. Los resultados de la prueba se presentan en el apéndice A de este texto.

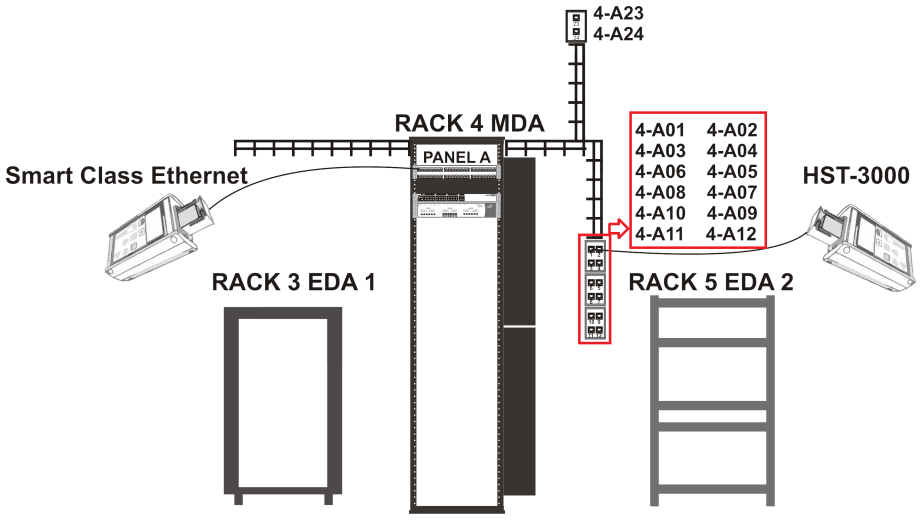


Figura 3.4: La etiqueta 4-A01 significa que el nodo de red pertenece al puerto 01 ubicado en el patch panel A en el rack 4.

Figura 3.5: Equipo JDSU utilizado para verificar los enlaces permanentes. A la izquierda el tester SmartClass. A la derecha equipo HST-3000.

3.2.4. Montaje y conexión de los equipos de cómputo y comunicaciones

En la figura 3.6 se muestra el esquema final del sistema distribuido. En la tabla 3.2 se listan los elementos por rack y en las figuras 3.7 y 3.8 el sistema real.

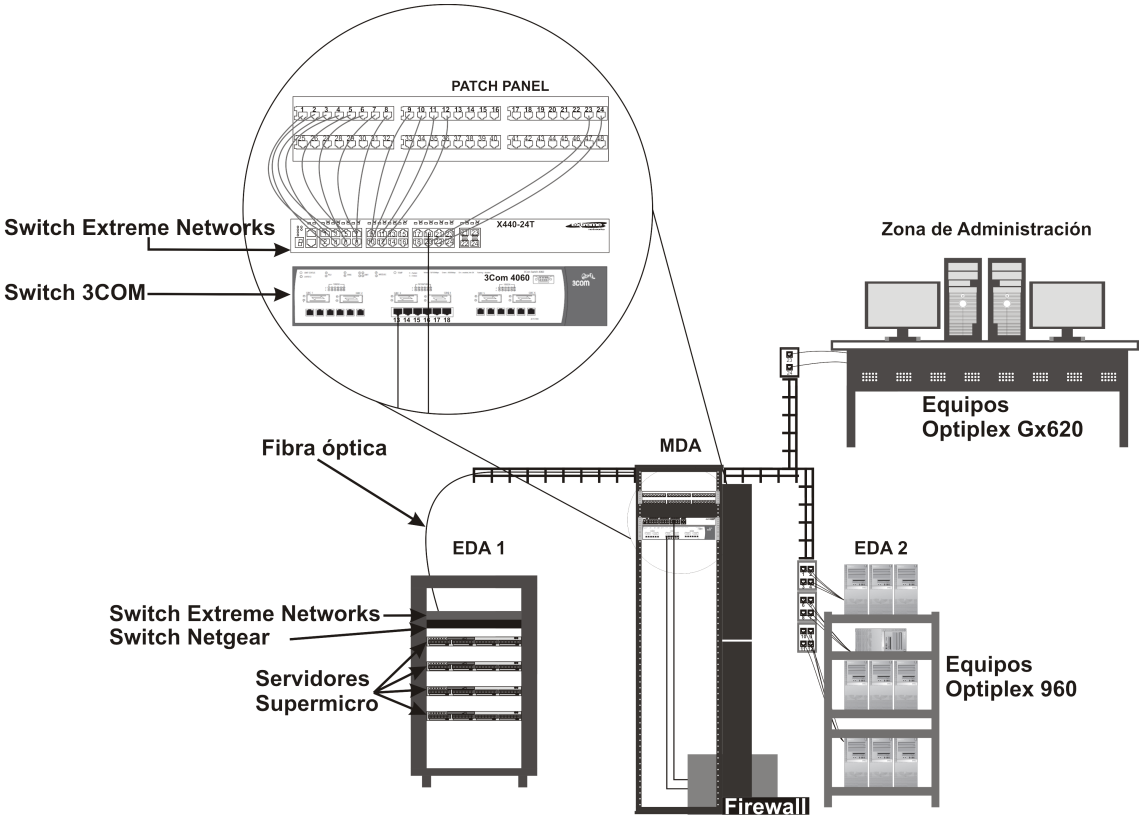


Figura 3.6: Esquema del sistema de cómputo distribuido.

Tabla 3.2: Equipos por rack.

RACK	EQUIPO
RACK 3 EDA 1	4 Servidores Supermicro 1 Switch Extreme Networks X440-24T 1 Switch ProSAFE XS708E NETGEAR
RACK 4 MDA	1 Switch Extreme Networks X440-24T 1 Switch 3 COM 4060
RACK 5 EDA 2	10 Equipos Optiplex 960

Figura 3.7: Nodos de red y área de distribución principal.

Figura 3.8: Clúster xexelo0, zona de administración y clúster xexelo1.

Después de concluir la etapa de implementación y de verificar el cableado estructurado, se describen las instalaciones y los ajustes de software para los nodos de los clústeres, para los equipos de la zona de administración, para el firewall y para los switches.

3.3. Etapa de configuración

Un clúster de cómputo de alto rendimiento (o de cualquier otro tipo) debe tener equipos dedicados y herramientas de software para atención a usuarios, comúnmente vía web; equipos de control para los servicios de asignación de direcciones IP, sistemas de nombres de dominio y sistemas de archivos en red. También deben poseer equipos de monitoreo, para atender las alertas provenientes de los equipos de cómputo y comunicaciones; equipos de almacenamiento, con arreglos RAID; equipos de instalación, con servicios NFS o FTP y equipos de operación con MPI, aplicaciones distribuidas, entre otras [10]. Sin embargo, en este proyecto sólo se incluyeron equipos de operación, de control y de monitoreo. A continuación se describirán los ajustes de software realizados.

3.3.1. Configuración de los clústeres xexelo0 y xexelo1

Para ambos clústeres se utilizó CentOS 6.6 como sistema operativo porque es una plataforma previsible, administrable y reproducible de los repositorios de RHEL (Linux Empresarial Red Hat) orientado a servidores y estaciones de trabajo. Estar basado en Red Hat garantiza un soporte prolongado. En la tabla 3.3 se muestran los parámetros que se establecieron en todos los equipos de cómputo de xexelo0 y xexelo1.

Tabla 3.3: Parámetros de instalación de CentOS.

	XEXELO0	XEXELO1
IDIOMA DEL SISTEMA	Español	Español
IDIOMA DEL TECLADO	Latinoamericano	Latinoamericano
TIPO DE DISPOSITIVO	Almacenamiento básico	Almacenamiento básico
NOMBRE DEL HOST	frontend host01 host02 host03	FrontEnd host01 host02 host03 host04 host05 host06 host07 host08 host09
ZONA HORARIA	América/Ciudad de México	América/Ciudad de México
USO EN DISCO DURO	Usar todo el espacio	Usar todo el espacio
TIPO DE INSTALACIÓN	Mínima	Mínima

Después de la instalación del sistema operativo se configuraron los archivos *ifcfg-identificador* que controlan las interfaces de red de los nodos coordinadores (o frontends) y de los nodos esclavos. Una interfaz sirve para la comunicación con los demás nodos y la otra permite la salida a Internet. Además, en el caso de los servidores de xexelo0, se tiene una interfaz extra que permite el uso de la interfaz IPMI (*Intelligent Platform Management Interface* 'Interface de Administración de Plataforma Inteligente'). En cuanto a los nodos esclavos de xexelo1 solo poseen una interfaz de red y los nodos esclavos de xexelo0 poseen dos interfaces de red. En la tablas 3.4 y 3.5 se muestran las interfaces de los hosts de cada clúster y su plan de

direccionamiento IP. Cabe mencionar que las direcciones IP mostradas en ambas tablas no son las establecidas en los equipos de cómputo, como simple práctica de seguridad.

Tabla 3.4: Interfaces de red xexelo0.

XEXELO0				
#	**Nombre del Host**	**Interfaz de Red**	**Función**	**Dirección IP**
1	Frontend	eth0	Salida a Internet y administración remota	11.0.0.44/16
		eth3	Comunicación entre nodos	5.5.5.2/6
		IPMI	Monitor y administración (web)	5.0.0.1/4
2	host01	eth3	Comunicación entre nodos	5.5.5.3/6
		IPMI	Monitor y administración (web)	5.0.0.2/4
3	host02	eth3	Comunicación entre nodos	5.5.5.4/6
		IPMI	Monitor y administración (web)	5.0.0.3/4
4	host03	eth4	Comunicación entre nodos	5.5.5.5/6
		IPMI	Monitor y administración (web)	5.0.0.4/4

Tabla 3.5: Interfaces de red xexelo1.

XEXELO1				
#	**Nombre del Host**	**Interfaz de Red**	**Función**	**Dirección IP**
1	frontend	eth1	Salida a Internet y administración remota	11.0.0.45/16
		eth0	Comunicación con nodos	6.6.6.1/8
2	host01	eth0	Comunicación con nodos	6.6.6.2/8
3	host02	eth0	Comunicación con nodos	6.6.6.3/8
4	host03	eth0	Comunicación con nodos	6.6.6.4/8
5	host04	eth0	Comunicación con nodos	6.6.6.5/8
6	host05	eth0	Comunicación con nodos	6.6.6.6/8
7	host06	eth0	Comunicación con nodos	6.6.6.7/8
8	host07	eth0	Comunicación con nodos	6.6.6.8/8
9	host08	eth0	Comunicación con nodos	6.6.6.9/8
10	host09	eth0	Comunicación con nodos	6.6.6.10/8

Para que el frontend de cada clúster pudiera resolver las peticiones de red de los demás nodos esclavos, se instaló el paquete *squid*, el cual es un software proxy que habilita a los frontends para fungir como enrutador de servicios http, https, ftp, entre otros, además de actuar como firewall, de reducir los tiempos de acceso a los sitios web más solicitados por los nodos esclavos y de activar y desactivar la conexión a internet automáticamente. También se descomentó la línea *net.ipv4.ip_forward = 1* del archivo */etc/sysctl.conf*, ya que esto habilita la función de traslación de direcciones de red. Por otro lado, se realizó con superusuario y en la línea de comandos, una regla NAT con el fin de bloquear las peticiones directas hacia internet desde los nodos esclavos (figura 3.9). Por otra parte, en los nodos esclavos se deshabilitó el firewall nativo de CentOS con la línea (*system-config-firewall-tui*) debido al bloqueo de puertos de servicios como ssh, telnet, smtp, time, ntp, https, entre otros.

Seguidamente, en todos los nodos, tanto coordinadores como esclavos:
- Se deshabilitó el módulo de seguridad SELinux con la línea *SELINUX = disabled* del archivo */etc/selinux/config* para no complicar la gestión de los permisos de ejecución de servicios.
- Se creó el usuario *tesista* con su respectivo password, perteneciente al grupo *users* porque no se puede utilizar al usuario root para ejecutar tareas en los clústeres.
- Se creó el directorio *openmpi* en la ruta */home/tesista/* para que el frontend y los nodos

esclavos tuvieran un directorio de trabajo compartido.
- Se renovaron los repositorios de CentOS para actualizar las fuentes de más bibliotecas de software (ver Apéndice B).
- Se actualizó y ascendió el sistema operativo, con *#yum -y update* y *#yum -y upgrade* respectivamente, para asegurar que todos los paquetes de software instalados en CentOS se establecieran a la última versión.
- Se instalaron las herramientas de desarrollo *(Development Tools)* que incluye los compiladores *gcc-c++* y *gcc c/c++* para poder programar, compilar y ejecutar las tareas en paralelo.
- Se instaló htop para monitorear el uso de recursos y procesos de todos los nodos de los clústeres.

```
/sbin/iptables -P INPUT ACCEPT
/sbin/iptables -F INPUT
/sbin/iptables -P OUTPUT ACCEPT
/sbin/iptables -F OUTPUT
/sbin/iptables -P FORWARD DROP
/sbin/iptables -F FORWARD
/sbin/iptables -t nat -F
/sbin/iptables -A FORWARD -i (interfaz red local) -o (interfaz Internet) -m state --state ESTABLISHED,RELATED -j ACCEPT
/sbin/iptables -A FORWARD -i (interfaz Internet) -o (interfaz local) -j ACCEPT
/sbin/iptables -t nat -A POSTROUTING -s (red local con prefijo de red xx.xx.xx.xx/xx) –o (interfaz Internet) -j MASQUERADE
iptables -A FORWARD -i (interfaz Internet) -j ACCEPT
iptables -A FORWARD -o (interfaz Internet) -j ACCEPT
/sbin/iptables-save >/etc/sysconfig/iptables
/sbin/service iptables restart
```

Figura 3.9: Regla NAT para los frontends de cada clúster.

Por otro lado, para la comunicación entre los frontends y los demás nodos, se utilizó el servicio ssh y se generó una llave con el algoritmo RSA para evitar escribir la contraseña cada que se ejecute una tarea en paralelo. En la figura 3.10 se muestra la configuración.

```
En los frontends y con el usuario tesista

$ssh-keygen -t rsa
$chmod 700 ~/.ssh
$chmod 600 ~/.ssh/id_rsa
$cat id_rsa.pub >> ~/.ssh/authorized_keys
$scp authorized_keys tesista@5.5.5.x:/home/tesista/.ssh          Xexelo0
$scp authorized_keys tesista@6.6.6.x:/home/tesista/.ssh          Xexelo1

En los nodos esclavo y con el usuario tesista

$chmod 700 ~/.ssh
$chmod 600 ~/.ssh/authorized_keys
```

Figura 3.10: Compartición de llave pública.

En la ejecución de una tarea en paralelo es necesario que todos los nodos tengan un directorio de trabajo en común, así, los frontends comparten el directorio */home/tesista/openmpi* y todos los nodos esclavo lo establecen en el directorio */home/tesista/opnempi* (figura 3.11). El parámetro *rw* habilita la lectura/escritura en el directorio compartido. El parámetro *sync* indica que el servidor sólo reconocerá los datos después de que hayan sido escritos en el directorio. La línea *no_root_squash* habilita la compartición de archivos creados por el usuario

root, desde los nodos cliente. La línea *no_subtree_check* deshabilita la verificación de la ubicación de los archivos a través de la rutina llamada subtree.

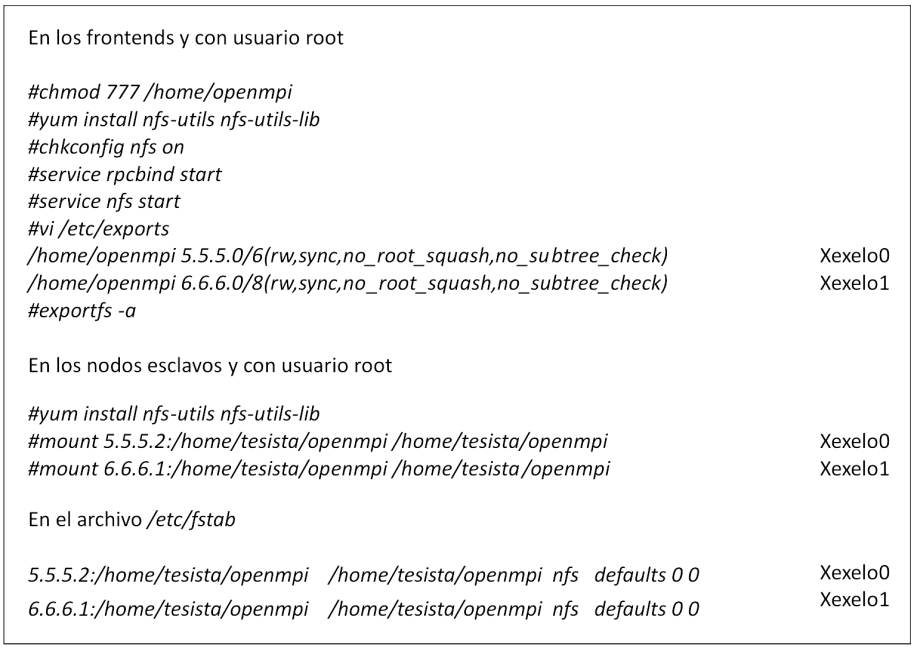


Figura 3.11: Compartición del directorio de trabajo.

El siguiente paso (figura 3.12), es el que hace del conjunto de máquinas un sistema distribuido. Se instala la capa de middleware.

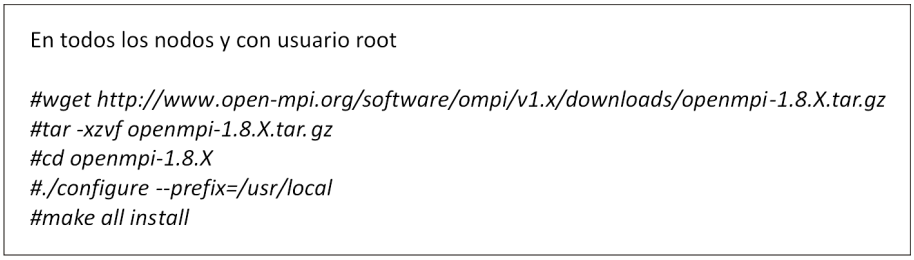


Figura 3.12: Instalación del middleware OpenMPI.

Hasta aquí, la configuración de ambos clústeres es prácticamente igual (sistema operativo y paquetes de software). Sin embargo, los servidores de xexelo0 poseen la interfaz IPMI, la cual permite monitorear el estado físico de los servidores y permite el acceso a las opciones del BIOS. En la figura 3.13 se muestran los parámetros que se deben establecer (según el manual consultado en la dirección electrónica *http://www.supermicro.com/support/manuals/*).

Desde el BIOS de los servidores Supermicro se habilitan las opciones:

-Redirección de consola COM2/SOL.
-Todos los dispositivos USB.
-Se establece la IP y la máscara de red para cada interfaz.

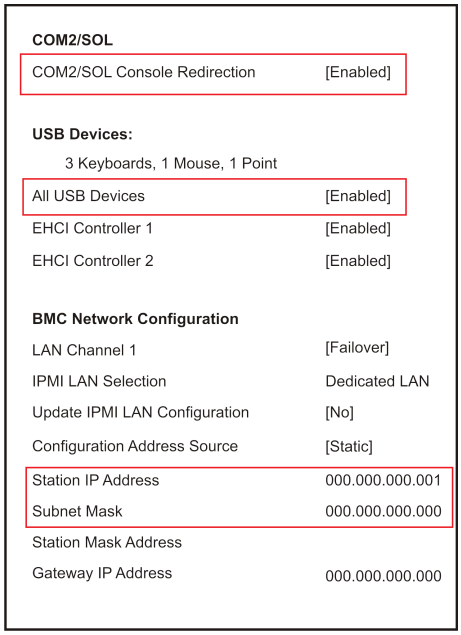


Figura 3.13: Configuración interfaz IPMI.

3.3.2. Configuración de los switches

Respecto a los switches, se configuraron cuatro redes locales virtuales, *administración, xexelo0, xexelo1 e ipmi* (en la tabla 3.6 se observan las vlans por cada switch). Por otra parte, los switches Extreme Networks se configuraron para ser administrados vía web. El switch Netgear sólo permite administrarlo a través de una utilidad ejecutable que usa WinPCap (exclusivo para sistemas operativos Microsoft Windows). En cuanto al switch 3COM sólo se habilitó la administración a través del puerto de consola RS232. La forma de cómo se conectaron los switches se muestra en la figura 3.14.

Tabla 3.6: Switches y redes locales virtuales.

Switch	VLANs	Ubicación
3Com 4060	administracion	Rack 4 MDA
Extreme Networks X440-24T	administracion xexelo1	Rack 4 MDA
Extreme Networks X440-24T	administracion ipmi	Rack 3 EDA 1
Netgear Prosafe XS708E	administracion xexelo0	Rack EDA 1

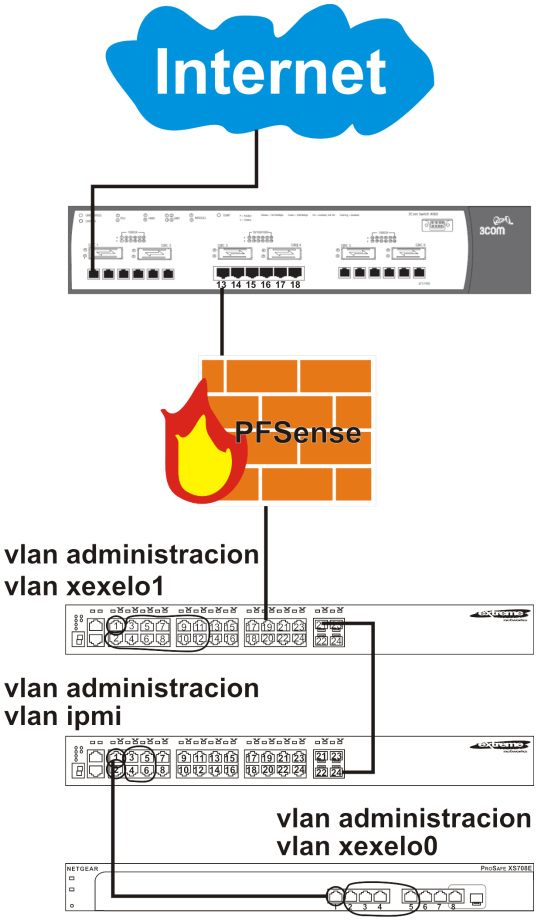


Figura 3.14: VLANs

3.3.3. Configuración de los equipos de administración

En los equipos de administración remota se instaló la versión 6.6 de CentOS con entorno gráfico; se actualizaron los repositorios para robustecer las fuentes de búsqueda de software; se instaló el software VirtualBox de Oracle (figura 3.15) para ejecutar la máquina virtual de windows xp y poder controlar el switch Netgear (figura 3.16). También se instaló el software Icedtea-Web, con el fin de conectarse a la interfaz de administración de IPMI.

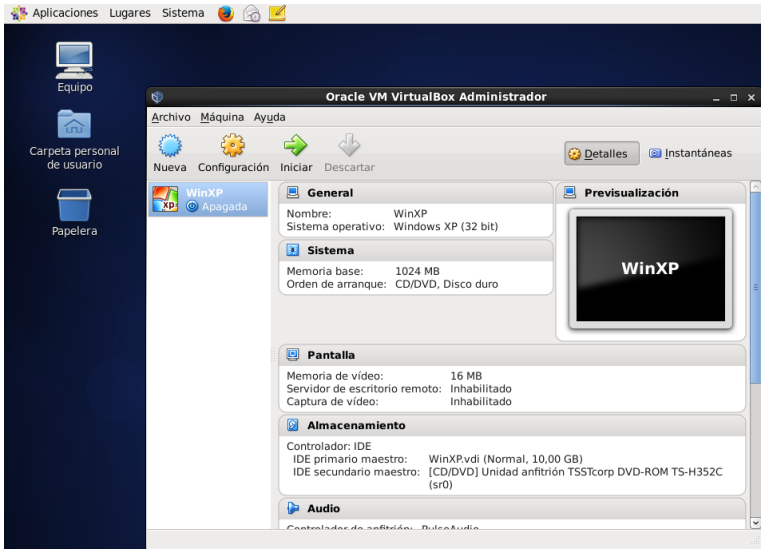


Figura 3.15: Software VirtualBox instalado sobre CentOS.

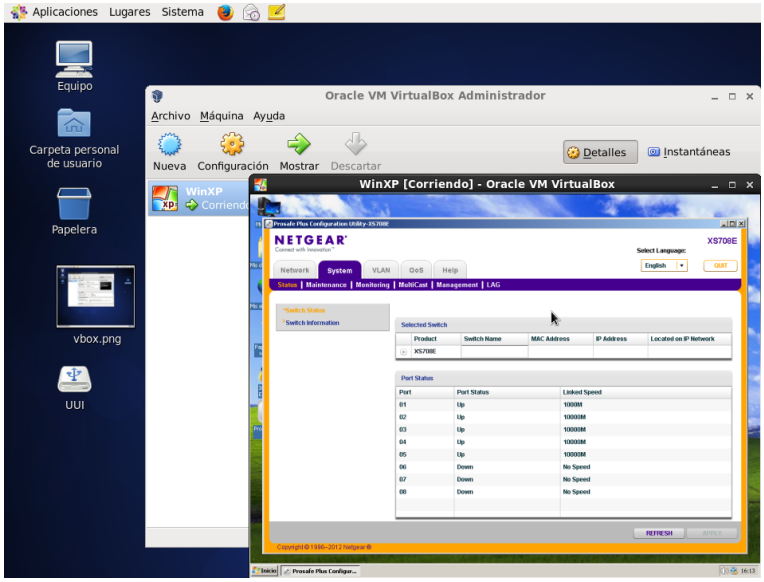


Figura 3.16: Máquina virtual windows XP y software del switch Netgear.

Para controlar todos los equipos de cómputo y comunicaciones del sistema distribuido, se crearon cuatro conexiones de red lógicas en los equipos de administración remota (figura 3.17). La conexión *Internet* permite la salida al servicio de internet; la conexión *IPMI* es para controlar y monitorear los servidores Supermicro de xexelo0; la conexión *ExtremeNet_MDA* permite el enlace al switch de Extreme Networks ubicado en el área de distribución principal y la conexión *ExtremeNetworks_EDA_1* permite el enlace al segundo switch Extreme Networks ubicado en el área de distribución de equipos. Cabe mencionar que también se puede configurar un redireccionamiento de puertos en el firewall PFSense para que, en algún browser, al escribir la dirección IP asignada a cada switch, automáticamente gestione la comunicación con éste.

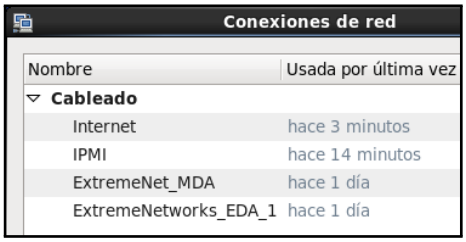
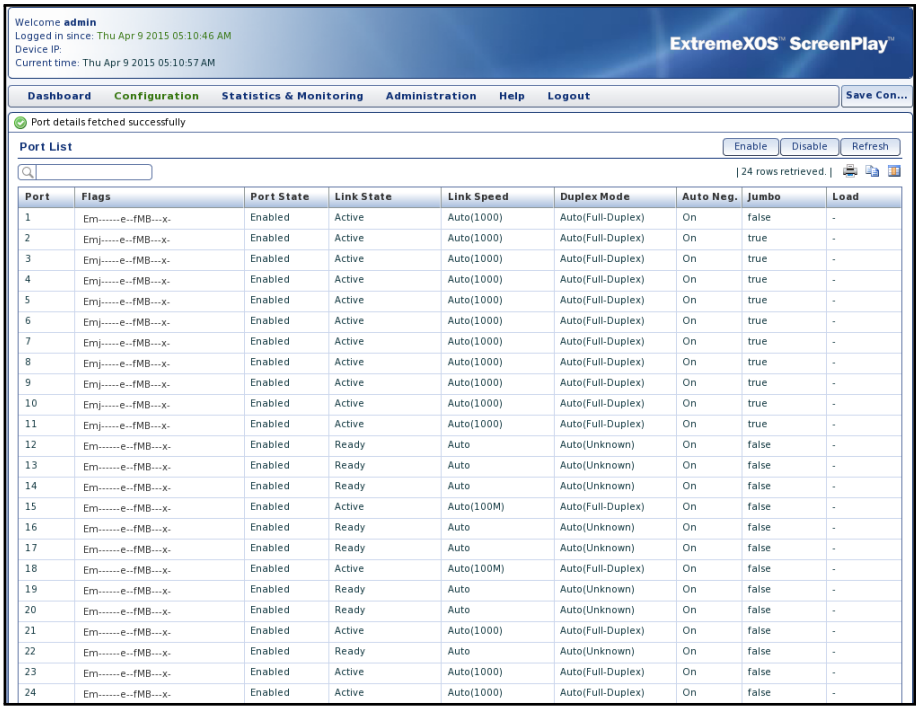


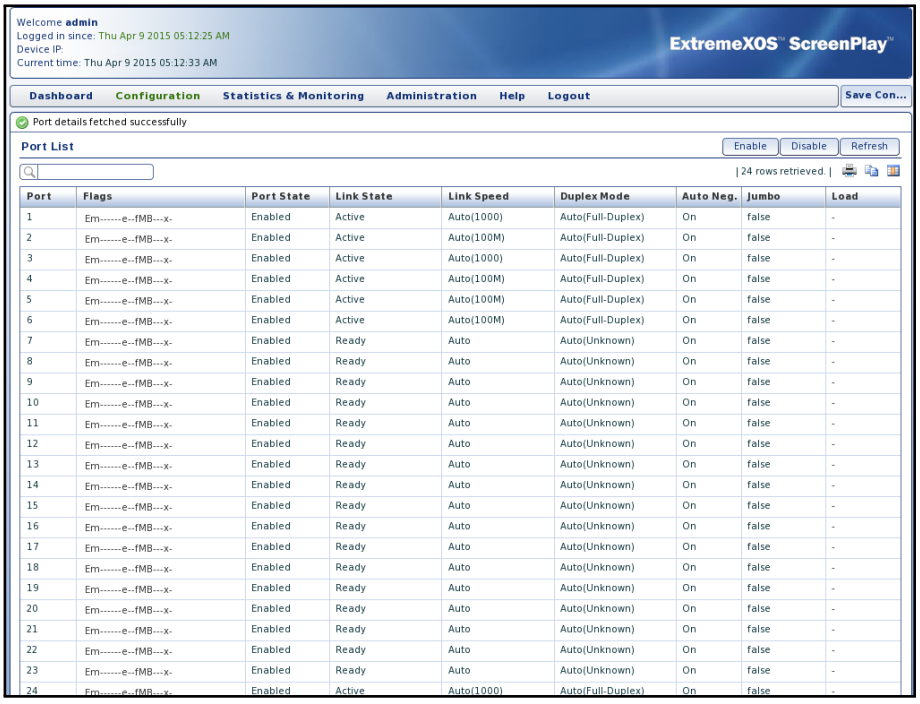
Figura 3.17: Conexiones de red para administrar los equipos de cómputo y los switches del sistema de cómputo distribuido.

En la figura 3.18 y 3.19 se muestra el monitoreo vía web de los switches Extreme Networks (desde los equipos de administración). A través de estas interfaces se pueden consultar si están habilitados todos los puertos del switch, si están activos los enlaces de red, las tasas de transferencia, el modo de transmisión de datos, entre otros.



Port	Flags	Port State	Link State	Link Speed	Duplex Mode	Auto Neg.	Jumbo	Load
1	Em------e--fMB---x-	Enabled	Active	Auto(1000)	Auto(Full-Duplex)	On	false	-
2	Emj-----e--fMB---x-	Enabled	Active	Auto(1000)	Auto(Full-Duplex)	On	true	-
3	Emj-----e--fMB---x-	Enabled	Active	Auto(1000)	Auto(Full-Duplex)	On	true	-
4	Emj-----e--fMB---x-	Enabled	Active	Auto(1000)	Auto(Full-Duplex)	On	true	-
5	Emj-----e--fMB---x-	Enabled	Active	Auto(1000)	Auto(Full-Duplex)	On	true	-
6	Emj-----e--fMB---x-	Enabled	Active	Auto(1000)	Auto(Full-Duplex)	On	true	-
7	Emj-----e--fMB---x-	Enabled	Active	Auto(1000)	Auto(Full-Duplex)	On	true	-
8	Emj-----e--fMB---x-	Enabled	Active	Auto(1000)	Auto(Full-Duplex)	On	true	-
9	Emj-----e--fMB---x-	Enabled	Active	Auto(1000)	Auto(Full-Duplex)	On	true	-
10	Emj-----e--fMB---x-	Enabled	Active	Auto(1000)	Auto(Full-Duplex)	On	true	-
11	Emj-----e--fMB---x-	Enabled	Active	Auto(1000)	Auto(Full-Duplex)	On	true	-
12	Em------e--fMB---x-	Enabled	Ready	Auto	Auto(Unknown)	On	false	-
13	Em------e--fMB---x-	Enabled	Ready	Auto	Auto(Unknown)	On	false	-
14	Em------e--fMB---x-	Enabled	Ready	Auto	Auto(Unknown)	On	false	-
15	Em------e--fMB---x-	Enabled	Active	Auto(100M)	Auto(Full-Duplex)	On	false	-
16	Em------e--fMB---x-	Enabled	Ready	Auto	Auto(Unknown)	On	false	-
17	Em------e--fMB---x-	Enabled	Ready	Auto	Auto(Unknown)	On	false	-
18	Em------e--fMB---x-	Enabled	Active	Auto(100M)	Auto(Full-Duplex)	On	false	-
19	Em------e--fMB---x-	Enabled	Ready	Auto	Auto(Unknown)	On	false	-
20	Em------e--fMB---x-	Enabled	Ready	Auto	Auto(Unknown)	On	false	-
21	Em------e--fMB---x-	Enabled	Active	Auto(1000)	Auto(Full-Duplex)	On	false	-
22	Em------e--fMB---x-	Enabled	Ready	Auto	Auto(Unknown)	On	false	-
23	Em------e--fMB---x-	Enabled	Active	Auto(1000)	Auto(Full-Duplex)	On	false	-
24	Em------e--fMB---x-	Enabled	Active	Auto(1000)	Auto(Full-Duplex)	On	false	-

Figura 3.18: Monitoreo del switch Extreme Networks ubicado en el MDA.



Port	Flags	Port State	Link State	Link Speed	Duplex Mode	Auto Neg.	Jumbo	Load
1	Em------e--fMB---x-	Enabled	Active	Auto(1000)	Auto(Full-Duplex)	On	false	-
2	Em------e--fMB---x-	Enabled	Active	Auto(100M)	Auto(Full-Duplex)	On	false	-
3	Em------e--fMB---x-	Enabled	Active	Auto(1000)	Auto(Full-Duplex)	On	false	-
4	Em------e--fMB---x-	Enabled	Active	Auto(100M)	Auto(Full-Duplex)	On	false	-
5	Em------e--fMB---x-	Enabled	Active	Auto(100M)	Auto(Full-Duplex)	On	false	-
6	Em------e--fMB---x-	Enabled	Active	Auto(100M)	Auto(Full-Duplex)	On	false	-
7	Em------e--fMB---x-	Enabled	Ready	Auto	Auto(Unknown)	On	false	-
8	Em------e--fMB---x-	Enabled	Ready	Auto	Auto(Unknown)	On	false	-
9	Em------e--fMB---x-	Enabled	Ready	Auto	Auto(Unknown)	On	false	-
10	Em------e--fMB---x-	Enabled	Ready	Auto	Auto(Unknown)	On	false	-
11	Em------e--fMB---x-	Enabled	Ready	Auto	Auto(Unknown)	On	false	-
12	Em------e--fMB---x-	Enabled	Ready	Auto	Auto(Unknown)	On	false	-
13	Em------e--fMB---x-	Enabled	Ready	Auto	Auto(Unknown)	On	false	-
14	Em------e--fMB---x-	Enabled	Ready	Auto	Auto(Unknown)	On	false	-
15	Em------e--fMB---x-	Enabled	Ready	Auto	Auto(Unknown)	On	false	-
16	Em------e--fMB---x-	Enabled	Ready	Auto	Auto(Unknown)	On	false	-
17	Em------e--fMB---x-	Enabled	Ready	Auto	Auto(Unknown)	On	false	-
18	Em------e--fMB---x-	Enabled	Ready	Auto	Auto(Unknown)	On	false	-
19	Em------e--fMB---x-	Enabled	Ready	Auto	Auto(Unknown)	On	false	-
20	Em------e--fMB---x-	Enabled	Ready	Auto	Auto(Unknown)	On	false	-
21	Em------e--fMB---x-	Enabled	Ready	Auto	Auto(Unknown)	On	false	-
22	Em------e--fMB---x-	Enabled	Ready	Auto	Auto(Unknown)	On	false	-
23	Em------e--fMB---x-	Enabled	Ready	Auto	Auto(Unknown)	On	false	-
24	Em------e--fMB---x-	Enabled	Active	Auto(1000)	Auto(Full-Duplex)	On	false	-

Figura 3.19: Monitoreo del switch Extreme Networks ubicado en el EDA 1.

Por otro lado, en la figura 3.20 se aprecian las lecturas de los sensores de los cuatro servidores Supermicro de xexelo0. El campo nombre, el campo estado y una etiqueta en color verde facilitan la comprobación del estado físico de los cuatro servidores.

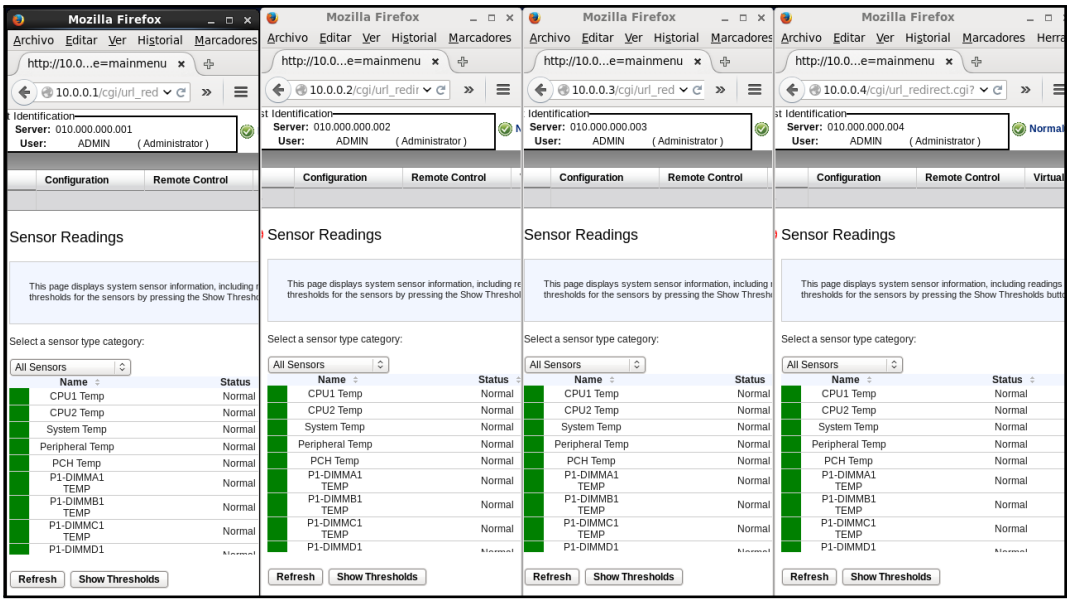


Figura 3.20: Acceso a la interfaz de monitoreo de los servidores Supermicro.

En caso de necesitar controlar directamente todos los servidores de xexelo0, se pueden iniciar sesiones remotas utilizando el servicio ssh o mediante http y una máquina cliente de Java (interfaz IPMI). La ventaja de IPMI es que la experiencia de control es más completa porque se puede acceder a las opciones del BIOS de manera remota (figura 3.21).

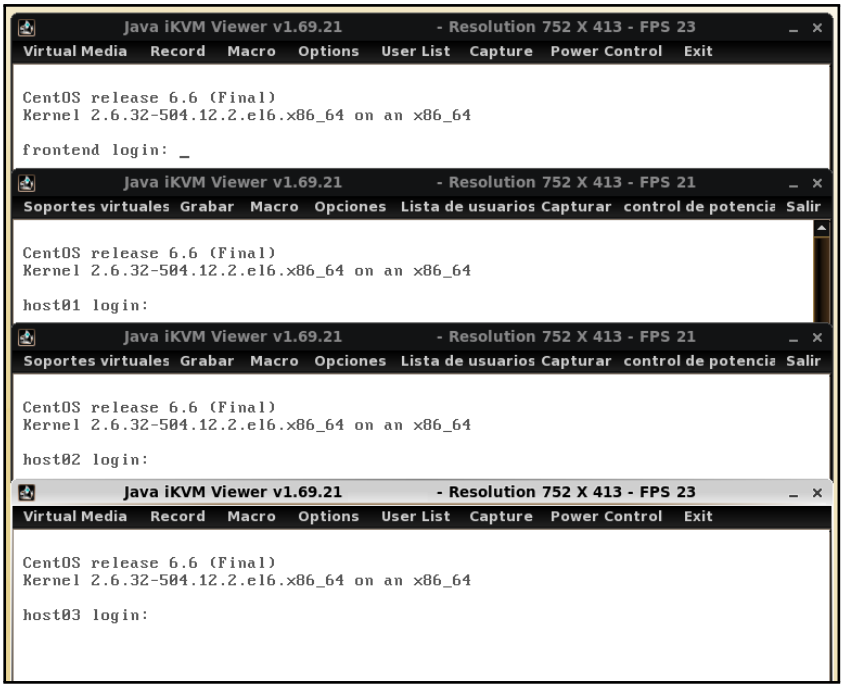


Figura 3.21: Administración de los servidores Supermicro.

3.3.4. Configuración del firewall

Existen herramientas de software libre para brindar seguridad perimetral como IPcop, Shorewall, UFW, entre otras. Sin embargo, se empleó la distribución personalizada *PFSense* de FreeBSD. Ésta actúa como firewall y router. Se eligió porque posee las mismas funcionalidades (o incluso más) que los firewalls comerciales más populares, por ejemplo Check Point, Cisco, PIX, Cisco ASA, Juniper, entre otros [20].

Algunas de las características de PFSense: gracias a su asistente paso a paso, la instalación es sencilla; no requiere controladores especiales para diferentes arquitecturas de hardware; tiene la opción de redireccionamiento de puertos y servidor DHCP, por mencionar algunas.

El equipo donde se instaló PFSense posee dos tarjetas de red a 1 Gbps. La primera de ellas gestiona las entradas de paquetes provenientes de internet. La segunda tarjeta resuelve todas las peticiones provenientes de la zona de administración y de los nodos coordinadores de los clústeres.

Para las interfaces del firewall se establecieron dos reglas básicas. Del lado de los clústeres (red LAN) todas las peticiones hacia cualquier puerto de cualquier red (LAN o WAN) son permitidas y del lado de la salida a internet (red WAN) se bloquean todas las peticiones y conexiones provenientes de redes privadas (figura 3.22).

En el enlace *https://doc.pfsense.org/index.php/Main_Page* se puede encontrar toda la información referente a la configuración de pfSense.

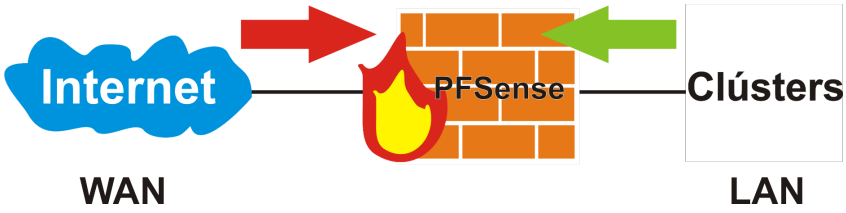


Figura 3.22: Interfaces del firewall.

Capítulo 4
Pruebas de funcionamiento del sistema de cómputo distribuido

En este capítulo se describen las condiciones necesarias para ejecutar tareas en paralelo en los clústeres xexelo0 y xexelo1. También se muestran los tiempos de solución contra el número de procesos del producto de dos matrices cuadradas de 9000 X 9000 (números enteros), la cual implica un billón cuatrocientos cincuenta y siete mil novecientos diecinueve millones de operaciones (repartidas en multiplicaciones y sumas). El programa de solución de matrices en paralelo se describe en el apéndice D. Cabe señalar que las matrices a multiplicar fueron de 9000 X 9000 porque la dimensión del arreglo se ajusta al tamaño de bloque tolerado para arquitecturas x86_64, de otro modo se obtendría un error de reubicación de datos truncado [21]. Por otra parte, se presenta el valor aproximado del poder de cómputo del clúster xexelo1, el cual se obtuvo al ejecutar una adaptación de la prueba de Linpack Benchmark (Prueba de Referencia Linkpack).

Aunado a lo anterior, se observó que emplear una conexión de 10 Gbps y 1 Gbps en xexelo0 y xexelo1, respectivamente, no representó una enorme diferencia en el rendimiento de los clústeres, dado que las tasas de transferencia de los switches son más rápidas que la frecuencia de reloj a la que trabajan los procesadores de los servidores de xexelo0 y los equipos de cómputo de xexelo1, por lo tanto, el middleware, a través de la capa de enlace de datos, controla el flujo de información para no saturar a los nodos coordinadores.

4.1. Condiciones de funcionamiento

Como se mencionó en el capítulo tres, un único nodo de cada clúster actúa como frontend (o nodo coordinador) y desde éste se ejecutan todas las tareas paralelas. Sin embargo, el control de los nodos coordinadores empieza desde la zona de administración y después hacia los nodos esclavos (figura 4.1).

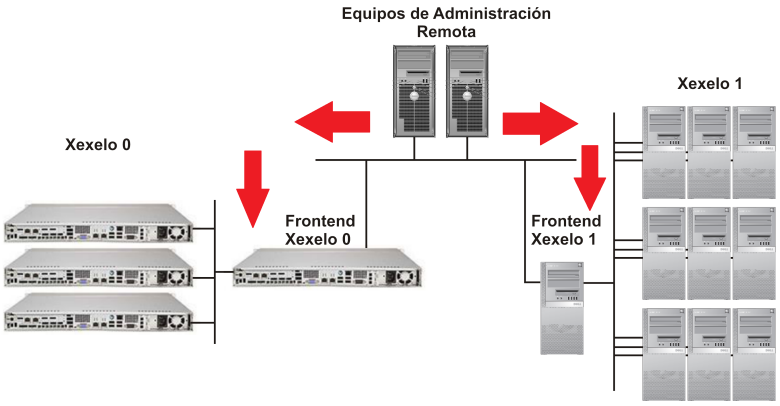


Figura 4.1: Equipos de administración comunicándose con los nodos maestro de cada clúster.

Para ejecutar tareas en paralelo es necesario:

-Iniciar una sesión en el nodo coordinador (de xexelo0 o xexelo1), desde un equipo de administración remota.

-Verificar que todos los nodos de los clústeres puedan acceder al directorio compartido a través del sistema de archivos por red NFS y que al escribir el comando *ompi_info* se muestre el resumen de instalación y configuración de OpenMPI.

-Crear un archivo de texto que contenga las direcciones IP de todos los nodos en el directorio */home/tesista/openmpi* (figura 4.2).

-En el mismo directorio deben estar los programas en lenguaje C que desean ejecutarse.

-Compilar los archivos en lenguaje C con el comando: *mpicc codigo_fuente.c -o ejecutable* (donde *codigo_fuente.c* es el archivo programado y *ejecutable* es el programa compilado y que se ejecutará en todo el clúster).

-Por último ejecutar, desde el frontend, los programas con la línea: *mpirun -hostfile hosts -np X ejecutable* (donde X es el número de procesos o procesadores deseado).

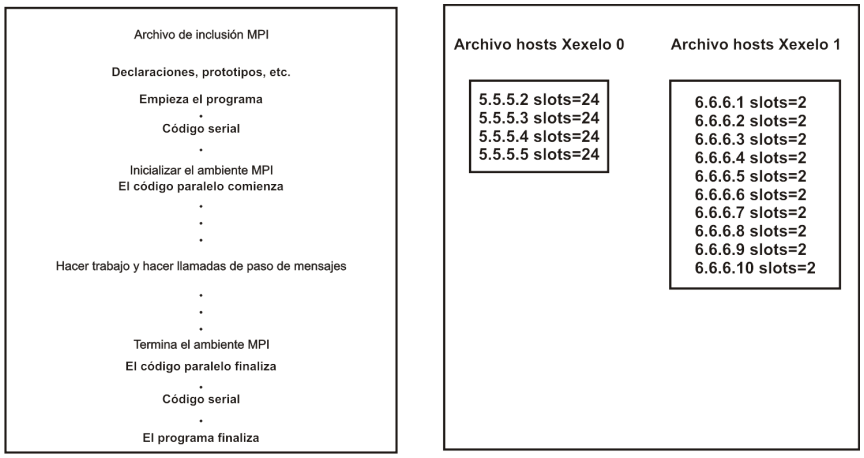


Figura 4.2: Estructura de un programa paralelizado y direcciones IP de los nodos de los clústeres.

Antes de ejecutar la matriz de 9000 X 9000 y la prueba de Linpack, se utilizó el programa holamundo (de *http://people.sc.fsu.edu/jburkardt/c_src/hello_mpi/hello_mpi.c*). Este programa detecta el número de núcleos por procesador y les ordena presentar un mensaje en pantalla (figura 4.3). Los códigos fuente de los programas se pueden consultar en el apéndice D y E.

```
Tenemos 20 procesadores, yo soy el proceso maestro:0 en maquina:FrontEnd
¡Holaaa 1! Proceso 1 en maquina:FrontEnd reportandose
¡Holaaa 2! Proceso 2 en maquina:host01 reportandose
¡Holaaa 3! Proceso 3 en maquina:host01 reportandose
¡Holaaa 4! Proceso 4 en maquina:host02 reportandose
¡Holaaa 5! Proceso 5 en maquina:host02 reportandose
¡Holaaa 6! Proceso 6 en maquina:host03 reportandose
¡Holaaa 7! Proceso 7 en maquina:host03 reportandose
¡Holaaa 8! Proceso 8 en maquina:host04 reportandose
¡Holaaa 9! Proceso 9 en maquina:host04 reportandose
¡Holaaa 10! Proceso 10 en maquina:host05 reportandose
¡Holaaa 11! Proceso 11 en maquina:host05 reportandose
¡Holaaa 12! Proceso 12 en maquina:host06 reportandose
¡Holaaa 13! Proceso 13 en maquina:host06 reportandose
¡Holaaa 14! Proceso 14 en maquina:host07 reportandose
¡Holaaa 15! Proceso 15 en maquina:host07 reportandose
¡Holaaa 16! Proceso 16 en maquina:host08 reportandose
¡Holaaa 17! Proceso 17 en maquina:host08 reportandose
¡Holaaa 18! Proceso 18 en maquina:host09 reportandose
¡Holaaa 19! Proceso 19 en maquina:host09 reportandose
```

(a) Veinte procesos disponibles para xexelo1.

```
Tenemos 48 procesadores, yo soy el proceso maestro:0 en maquina:host03
¡Holaaa 1! Proceso 1 en maquina:host03 reportandose
¡Holaaa 2! Proceso 2 en maquina:host03 reportandose
¡Holaaa 3! Proceso 3 en maquina:host03 reportandose
¡Holaaa 4! Proceso 4 en maquina:host03 reportandose
¡Holaaa 5! Proceso 5 en maquina:host03 reportandose
¡Holaaa 6! Proceso 6 en maquina:host03 reportandose
¡Holaaa 7! Proceso 7 en maquina:host03 reportandose
¡Holaaa 8! Proceso 8 en maquina:host03 reportandose
¡Holaaa 9! Proceso 9 en maquina:host03 reportandose
¡Holaaa 10! Proceso 10 en maquina:host03 reportandose
¡Holaaa 11! Proceso 11 en maquina:host03 reportandose
¡Holaaa 12! Proceso 12 en maquina:frontend reportandose
¡Holaaa 13! Proceso 13 en maquina:frontend reportandose
¡Holaaa 14! Proceso 14 en maquina:frontend reportandose
¡Holaaa 15! Proceso 15 en maquina:frontend reportandose
¡Holaaa 16! Proceso 16 en maquina:frontend reportandose
¡Holaaa 17! Proceso 17 en maquina:frontend reportandose
¡Holaaa 18! Proceso 18 en maquina:frontend reportandose
¡Holaaa 19! Proceso 19 en maquina:frontend reportandose
¡Holaaa 20! Proceso 20 en maquina:frontend reportandose
¡Holaaa 21! Proceso 21 en maquina:frontend reportandose
¡Holaaa 22! Proceso 22 en maquina:frontend reportandose
¡Holaaa 23! Proceso 23 en maquina:frontend reportandose
¡Holaaa 24! Proceso 24 en maquina:host01 reportandose
¡Holaaa 25! Proceso 25 en maquina:host01 reportandose
¡Holaaa 26! Proceso 26 en maquina:host01 reportandose
¡Holaaa 27! Proceso 27 en maquina:host01 reportandose
¡Holaaa 28! Proceso 28 en maquina:host01 reportandose
¡Holaaa 29! Proceso 29 en maquina:host01 reportandose
¡Holaaa 30! Proceso 30 en maquina:host01 reportandose
¡Holaaa 31! Proceso 31 en maquina:host01 reportandose
¡Holaaa 32! Proceso 32 en maquina:host01 reportandose
¡Holaaa 33! Proceso 33 en maquina:host01 reportandose
¡Holaaa 34! Proceso 34 en maquina:host01 reportandose
¡Holaaa 35! Proceso 35 en maquina:host01 reportandose
¡Holaaa 36! Proceso 36 en maquina:host02 reportandose
¡Holaaa 37! Proceso 37 en maquina:host02 reportandose
¡Holaaa 38! Proceso 38 en maquina:host02 reportandose
¡Holaaa 39! Proceso 39 en maquina:host02 reportandose
¡Holaaa 40! Proceso 40 en maquina:host02 reportandose
¡Holaaa 41! Proceso 41 en maquina:host02 reportandose
¡Holaaa 42! Proceso 42 en maquina:host02 reportandose
¡Holaaa 43! Proceso 43 en maquina:host02 reportandose
¡Holaaa 44! Proceso 44 en maquina:host02 reportandose
¡Holaaa 45! Proceso 45 en maquina:host02 reportandose
¡Holaaa 46! Proceso 46 en maquina:host02 reportandose
¡Holaaa 47! Proceso 47 en maquina:host02 reportandose
```

(b) Cuarenta y ocho procesos disponibles para xexelo0 (sin usar HyperThreding).

Figura 4.3: Resultado de ejecutar el programa holamundo en xexelo1 (a) y xexelo0 (b).

4.2. Producto de dos matrices de 9000 X 9000

Después de comprobar que todos los núcleos de todos los procesadores de ambos clústeres responden adecuadamente, se compila y ejecuta el programa productomatrices.c.

A la línea de ejecución ***mpirun -hostfile hosts -np X productomatrices*** se le añade al inicio el comando ***time***, el cual ayudará a determinar la duración de ejecución del programa (*tiempo total o tiempo tope*). También se agrega al final de la línea de ejecución el modificador explícito > ***outmatrices***, el cual ordena que la matriz resultante se almacene en un archivo de texto plano. Así, el producto de las dos matrices se ejecuta como sigue:

time mpirun -hostfile hosts -np X productomatrices > outmatrices.

Donde la ***X*** varía de 1 hasta 96 para xexelo0 y de 1 hasta 20 para xexelo1. Cabe mencionar que en el caso particular de X igual a 1, se utilizó el programa ***serial.c*** (ver apéndice E), el cual es una modificación del programa ***productomatrices.c***. La diferencia es que el primero no utiliza ningún método de paralelización para resolver las matrices, es decir, la estructura de programación es 100 % serial.

En la tabla 4.1, se muestran los valores de tiempo obtenidos para una cierta cantidad de procesos. La ejecución del programa se realizó tres y después se calculó el promedio de los tiempos obtenidos. Se resaltan las filas al utilizar un solo procesador y en donde se alcanzaron los valores de tiempo más pequeños. El factor de speedup se logró con la ecuación 2.3 del capítulo 2.

Tabla 4.1: Valores obtenidos en la evaluación del producto de matrices.

xexelo0			xexelo1		
Procesos	**Tiempo en Minutos**	**Speedup aproximado**	**Procesos**	**Tiempo en Minutos**	**Speedup aproximado**
1	105.141	-	1	108.653	-
2	119.979	0	2	115.810	0
5	34.851	3	3	61.736	1
10	19.396	5	4	44.559	2
15	14.425	7	5	33.561	3
20	11.887	8	6	28.046	3
25	10.167	10	7	24.133	4
30	9.085	11	8	21.539	5
35	8.359	12	9	19.655	5
40	8.058	13	10	17.657	6
45	7.630	13	11	16.619	6
50	10.113	10	12	15.521	7
55	10.161	10	13	14.703	7
60	9.631	10	14	13.564	8
65	9.457	11	15	13.444	8
70	8.897	11	16	12.480	8
72	9.668	10	17	12.197	8
84	8.371	12	18	11.637	9
90	7.214	14	19	11.386	9
96	8.969	11	20	11.209	9

En la figura 4.4 se muestran las cantidades obtenidas para xexelo0. Se advierte que el tiempo máximo de ejecución fue de 119.979 minutos y el tiempo mínimo de 7.214 minutos.

Por otra parte, en la figura 4.5 se observan las cifras conseguidas para xexelo1. Se aprecia que el tiempo máximo de ejecución fue de 115.810 minutos y el tiempo mínimo de 11.209 minutos.

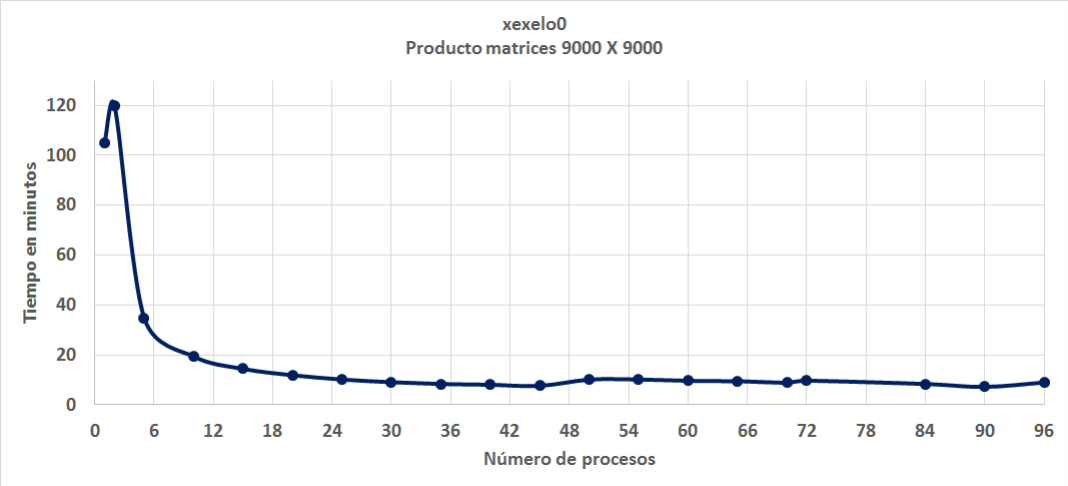


Figura 4.4: Número de procesos versus tiempo de solución de una matriz cuadrada de 9000 X 9000 ejecutada en xexelo0.

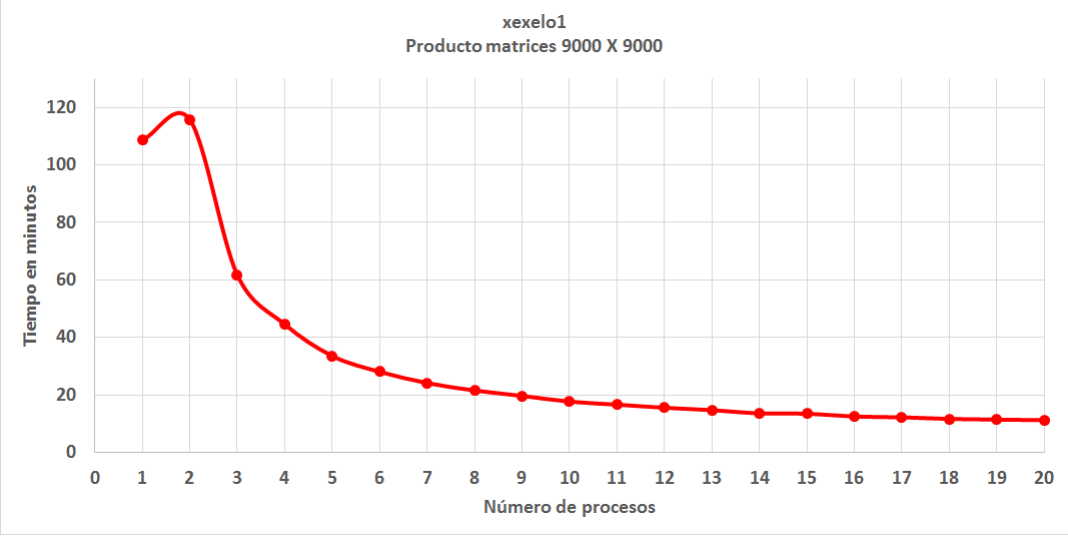


Figura 4.5: Número de procesos versus tiempo de solución de una matriz cuadrada de 9000 X 9000 ejecutada en xexelo1.

Es importante resaltar el comportamiento dispar en xexelo0. Como se aprecia en la tabla 4.1, utilizando cuarenta y cinco procesos se obtuvo un tiempo de respuesta de 7.630 minutos y utilizando cincuenta procesos un tiempo de 10.113 minutos. Más tarde, utilizando setenta procesos, se redujo nuevamente el tiempo a 8.897 minutos. Es decir, la disminución del tiempo no fue proporcional al uso de más procesadores. Una posible explicación es la configuración de OpenMPI con respecto a la arquitectura de los procesadores Xeon E5-2620 y específicamente en la opción Hyper-Threading (Multi-Hilo). Aunado a lo anterior, al monitorear con *htop* el porcentaje de utilización de cada núcleo del clúster, se observó que en cierto momento de la ejecución, se utilizaban en forma serial las unidades de procesamiento (de la 1 a la 46). Después, al utilizar más procesos, aquel orden ya no se mantenía, sino que las unidades de procesamiento se activaban de manera aleatoria para realizar las operaciones. En la gráfica 4.6 se comparan el comportamiento de ambos clústeres.

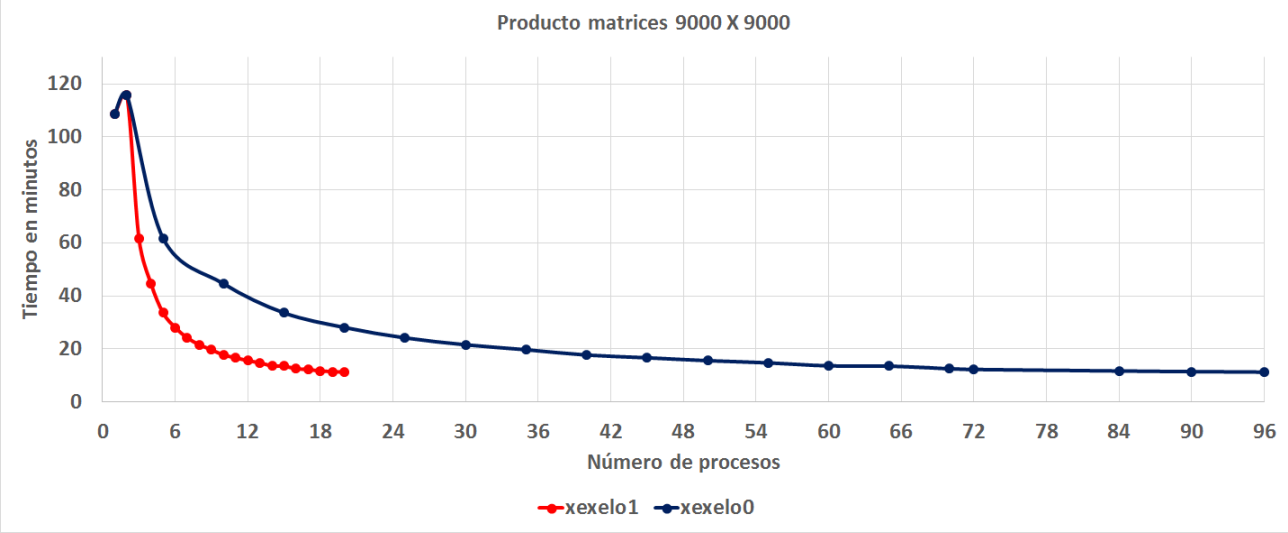


Figura 4.6: Comparación entre xexelo0 (línea roja) y xexelo1 (línea azul).

4.3. Prueba de Linpack en xexelo1

Ahora, para calcular el poder de cómputo de una computadora paralela, existe una prueba que evalúa el número de operaciones de punto flotante por segundo que esta puede realizar. Consiste en resolver un sistema denso de ecuaciones lineales a través de la factorización LU con pivoteo parcial. Ésta prueba se hace llamar Linpack Benchmark y fue introducida por Jack Dongarra, distinguido profesor norteamericano de ciencias de la computación en la Universidad de Tennessee.

Para las computadoras de memoria distribuida, existe una implementación portable de la prueba de Linpack HPL (*High-Performance Linpack Benchmark* 'Prueba de Referencia Linpack de Alto Rendimiento'). Para utilizarla es necesario tener una distribución de la Interfaz de Paso de Mensajes, en este proyecto se tiene OpenMPI, y una distribución del software BLAS (*Basic Linear Algebra Subprograms* 'Subprogramas Básicos de Álgebra Líneal'), el cual consiste de subrutinas que proveen un estándar de construcción de bloques para resolver operaciones de vector básico y matrices. Para este trabajo se utilizó la implementación Goto BLAS, desarrollado por el Centro de Cómputo Avanzado de Texas como software de código abierto bajo la licencia BSD (*Berkeley Software Distribution* 'Distribución de Software Berkeley').

Las instrucciones de cómo instalar, configurar y ejecutar la prueba de Linpack se encuentran en el apéndice F de este documento.

En la figura 4.7 se observa que el clúster xexelo1 tiene un poder de cómputo de 150.9 Gflops, es decir, resuelve aproximadamente ciento cincuenta mil millones de operaciones de punto flotante por segundo.

Por otra parte, la prueba de Linpack no se logró realizar en el clúster xexelo0 debido a que las bibliotecas de HPL no contienen los archivos que controlan la forma de realizar las operaciones lineales de los procesadores multinúcleo Xeon, por lo tanto, esta prueba se puede considerar para trabajos futuros.

```
================================================================================
HPLinpack 2.1  --  High-Performance Linpack benchmark  --   October 26, 2012
Written by A. Petitet and R. Clint Whaley,  Innovative Computing Laboratory, UTK
Modified by Piotr Luszczek, Innovative Computing Laboratory, UTK
Modified by Julien Langou, University of Colorado Denver
================================================================================

An explanation of the input/output parameters follows:
T/V    : Wall time / encoded variant.
N      : The order of the coefficient matrix A.
NB     : The partitioning blocking factor.
P      : The number of process rows.
Q      : The number of process columns.
Time   : Time in seconds to solve the linear system.
Gflops : Rate of execution for solving the linear system.

The following parameter values will be used:

N      :   43392
NB     :     192
PMAP   : Row-major process mapping
P      :       4
Q      :       5
PFACT  :   Right
NBMIN  :       4
NDIV   :       2
RFACT  :   Crout
BCAST  :  1ringM
DEPTH  :       1
SWAP   : Mix (threshold = 64)
L1     : transposed form
U      : transposed form
EQUIL  : yes
ALIGN  : 8 double precision words

--------------------------------------------------------------------------------

- The matrix A is randomly generated for each test.
- The following scaled residual check will be computed:
      ||Ax-b||_oo / ( eps * ( || x ||_oo * || A ||_oo + || b ||_oo ) * N )
- The relative machine precision (eps) is taken to be               1.110223e-16
- Computational tests pass if scaled residuals are less than                16.0


================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR11C2R4       43392   192     4     5             360.95             1.509e+02
HPL_pdgesv() start time Mon Apr 20 20:20:22 2015

HPL_pdgesv() end time   Mon Apr 20 20:26:23 2015

--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0007863 ..... PASSED
================================================================================

Finished      1 tests with the following results:
              1 tests completed and passed residual checks,
              0 tests completed and failed residual checks,
              0 tests skipped because of illegal input values.
--------------------------------------------------------------------------------

End of Tests.
================================================================================
```

Figura 4.7: Resultado de la prueba de Linpack en xexelo1.

Capítulo 5
Conclusiones

En el diseño e implementación del sistema de cómputo distribuido, apoyados en el estándar EIA/TIA568-C.1, se aprovecharon adecuadamente los espacios del laboratorio B404 para la ocupación de los elementos de soporte y para la realización del cableado estructurado. Además, gracias a la prueba RFC2544 se comprobó la calidad las conexiones realizadas.

Referente a los equipos de cómputo y comunicaciones, se emplearon equipos de marcas reconocidas en el mundo tecnológico, como son: Supermicro, Dell y Extreme Networks.

En cuanto a la configuración de los equipos de cómputo, el acoplamiento de CentOS y OpenMPI demostró ser una solución funcional y económica, dado que su utilización no generó algún costo monetario. Por otro lado, la relación costo-beneficio en la construcción del sistema distribuido resultó conveniente, debido a que muchos elementos de hardware fueron reutilizados.

Respecto a las pruebas de funcionamiento se confirmó el beneficio de emplear un clúster para resolver problemas que implican un gran número de operaciones aritméticas. Esto se afirma porque en la multiplicación de las matrices cuadradas de 9000 X 9000, la tarea se realizó aproximadamente nueve veces más rápido en xexelo1, utilizando sus veinte unidades de procesamiento, y aproximadamente catorce veces más rápido en xexelo0, ocupando noventa unidades de procesamiento, sin embargo, se deben realizar ajustes en OpenMPI para el manejo de la opción multi-hilo en los servidores Supermicro y especificar las unidades de procesamiento a tareas particulares, con el fin de obtener mejores tiempos de solución en los problemas paralelizados.

5.1. Trabajo futuro

El modelo que se siguió para la construcción de los clústeres, sólo contempló un servidor central que se encargó de generar y repartir las tareas a los nodos esclavos, además de dar acceso a internet y compartir el directorio de trabajo. Si este nodo maestro falla, todo el clúster quedaría inservible, por lo tanto es necesario añadir redundancia al sistema a través de un servidor de respaldo. Aunado a ésto, es sustancial incorporar herramientas que automaticen la instalación del software en los ordenadores que presenten problemas o en los nuevos ordenadores que favorezcan la escalación de los clústeres.

También se requiere instalar y configurar una interfaz de atención para que los usuarios puedan conectarse vía remota y se les asigne una cantidad adecuada de procesos para ejecutar sus tareas.

En cuanto a mediciones, es necesario realizar una prueba de rendimiento para xexelo0 y ejecutar programas que impliquen granularidad gruesa (como imágenes o archivos que contengan grandes cantidades de datos).

Respecto a la infraestructura del laboratorio de redes, se pueden considerar unidades de respaldo de energía eléctrica (UPS) y el uso de algún sistema de enfriamiento para el gabinete donde se ubican los servidores Supermicro, debido a que éstos generan mucha energía calorífica.

Apéndice A
Test RFC2544

En las imágenes se aprecia la información general, los resultados, las condiciones y las estadísticas de la prueba RFC2544 realizada a los enlaces permanentes del sistema de cómputo distribuido.

RFC 2544 Ethernet Test Report

Configuration Name	25
Customer	reporte25031511
Technician	fernando octavo
Location	lab404uacm2
Comments	port01
Date	Mar-25-15
Time Start	18:10:56
Time End	18:21:04
HST SW Revision	6.33.01

(a) Información general.

Throughput Test Results: PASS

Throughput Threshold: 100%

Frame Length (Bytes)	Measured Rate (%)	PASS/FAIL
64	99.988	PASS
128	99.988	PASS
256	99.988	PASS
512	99.988	PASS
1024	99.988	PASS
1280	99.989	PASS
1518	99.988	PASS
9600	99.994	PASS

(b) Resultados de la prueba de rendimiento.

Port 1 HST Setup

Termination	10/100/1G Electrical Eth.
Test Mode	Layer 2 Traffic
AutoNeg	Speed 1000,Duplex Full
Framing	DIX
Encapsulation	None
Destination Address Type	UNICAST
Source Address	00:80:16:45:2E:9A
Destination Address	00:80:16:8C:0C:73

Auto Negotiation Statistics

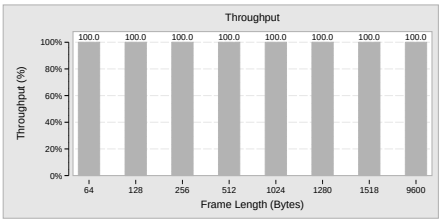
Speed (Mbps)	1000
Duplex	Full
10Base-TX FDX/HDX	Yes/Yes
100Base-TX FDX/HDX	Yes/Yes
1000Base-TX FDX/HDX	Yes/Yes

Test Configuration

Tests to Run	Throughput
Maximum Test Bandwidth	100%
Frame Lengths	64,128,256,512,1024,1280, 1518,9600
Bandwidth Measurement Accuracy	To within 1%
Throughput Frame Loss Tolerance	0%
Throughput Trial Duration	60 seconds
Throughput Pass Threshold	100%

(c) Condiciones de la prueba.

Throughput Test Results:



Frame Length (Bytes)	Cfg Rate (Mbps)	Measured Rate (Mbps)	Measured Rate (%)	Measured Rate (frms/sec)	Pause Detected
64	1000.000	999.880	99.988	1487916	No
128	1000.000	999.880	99.988	844493	No
256	1000.000	999.880	99.988	452844	No
512	1000.000	999.880	99.988	234935	No
1024	1000.000	999.880	99.988	119718	No
1280	1000.000	999.890	99.989	96143	No
1518	1000.000	999.880	99.988	81265	No
9600	1000.000	999.940	99.994	12993	No

(d) Estadísticas de la prueba.

Figura A.1: Prueba RFC2544 del nodo de red del puerto 01 del panel A del Rack 4 MDA.

Apéndice B

Establecimiento de los repositorios de CentOS

Para que CentOS obtenga las bibliotecas de software de sitios específicos y confiables, se realiza lo siguiente:

-Se verifican las llaves GPG (GNU Privacy Guard)
*rpm –import /etc/pki/rpm-gpg/RPM-GPG-KEY**

-Se habilitan los repositorios EPEL y RPMforge
rpm –import http://dag.wieers.com/rpm/packages/RPM-GPG-KEY.dag.txt

-Se posiciona, a través del prompt, al directorio tmp
cd/tmp

-Se obtienen los manejadores de paquetes de redhat
wget http://pkgs.repoforge.org/rpmforge-release/rpmforge-release-0.5.2-2.el6.rf.x86_ 64.rpm
rpm -ivh rpmforge-release-0.5.2-2.el6.rf.x86_ 64.rpm

-Para revisar los repositorios más recientes visitar *http://packages.sw.be/rpmforge-release/*

-Se agregan los repositorios de Fedora.
rpm –import https://fedoraproject.org/static/0608B895.txt
wget http://dl.fedoraproject.org/pub/epel/6/x86_ 64/epel-release-6-8.noarch.rpm
rpm -ivh epel-release-6-8.noarch.rpm

Apéndice C
Programa holamundo.c

El programa holamundo.c toma el un número total de procesos, los vincula con el número total de unidades de procesamiento de todos los nodos. Enseguida les pide su nombre de host a los nodos y hace que estos envíen su identificador del proceso asignado junto con su nombre de host.

```
#include <mpi.h>       /*Se establecen los archivos de cabecera MPI*/
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
   char idstr[32];
   char buff[128];
   char hostname[60];
   int numprocs;
   int myid;
   int i;
   MPI_Status stat;

   /*Inicia el ambiente de ejecución de MPI*/
   MPI_Init(&argc,&argv);

   /*Lee el número de procesos y se lo asigna a la variable
   de tipo entero numprocs*/
   MPI_Comm_size(MPI_COMM_WORLD,&numprocs);

   /*Especifica las etiquetas de los procesos invocados*/
   MPI_Comm_rank(MPI_COMM_WORLD,&myid);
   if(myid == 0)
   {
     gethostname(hostname, 50);
     printf("Tenemos %d procesadores, yo soy el proceso maestro:%d en maquina:%s\n", numprocs,myid,hostname);
     for(i=1;i<numprocs;i++)
     {
      sprintf(buff, "¡Holaaa %d! ", i);
      MPI_Send(buff, 128, MPI_CHAR, i, 0, MPI_COMM_WORLD);  /*Envía un mensaje*/
     }
     for(i=1;i<numprocs;i++)
     {
      MPI_Recv(buff, 128, MPI_CHAR, i, 0, MPI_COMM_WORLD, &stat); /*Recibe un mensaje*/
      printf("%s\n", buff);
     }

  }
   else
  {
```

```
     gethostname(hostname, 50);
     MPI_Recv(buff, 128, MPI_CHAR, 0, 0, MPI_COMM_WORLD, &stat);
     sprintf(idstr, "Proceso %d en maquina:%s ", myid,hostname);
     strcat(buff, idstr);
     strcat(buff, "reportandose");
     MPI_Send(buff, 128, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
}

/*Finaliza el ambiente de ejecución de MPI*/
MPI_Finalize();
return 0;
}
```

Apéndice D
Programa productomatrices.c

```
/*****************************************************************************************
* Programa de multiplicación de matrices usando MPI.
*
* Viraj Brian Wijesuriya - Universidad de la Escuela de Informática de Colombo, Sri Lanka.
*
* Trabaja con cualquier tipo de dos matrices [A], [B] las cuales son multiplicadas para producir otra matriz [c].
*
* El proceso maestro inicia los operandos de la multiplicación, distribuye las operaciones de multiplicación hacia los procesos obreros y reduce los resultados de los mismos para construir la salida final.
*
*****************************************************************************************/

    #include<stdio.h>
    #include<mpi.h>
    #define NUM_ROWS_A 9000 //rows of input [A]
    #define NUM_COLUMNS_A 9000 //columns of input [A]
    #define NUM_ROWS_B 9000 //rows of input [B]
    #define NUM_COLUMNS_B 9000 //columns of input [B]
    #define MASTER_TO_SLAVE_TAG 1 //tag for messages sent from master to slaves
    #define SLAVE_TO_MASTER_TAG 4 //tag for messages sent from slaves to master
    void makeAB(); //makes the [A] and [B] matrixes
    void printArray(); //print the content of output matrix [C];
    int rank; //process rank
    int size; //number of processes
    int i, j, k; //helper variables
    double mat_a[NUM_ROWS_A][NUM_COLUMNS_A]; //declare input [A]
    double mat_b[NUM_ROWS_B][NUM_COLUMNS_B]; //declare input [B]
    double mat_result[NUM_ROWS_A][NUM_COLUMNS_B]; //declare output [C]
    double start_time; //hold start time
    double end_time; // hold end time
    int low_bound; //low bound of the number of rows of [A] allocated to a slave
    int upper_bound; //upper bound of the number of rows of [A] allocated to a slave
    int portion; //portion of the number of rows of [A] allocated to a slave
    MPI_Status status; // store status of a MPI_Recv
    MPI_Request request; //capture request of a MPI_Isend
    int main(int argc, char *argv[])
    {
    MPI_Init(&argc, &argv); //initialize MPI operations
    MPI_Comm_rank(MPI_COMM_WORLD, &rank); //get the rank
    MPI_Comm_size(MPI_COMM_WORLD, &size); //get number of processes
    /* master initializes work*/
    if (rank == 0) {
    makeAB();
    start_time = MPI_Wtime();
    for (i = 1; i < size; i++) {//for each slave other than the master
    portion = (NUM_ROWS_A / (size - 1)); // calculate portion without master
    low_bound = (i - 1) * portion;
    if (((i + 1) == size) && ((NUM_ROWS_A % (size - 1)) != 0)) {//if rows of [A] cannot be equally divided among slaves
    upper_bound = NUM_ROWS_A; //last slave gets all the remaining rows
    } else {
    upper_bound = low_bound + portion; //rows of [A] are equally divisable among slaves
    }
    //send the low bound first without blocking, to the intended slave
    MPI_Isend(&low_bound, 1, MPI_INT, i, MASTER_TO_SLAVE_TAG, MPI_COMM_WORLD, &request);
    //next send the upper bound without blocking, to the intended slave
    MPI_Isend(&upper_bound, 1, MPI_INT, i, MASTER_TO_SLAVE_TAG + 1, MPI_COMM_WORLD, &request);
```

```
//finally send the allocated row portion of [A] without blocking, to the intended slave
MPI_Isend(&mat_a[low_bound][0], (upper_bound - low_bound) * NUM_COLUMNS_A, MPI_DOUBLE, i, MASTER_TO_SLAVE_TAG + 2, MPI_COMM_WORLD, &request);
}
}
//broadcast [B] to all the slaves
MPI_Bcast(&mat_b, NUM_ROWS_B*NUM_COLUMNS_B, MPI_DOUBLE, 0, MPI_COMM_WORLD);
/* work done by slaves*/
if (rank > 0) {
//receive low bound from the master
MPI_Recv(&low_bound, 1, MPI_INT, 0, MASTER_TO_SLAVE_TAG, MPI_COMM_WORLD, &status);
//next receive upper bound from the master
MPI_Recv(&upper_bound, 1, MPI_INT, 0, MASTER_TO_SLAVE_TAG + 1, MPI_COMM_WORLD, &status);
//finally receive row portion of [A] to be processed from the master
MPI_Recv(&mat_a[low_bound][0], (upper_bound - low_bound) * NUM_COLUMNS_A, MPI_DOUBLE, 0, MASTER_TO_SLAVE_TAG + 2, MPI_COMM_WORLD, &status);
for (i = low_bound; i < upper_bound; i++) {//iterate through a given set of rows of [A]
for (j = 0; j < NUM_COLUMNS_B; j++) {//iterate through columns of [B]
for (k = 0; k < NUM_ROWS_B; k++) {//iterate through rows of [B]
mat_result[i][j] += (mat_a[i][k] * mat_b[k][j]);
}
}
}
//send back the low bound first without blocking, to the master
MPI_Isend(&low_bound, 1, MPI_INT, 0, SLAVE_TO_MASTER_TAG, MPI_COMM_WORLD, &request);
//send the upper bound next without blocking, to the master
MPI_Isend(&upper_bound, 1, MPI_INT, 0, SLAVE_TO_MASTER_TAG + 1, MPI_COMM_WORLD, &request);
//finally send the processed portion of data without blocking, to the master
MPI_Isend(&mat_result[low_bound][0], (upper_bound - low_bound) * NUM_COLUMNS_B, MPI_DOUBLE, 0, SLAVE_TO_MASTER_TAG + 2, MPI_COMM_WORLD, &request);
}
/* master gathers processed work*/
if (rank == 0) {
for (i = 1; i < size; i++) {// untill all slaves have handed back the processed data
//receive low bound from a slave
MPI_Recv(&low_bound, 1, MPI_INT, i, SLAVE_TO_MASTER_TAG, MPI_COMM_WORLD, &status);
//receive upper bound from a slave
MPI_Recv(&upper_bound, 1, MPI_INT, i, SLAVE_TO_MASTER_TAG + 1, MPI_COMM_WORLD, &status);
//receive processed data from a slave
MPI_Recv(&mat_result[low_bound][0], (upper_bound - low_bound) * NUM_COLUMNS_B, MPI_DOUBLE, i, SLAVE_TO_MASTER_TAG + 2, MPI_COMM_WORLD, &status);
}
end_time = MPI_Wtime();
printf("\nRunning Time = %f\n\n", end_time - start_time);
printArray();
}
MPI_Finalize(); //finalize MPI operations
return 0;
}
void makeAB()
{
for (i = 0; i < NUM_ROWS_A; i++) {
for (j = 0; j < NUM_COLUMNS_A; j++) {
mat_a[i][j] = i + j;
}
}
for (i = 0; i < NUM_ROWS_B; i++) {
for (j = 0; j < NUM_COLUMNS_B; j++) {
mat_b[i][j] = i*j;
}
}
}
void printArray()
{
for (i = 0; i < NUM_ROWS_A; i++) {
printf("\n");
for (j = 0; j < NUM_COLUMNS_A; j++)
printf("%8.2f ", mat_a[i][j]);
}
printf("\n\n\n");
for (i = 0; i < NUM_ROWS_B; i++) {
printf("\n");
for (j = 0; j < NUM_COLUMNS_B; j++)
printf("%8.2f ", mat_b[i][j]);
}
printf("\n\n\n");
for (i = 0; i < NUM_ROWS_A; i++) {
printf("\n");
for (j = 0; j < NUM_COLUMNS_B; j++)
printf("%8.2f ", mat_result[i][j]);
}
printf("\n\n");
}
```

Apéndice E
Programa serial.c

Adaptación del programa productomatrices.c de Viraj Brian Wijesuriya - Universidad de la Escuela de Informática de Colombo, Sri Lanka (Apéndice D).

Se suprimieron las bibliotecas de openmpi y todos los métodos que implicaban paralelismo.

Un único procesador genera las matrices A y B, y todas las operaciones implícitas.

```
#include<stdio.h>

#define NUM_ROWS_A 9000
#define NUM_COLUMNS_A 9000
#define NUM_ROWS_B 9000
#define NUM_COLUMNS_B 9000

void makeAB();
void printArray();
int i,j,k;

double mat_a[NUM_ROWS_A][NUM_COLUMNS_A];
double mat_b[NUM_ROWS_B][NUM_COLUMNS_B];
double mat_result[NUM_ROWS_A][NUM_COLUMNS_B];

int main(){
makeAB();
for (i=0;i<NUM_ROWS_A;i++){
for (j=0;j<NUM_COLUMNS_B;j++){
mat_result[i][j]=0;
for(k=0;k<NUM_COLUMNS_B;k++){
mat_result[i][j]=mat_result[i][j]+(mat_a[i][k]*mat_b[k][j]);
}
}
}
printArray();
return 0;
}

void makeAB()
{
for (i=0;i<NUM_ROWS_A;i++){
for (j=0;j<NUM_COLUMNS_A;j++){
mat_a[i][j]=i+j;
}
}
for (i=0;i<NUM_ROWS_B;i++){
for (j=0;j<NUM_COLUMNS_B;j++){
```

```
mat_b[i][j]=i*j;
}
}
}
void printArray()
{
for (i=0;i<NUM_ROWS_A;i++){
printf("\n");
for (j=0;j<NUM_COLUMNS_A;j++)
printf("%8.2f",mat_a[i][j]);
}
printf ("\n\n\n");
for(i=0;i<NUM_ROWS_B;i++){
printf("\n");
for (j=0;j<NUM_COLUMNS_B;j++)
printf("%8.2f",mat_b[i][j]);
}
printf("\n\n\n");
for(i=0;i<NUM_ROWS_A;i++){
printf("\n");
for(j=0;j<NUM_COLUMNS_B;j++)
printf("%8.2f",mat_result[i][j]);
}
printf("\n\n");
}
```

Apéndice F
Benchmark Linpack

Para ejecutar la prueba de Linpack, el nodo coordinador del clúster xexelo1 debe tener instalado tres paquetes de software, HPL, GotoBLAS y OpenMPI (la instalación de este último se explicó en el capítulo tres del presente documento). A continuación se describe la instalación de GotoBLAS2.

En una sesión de superusuario, se descarga el paquete ***GotoBLAS2-1.13.tar*** de la página ***https://www.tacc.utexas.edu/research-development/tacc-software/gotoblas2*** .

Enseguida se descomprime y se instala en el directorio ***/home/tesista/openmpi/*** .

```
# tar xzf GotoBLAS2-1.13.tar.gz
# cd GotoBLAS2
# make
# GotoBLAS2 didn't detect Core i7 automatically.
# make TARGET=NEHALEM %Para detectar automáticamente los núcleos i7.
```

Después, del sitio ***http://www.netlib.org/benchmark/hpl/hpl-2.1.tar.gz*** se descarga el paquete ***hpl-2.1.tar.gz*** .

Se descomprime en el directorio GotoBLAS2.

```
# tar -xvzf hpl-2.1.tar.gz
```

Se accede al directorio ***setup*** y se copia en el directorio principal ***hpl*** el archivo ***Make.Linux_PII_CBLAS*** con el nombre ***Make.Linux_gotoblas2*** . Éste archivo se debe modificar como sigue:

```
#
#  -- High Performance Computing Linpack Benchmark (HPL)
#     HPL - 2.1 - October 26, 2012
#     Antoine P. Petitet
#     University of Tennessee, Knoxville
#     Innovative Computing Laboratory
#     (C) Copyright 2000-2008 All Rights Reserved
#
#  -- Copyright notice and Licensing terms:
#
#  Redistribution  and  use in  source and binary forms, with or without
#  modification, are  permitted provided  that the following  conditions
```

```
#  are met:
#
#  1. Redistributions  of  source  code  must retain the above copyright
#  notice, this list of conditions and the following disclaimer.
#
#  2. Redistributions in binary form must reproduce  the above copyright
#  notice, this list of conditions,  and the following disclaimer in the
#  documentation and/or other materials provided with the distribution.
#
#  3. All  advertising  materials  mentioning  features  or  use of this
#  software must display the following acknowledgement:
#  This  product  includes  software  developed  at  the  University  of
#  Tennessee, Knoxville, Innovative Computing Laboratory.
#
#  4. The name of the  University,  the name of the  Laboratory,  or the
#  names  of  its  contributors  may  not  be used to endorse or promote
#  products  derived   from   this  software  without  specific  written
#  permission.
#
#  -- Disclaimer:
#
#  THIS  SOFTWARE  IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
#  ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES,  INCLUDING,  BUT NOT
#  LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
#  A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE UNIVERSITY
#  OR  CONTRIBUTORS  BE  LIABLE FOR ANY  DIRECT,  INDIRECT,  INCIDENTAL,
#  SPECIAL,  EXEMPLARY,  OR  CONSEQUENTIAL DAMAGES  (INCLUDING,  BUT NOT
#  LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
#  DATA OR PROFITS; OR BUSINESS INTERRUPTION)  HOWEVER CAUSED AND ON ANY
#  THEORY OF LIABILITY, WHETHER IN CONTRACT,  STRICT LIABILITY,  OR TORT
#  (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
#  OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
# ######################################################################
#
# ----------------------------------------------------------------------
# - shell --------------------------------------------------------------
# ----------------------------------------------------------------------
#
SHELL         = /bin/sh
#
CD            = cd
CP            = cp
LN_S          = ln -sf
MKDIR       = mkdir -p
RM            = /bin/rm -f
TOUCH       = touch
#
# ----------------------------------------------------------------------
# - Platform identifier ------------------------------------------------
# ----------------------------------------------------------------------
#
ARCH         = Linux_gotoblas2
#
# ----------------------------------------------------------------------
# - HPL Directory Structure / HPL library ------------------------------
# ----------------------------------------------------------------------
#
#TOPdir       = $(HOME)/hpl
TOPdir        = /home/tesista/openmpi/GotoBLAS2/hpl
INCdir        = $(TOPdir)/include
BINdir        = $(TOPdir)/bin/$(ARCH)
LIBdir        = $(TOPdir)/lib/$(ARCH)
#
HPLlib        = $(LIBdir)/libhpl.a
```

```
#
# ----------------------------------------------------------------------
# - Message Passing library (MPI) --------------------------------------
# ----------------------------------------------------------------------
# MPinc tells the  C  compiler where to find the Message Passing library
# header files,  MPlib  is defined  to be the name of  the library to be
# used. The variable MPdir is only used for defining MPinc and MPlib.
#
#MPdir         = /usr/local/mpi
#MPinc        = -I$(MPdir)/include
#MPlib         = $(MPdir)/lib/libmpich.a
MPdir         =
MPinc         =
MPlib         =
#
# ----------------------------------------------------------------------
# - Linear Algebra library (BLAS or VSIPL) -----------------------------
# ----------------------------------------------------------------------
# LAinc tells the  C  compiler where to find the Linear Algebra  library
# header files,  LAlib  is defined  to be the name of  the library to be
# used. The variable LAdir is only used for defining LAinc and LAlib.
#
#LAdir         = $(HOME)/netlib/ARCHIVES/Linux_PII
LAdir         = /home/tesista/openmpi/GotoBLAS2
LAinc         =
#LAlib         = $(LAdir)/libcblas.a $(LAdir)/libatlas.a
LAlib         = -L$(LAdir) -lgoto2
#
# ----------------------------------------------------------------------
# - F77 / C interface --------------------------------------------------
# ----------------------------------------------------------------------
# You can skip this section  if and only if  you are not planning to use
# a  BLAS  library featuring a Fortran 77 interface.  Otherwise,  it  is
# necessary  to  fill out the  F2CDEFS  variable  with  the  appropriate
# options.  **One and only one**  option should be chosen in **each** of
# the 3 following categories:
#
# 1) name space (How C calls a Fortran 77 routine)
#
# -DAdd_               : all lower case and a suffixed underscore  (Suns,
#                        Intel, ...),                           [default]
# -DNoChange         : all lower case (IBM RS6000),
# -DUpCase             : all upper case (Cray),
# -DAdd__              : the FORTRAN compiler in use is f2c.
#
# 2) C and Fortran 77 integer mapping
#
# -DF77_INTEGER=int   : Fortran 77 INTEGER is a C int,         [default]
# -DF77_INTEGER=long  : Fortran 77 INTEGER is a C long,
# -DF77_INTEGER=short : Fortran 77 INTEGER is a C short.
#
# 3) Fortran 77 string handling
#
# -DStringSunStyle    : The string address is passed at the string loca-
#                       tion on the stack, and the string length is then
#                       passed as  an  F77_INTEGER  after  all  explicit
#                       stack arguments,                       [default]
# -DStringStructPtr   : The address  of  a  structure  is  passed  by  a
#                       Fortran 77  string,  and the structure is of the
#                       form: struct {char *cp; F77_INTEGER len;},
# -DStringStructVal   : A structure is passed by value for each  Fortran
#                       77 string,  and  the  structure is  of the form:
#                       struct {char *cp; F77_INTEGER len;},
# -DStringCrayStyle   : Special option for  Cray  machines,  which  uses
```

```
#                       Cray  fcd  (fortran  character  descriptor)  for
#                       interoperation.
#
F2CDEFS      =
#
# ----------------------------------------------------------------------
# - HPL includes / libraries / specifics -------------------------------
# ----------------------------------------------------------------------
#
HPL_INCLUDES = -I$(INCdir) -I$(INCdir)/$(ARCH) $(LAinc) $(MPinc)
HPL_LIBS     = $(HPLlib) $(LAlib) $(MPlib)
#
# - Compile time options -----------------------------------------------
#
# -DHPL_COPY_L           force the copy of the panel L before bcast;
# -DHPL_CALL_CBLAS       call the cblas interface;
# -DHPL_CALL_VSIPL       call the vsip  library;
# -DHPL_DETAILED_TIMING  enable detailed timers;
#
# By default HPL will:
#    *) not copy L before broadcast,
#    *) call the BLAS Fortran 77 interface,
#    *) not display detailed timing information.
#
HPL_OPTS     = -DHPL_CALL_CBLAS
#
# ----------------------------------------------------------------------
#
HPL_DEFS     = $(F2CDEFS) $(HPL_OPTS) $(HPL_INCLUDES)
#
# ----------------------------------------------------------------------
# - Compilers / linkers - Optimization flags ---------------------------
# ----------------------------------------------------------------------
#
#CC            = /usr/bin/gcc
CC           = mpicc
CCNOOPT  = $(HPL_DEFS)
CCFLAGS   = $(HPL_DEFS) -fomit-frame-pointer -O3 -funroll-loops
#
# On some platforms,  it is necessary  to use the Fortran linker to find
# the Fortran internals used in the BLAS library.
#
LINKER       = mpif77
LINKFLAGS    = $(CCFLAGS)
#
ARCHIVER     = ar
ARFLAGS      = r
RANLIB       = echo
#
# ----------------------------------------------------------------------
```

Posterior a la modificación del archivo ***Make.Linux_gotoblas2***, éste se compila con el comando:
make arch=Linux_gotoblas2

Enseguida se crea un archivo con el nombre ***gotoblas2.sh*** en el directorio */etc/profile.d* (en todas las computadoras del clúster). El archivo debe incluir lo siguiente:

```
#!/bin/bash
export LD_LIBRARY_PATH=/home/tesista/GotoBLAS2:$LD_LIBRARY_PATH
```

Usando la herramienta del sitio web ***www.advancedclustering.com/faq/how-do-i-tune-my-hpldat-file.html*** (figura F.1) se adapta el archivo ***HPL.dat*** que se encuentra en ***/GotoBLAS2/hpl/bin/Linux_gotoblas2/*** .

Lo que se debe especificar es:

- El número de nodos.
- Núcleos por nodo.
- Memoria RAM por nodo en MB.
- Tamaño de Bloque (por default 192).

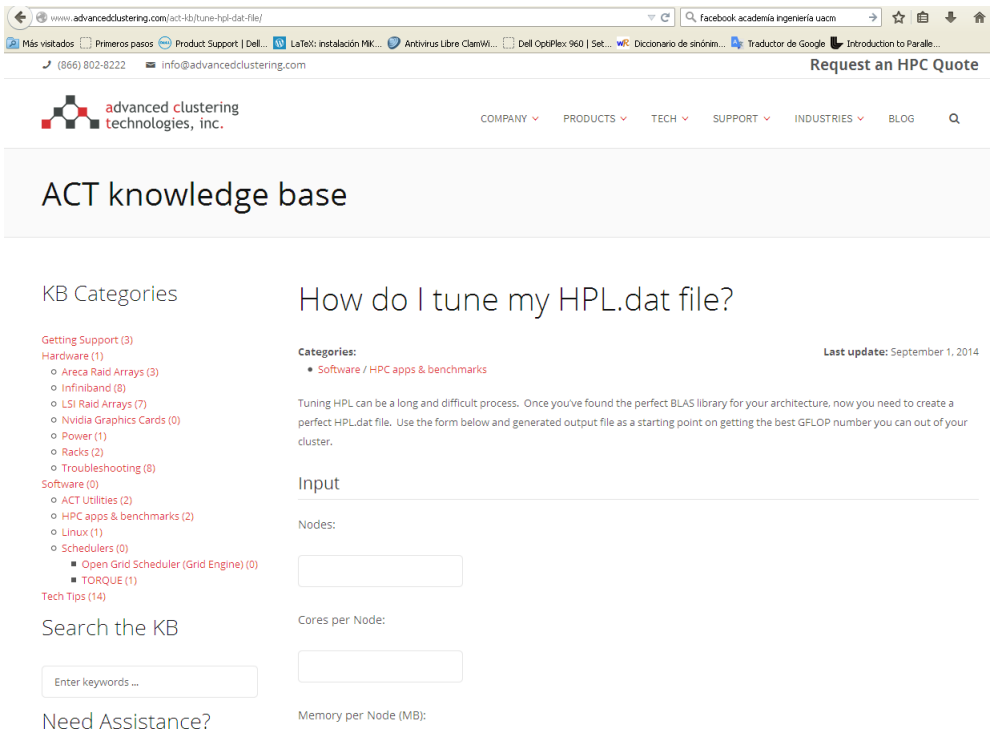


Figura F.1: Configuración del archivo HPL.dat.

El archivo HPL.dat para xexelo1 quedó de la siguiente manera:

```
HPLinpack benchmark input file
Innovative Computing Laboratory, University of Tennessee
HPL.out      output file name (if any)
6            device out (6=stdout,7=stderr,file)
1            # of problems sizes (N)
43392         Ns
1            # of NBs
192           NBs
0            PMAP process mapping (0=Row-,1=Column-major)
1            # of process grids (P x Q)
4            Ps
5            Qs
16.0         threshold
1            # of panel fact
2            PFACTs (0=left, 1=Crout, 2=Right)
1            # of recursive stopping criterium
4            NBMINs (>= 1)
1            # of panels in recursion
2            NDIVs
1            # of recursive panel fact.
1            RFACTs (0=left, 1=Crout, 2=Right)
1            # of broadcast
1            BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
1            # of lookahead depth
1            DEPTHs (>=0)
2            SWAP (0=bin-exch,1=long,2=mix)
64           swapping threshold
0            L1 in (0=transposed,1=no-transposed) form
0            U  in (0=transposed,1=no-transposed) form
1            Equilibration (0=no,1=yes)
8            memory alignment in double (> 0)
##### This line (no. 32) is ignored (it serves as a separator). ######
0                               Number of additional problem sizes for PTRANS
1200 10000 30000                values of N
0                               number of additional blocking sizes for PTRANS
40 9 8 13 13 20 16 32 64        values of NB
```

Después, se crea un archivo de texto plano (***hosts.txt***) que incluye las direcciones IP de todos los nodos del clúster.

6.6.6.1
6.6.6.2
6.6.6.3
6.6.6.4
6.6.6.5
6.6.6.6
6.6.6.7
6.6.6.8
6.6.6.9
6.6.6.10

Se copia todo el directorio GotoBLAS2 a todos los nodos del clúster.

scp -r /home/tesista/GotoBLAS2/ 6.6.6.1:/home/tesista/
scp -r /home/tesista/GotoBLAS2/ 6.6.6.2:/home/tesista/
.
.
.
scp -r /home/tesista/GotoBLAS2/ 6.6.6.10:/home/tesista/

Finalmente se ejecuta el siguiente comando:

mpiexec –hostfile hosts -np 20 ./xhpl

y se obtiene el resultado de la figura 4.7 del capítulo cuatro.

Referencias

[1] Introduction to Parallel Computing, [en línea]; 7 de Febrero de 2015, [consulta: Marzo de 2015], disponible: *https://computing.llnl.gov/tutorials/parallel*

[2] History of Computing [en línea]; 4 de junio de 2012, [consulta: Junio 2014], disponible: *http://www.livescience.com/20718-computer-history.html*

[3] Wilkinson and Allen, *Parallel Programming Techniques and Applications using Networked Workstations and Parallel computers*, Prentice Hall, March 2004, pp. 4-150.

[4] J. Ansel, C. Chan, Y. L. Wong, M. Olszewski, Q. Zhao, A. Edelman, and S. Amarasinghe. *PetaBricks: A language and compiler for algorithmic choice. In Conf. on Programming Language Design and Implementation*, Jun 2009.

[5] Tannenbaum, *Distributed Systems Principles and Paradigms*, Pearson Prentice Hall, 2007, pp 2-170.

[6] Coulouris, Dollimore, Kindberg, *Distributed Systems.Concepts and Design*, Addison Wesley, 2001, pp 2-227.

[7] Shin Yeo, Buyya, Pourreza, Eskicioglu, Graham, Sommers, *Handbook of Nature-Inspired and Innovative Computing*, Zomaya, Albert Y., 2006, pp 521-544.

[8] Sterling, *Beowulf Cluster Computing with Linux*, The MIT Press, 2002, pp 1-194.

[9] Historia de la Tecnología: Clúster Beowulf, la supercomputadora de los pobres [en línea]; 9 de Noviembre de 2011, [consulta: Febrero 2014], disponible: *http://hipertextual.com/2011/11/historia-de-la-tecnologia-cluster-beowulf-la-supercomputadora-de-los-pobres*

[10] Ferreira, Kettmann, Thomasch, Silcocks, Chen, Daunois, Ihamo, Harada, Hill, Bernocchi and Ford, *Linux HPC Cluster Installation*, IBM Corporation, International Technical Support Organization, June 2001, p. 11-21.

[11] List Statistics | TOP500 Supercomputer Sites [en línea]; Julio 2015, [consulta: Julio 2015], disponible: *http://top500.org/statistics/list/*

[12] Tanenbaum, *Redes de Computadoras*, Pearson Prentice Hall, 2003, pp 2-130.

[13] Introduction to Parallel Computing, [en línea]; 7 de Febrero de 2015, [consulta: Abril de 2015], disponible:
https://computing.llnl.gov/tutorials/parallel_comp/parallelClassifications.pdf

[14] Intel. Imagen Digital. Xeon 5600 Processor Chip and Die. 9 de Noviembre de 2011, [consulta: Febrero de 2015], disponible:
https://computing.llnl.gov/tutorials/linux_clusters/images/xeon5600processorDie2.jpg

[15] Introduction to Parallel Computing, [en línea]; 7 de Febrero de 2015, [consulta: Marzo de 2015], disponible:
https://computing.llnl.gov/tutorials/parallel_comp/images/hybrid_mem2.gif

[16] Open MPI: Open Source High Performance Computing [en línea]; julio de 2015, [consulta: Julio 2015], disponible: *http://www.open-mpi.org/*

[17] A2.Parallel Programming in C [en línea]; 2012, [consulta: Agosto 2015], disponible:
http://gribblelab.org/CBootcamp/A2_Parallel_Programming_in_C.html#sec-2-1-2

[18] Introduction to Parallel Computing, [en línea]; 7 de Febrero de 2015, [consulta: Mayo de 2015], disponible:
https://computing.llnl.gov/tutorials/parallel_comp/images/amdahl1.gif

[19] Introduction to Parallel Computing, [en línea]; 7 de Febrero de 2015, [consulta: Junio de 2015], disponible:
https://computing.llnl.gov/tutorials/parallel_comp/images/amdahl2.gif

[20] A Guide to Getting Started With pfSense [en línea]; 2014-2015, [consulta: Junio 2014], disponible: *https://www.pfsense.org/getting-started/*

[21] help, relocation truncated to fit [en línea]; 27 de junio de 2014, [consulta: Julio 2015], disponible: *https://software.intel.com/en-us/forums/topic/270714*

[22] Introduction to Parallel Computing, [en línea]; 7 de Febrero de 2015, [consulta: Abril de 2015], disponible:
https://computing.llnl.gov/tutorials/parallel_comp/images/vonNeumann1.gif

[23] Introduction to Parallel Computing, [en línea]; 7 de Febrero de 2015, [consulta: Mayo de 2015], disponible:
https://computing.llnl.gov/tutorials/parallel_comp/#Terminology