ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE INGENIERÍA

CLUSTERS DE COMPUTADORES PERSONALES CON LINUX

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

DIEGO ALEJANDRO FERNÁNDEZ AYALA

RAÚL DAVID MEJÍA NAVARRETE

DIRECTOR: IVAN BERNAL CARRILLO, Ph. D.

Quito, Octubre 2005

DECLARACIÓN

Nosotros, Diego Alejandro Fernández Ayala y Raúl David Mejía Navarrete, declaramos que el trabajo aquí descrito es de nuestra autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

La Escuela Politécnica Nacional, puede hacer uso de los derechos correspondientes a este trabajo, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

Diego Alejandro Fernández Ayala

Raúl David Mejía Navarrete

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Diego Alejandro Fernández Ayala y Raúl David Mejía Navarrete, bajo mi supervisión.

Iván Bernal Carrillo, Ph. D.

DIRECTOR DEL PROYECTO

AGRADECIMIENTOS

Agradezco afectuosamente a mi familia, a mis amigos, y a uno de ellos en especial: a David M. por su apoyo y dedicación durante varios meses de arduo trabajo.

Deseo agradecer al Dr. Iván Bernal por su apoyo incondicional durante el presente Proyecto de Titulación.

DIEGO FERNÁNDEZ A.

AGRADECIMIENTOS

Quiero empezar agradeciendo a mi familia, por su amor incondicional, por su cariño y confianza que me brindan en todo momento y en especial a mis padres por ser un ejemplo en mi vida.

Al Dr. Iván Bernal, director del presente Proyecto de Titulación, mi más sincera gratitud por su confianza, paciencia y disposición en todo momento, por su apoyo y conocimiento durante mis estudios en pregrado, por haberme enseñado el camino de la programación y por su orientación y guía en la realización de este proyecto, pero sobre todo gracias por su amistad.

También quisiera agradecer a Alexander Verdesoto, un buen amigo, que creyó en mí incluso cuando yo dejé de creer en mi mismo, y quién ha sido un apoyo invaluable en el inicio de mi carrera profesional y por los gratos momentos que hemos compartido juntos.

Al clan "SMT" con quienes comparto largas horas de entretenimiento en el mundo de los *LAN Games*, y de quienes tengo grabado en mi memoria los mejores momentos de mi paso por la Escuela Politécnica Nacional.

Quiero agradecer a todas las personas que me han apoyado en el desarrollo de este proyecto.

A los que agradecí antes, *a* los que se me olvida agradecer, y a los que deberé agradecer en el futuro: gracias.

DAVID MEJIA A.

CONTENIDO

ÍNDICE DE FIGURAS

CAPÍTULO 1

CAPÍTULO 2

CAPÍTULO 3

CAPÍTULO 4

CAPÍTULO 5

ANEXO A

ANEXO B

ANEXO E

ÍNDICE DE TABLAS

CAPÍTULO 1

CAPÍTULO 2

CAPÍTULO 3

CAPÍTULO 4

CAPÍTULO 5

ANEXO A

ANEXO B

RESUMEN

Las características y los costos de los computadores genéricos (*off-the-shelf*) de hoy en día, así como la disponibilidad de interconexión a altas velocidades (100 Mbps, 1 Gbps o 10 Gbps) ofrecidas por Ethernet, y el uso de software de distribución libre permiten la construcción de ambientes de computación paralela de alto rendimiento. Los supercomputadores disponibles en el mercado, no pueden competir con el precio del sistema que se puede obtener.

Desde una perspectiva de componentes de hardware y capacidad de procesamiento, los *clusters* son sistemas de cómputo fenomenales en precio y rendimiento. Obviamente, entre las desventajas de este sistema se pueden mencionar que no existen centros de soporte a los cuales poder pedir ayuda si existe algún tipo de problema. Para poder solucionar los problemas que se presenten existe una gran cantidad de información disponible en sitios Web y grupos de noticias.

Los *clusters* están conformados por computadores genéricos denominados nodos. Dichos nodos pueden estar dedicados exclusivamente a realizar tareas para el *cluster*, por lo que no requieren de monitor, teclado o mouse; o pueden estar dedicados a diferentes actividades y se utilizarán los ciclos libres del procesador para realizar las tareas que requiera el *cluster*.

Existen paquetes de software que automatizan el proceso de instalación, de configuración y de administración de un *cluster*, denominados *toolkits*. Este conjunto de paquetes permite configurar un *cluster* completo en una fracción del tiempo que tomaría el hacerlo de forma manual. Estos *toolkits*, para instalación automática de *clusters*, pueden incluir una distribución de Linux; mientras que otros se instalan sobre una instalación existente de Linux. Sin embargo, incluso si primero se debe instalar Linux, los *toolkits* realizan la configuración e instalación de los paquetes requeridos por el *cluster* de forma automática.

Del conjunto de *toolkits* existentes se pueden mencionar a NPACI *Rocks* y a OSCAR.

NPACI (*National Partnership for Advanced Computational Infrastructure*) *Rocks* es una colección de software de código abierto para crear un *cluster* sobre Red Hat Linux. *Rocks* instala tanto Linux como software para *clusters.* La instalación toma unos pocos minutos.

OSCAR es una colección de software de código abierto que se instala sobre una instalación existente de Linux (Red Hat, Mandrake, Mandriva, Fedora).

Tanto *Rocks* como OSCAR incluyen una diversidad de software para construir *clusters.* El software núcleo es el mismo en OSCAR y en *Rocks.* Sin embargo, algunos paquetes no están disponibles en ambos *toolkits.* OSCAR dispone de una gran cantidad de documentación, mientras que *Rocks* dispone de un poco de información disponible en su sitio Web.

El aparecimiento de la computación paralela permitió que emerjan métodos de programación que hagan posible la implementación de algoritmos utilizando recursos compartidos: procesador, memoria, datos y servicios. Los programas desarrollados para *clusters* usualmente están escritos en C o en Fortran, y utilizan librerías de paso de mensajes para realizar operaciones paralelas; también pueden hacer uso de librerías matemáticas para resolución de problemas que involucren matrices, derivación e integración compleja.

Las librerías para paso de mensajes permiten escribir programas paralelos eficientes, proveen rutinas para inicializar y configurar el ambiente de mensajes, así como para enviar y recibir paquetes de datos. Los sistemas más populares de paso de mensajes son: PVM (*Parallel Virtual Machine*) del Laboratorio Nacional Oak Ridge y MPI (*Message Passing Interface*) definido por el Foro MPI.

PVM es una librería de paso de mensajes. Puede usarse para desarrollar y ejecutar aplicaciones paralelas en sistemas que están dentro del rango que va desde supercomputadores hasta *clusters* de estaciones de trabajo.

MPI es una especificación de paso de mensajes, diseñada para ser el estándar de computación paralela de memoria distribuida usando paso de mensajes. Esta interfaz intenta establecer un estándar práctico, eficiente, portátil y flexible para el paso de mensajes.

Los recursos del *cluster* deben ser administrados adecuadamente para que el administrador invierta la menor cantidad de tiempo en detectar, investigar y recuperar fallos de hardware y software, y de este modo definir posibles medidas de contingencia y tratar que el sistema esté libre de errores. Existen algunas herramientas para administración de *clusters*, como C3 (*Cluster, Command & Control*) y Ganglia que se incluyen en OSCAR y *Rocks* NPACI.

La administración de un *cluster* implica disponer de herramientas para planificar las tareas a realizarse. Es posible que en un *cluster* se ejecuten cientos o miles de tareas de muchos usuarios; algunas tareas pueden ejecutarse sobre ciertos nodos debido a que no todos los nodos poseen las mismas capacidades computacionales; por otro lado, algunos usuarios requieren prioridad de acceso a parte o a todos los recursos del *cluster*; además, algunas tareas pueden ejecutarse en cierta hora del día o solamente después de que otras tareas hayan finalizado. Estos son algunos de los motivos que justifican la necesidad de disponer de herramientas para planificación de tareas. Existen algunos sistemas de planificación de tareas, como: Condor, PBS *(Portable Batch System)* y Maui.

PRESENTACIÓN

La idea principal que impulsa el crecimiento de las tecnologías de la computación es la necesidad de más capacidad de procesamiento. Las aplicaciones necesitan de mayores recursos computacionales, a medida que la complejidad de los problemas que intentan resolver, aumenta.

Hoy en día, es posible disponer de gran capacidad computacional, que incluso puede superar a la encontrada en supercomputadores, mediante *clusters*. Los *clusters* están conformados por computadores de bajo costo, interconectados mediante tecnologías de red de alta velocidad y haciendo uso de software de libre distribución.

Este proyecto denominado: "*Clusters* de computadores personales con Linux" presenta las ideas básicas involucradas en la instalación y operación de *clusters*. Entre los objetivos específicos planteados se pueden citar:

- Implementar un *cluster* con PCs (*Personal Computers*) y hardware de redes de bajo costo.

- Identificar, familiarizarse y configurar adecuadamente las herramientas de instalación del software que permite la operación del *cluster* y su administración.

- Estudiar la herramienta de programación paralela MPI (*Message Passing Interface*).

- Utilizar el *cluster* para el desarrollo de una aplicación paralela utilizando MPI, depurarla y ejecutarla.

La presente obra está conformada por cinco capítulos, como se explica a continuación:

En el Capítulo 1, se exponen algunas ideas sobre la computación paralela. Se trata de proveer una visión global acerca de la tecnología *cluster*, sus características y sus componentes de hardware y software.

En el Capítulo 2, se presentan dos herramientas que permiten instalar, configurar y administrar *clusters*: OSCAR y NPACI *Rocks*; se describen los componentes que conforman estas herramientas, se mencionan algunos detalles sobre el procedimiento que utilizan para instalar un *cluster* completo y se especifica como instalarlas.

En el Capítulo 3, se presentan algunas ideas involucradas con el desarrollo de aplicaciones paralelas. Se describen algunas de las funcionalidades más importantes de la librería de paso de mensajes MPI. Y se presenta un ejemplo realizado utilizando MPI.

En el Capítulo 4, se presentan ideas relacionadas con la administración de recursos en *clusters*. Se discuten algunas herramientas que permiten administrar, monitorear y planificar tareas. Además, se presentan algunos sistemas de archivos paralelos.

En el Capítulo 5, se presentan algunas aplicaciones que han sido implementadas para *clusters*. Además, integra todos los conocimientos adquiridos acerca de la tecnología *cluster*, pues aquí se construye un *cluster* y se implementa una aplicación, que permite resolver Cadenas de Markov a Tiempo Discreto y a Tiempo Continuo, que pretende mostrar las bondades del sistema obtenido. También se presenta los criterios utilizados para la construcción e instalación del *cluster*, así como los criterios que se utilizaron para el desarrollo de la aplicación.

En el Capítulo 6, se plantea conclusiones y recomendaciones en base a la experiencia obtenida en la realización de este trabajo.

Además, los Anexos A y B proporcionan información sobre la instalación de OSCAR y de NPACI *Rocks*, respectivamente.

El Anexo C incluye una descripción de la instalación de paquetes mediante fuentes binarias.

El Anexo D incluye la declaración de funciones más utilizadas y algunas de las convenciones del estándar MPI para el lenguaje C.

El Anexo E incluye la descripción del desarrollo de la aplicación sobre un sistema Windows.

El Anexo F presenta los archivos de configuración de OSCAR que fueron modificados para que se adapten a los requerimientos de hardware necesarios para la construcción del *cluster*.

El Anexo G presenta el costo de desarrollar e implementar el *cluster* del presente Proyecto de Titulación.

CAPÍTULO 1. INTRODUCCIÓN A LA TECNOLOGÍA *CLUSTER*

En este capítulo se realiza una breve revisión de las arquitecturas paralelas disponibles en la actualidad, se describe la tecnología *cluster*, se realiza un breve análisis de los componentes de hardware y de software que se usan en la conformación de *clusters*, se describen las diferentes tecnologías de red y se mencionan algunas herramientas y ambientes de programación, y algunas herramientas de administración que pueden ser usadas en *clusters*. Finalmente, se presentan algunos ejemplos de *clusters* implementados.

1.1. INTRODUCCIÓN

La cantidad de información que una aplicación puede llegar a manejar hoy en día, puede ser tal que el poder de procesamiento del que se disponga, resulte un limitante para conseguir procesar toda la información de una manera óptima y eficiente.

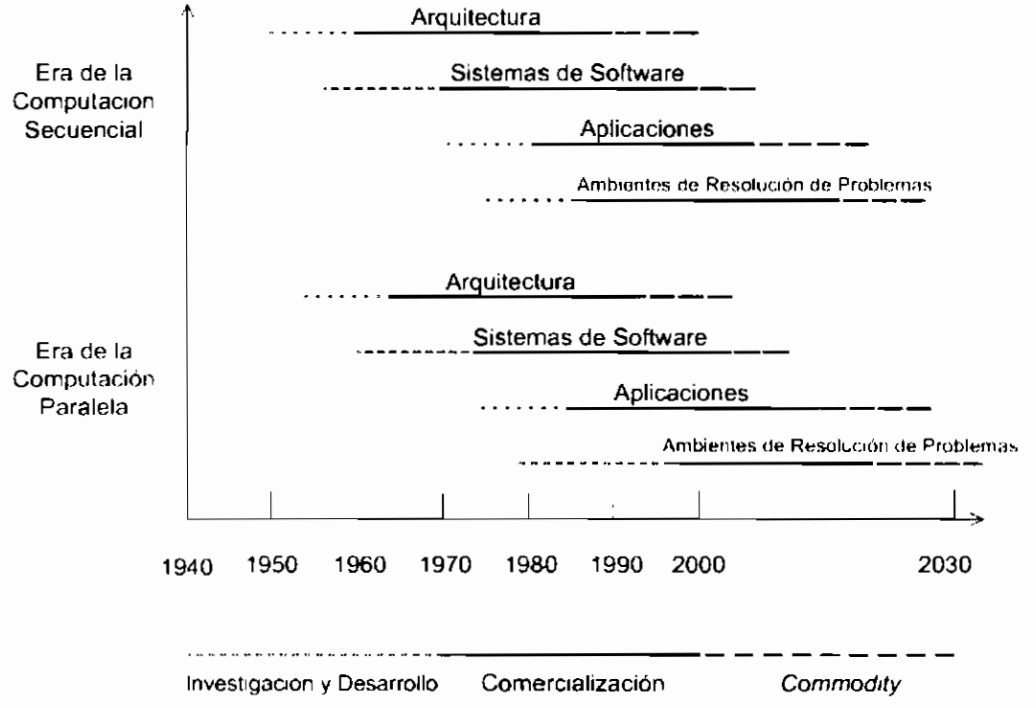
Una posible solución a este inconveniente es utilizar un supercomputador. En un supercomputador se dispone de una gran capacidad de procesamiento, se utilizan uno o más procesadores y componentes de alta velocidad. Sin embargo, la desventaja de esta solución radica en el costo del equipo y las limitaciones propias de los materiales que conforman los componentes, las limitaciones de los materiales limitan la velocidad que los componentes pueden llegar a alcanzar.

Otra solución es el interconectar múltiples procesadores con lo que se consigue satisfacer la necesidad de procesamiento a un costo relativamente bajo.

El desarrollo de nuevas tecnologías de hardware y software han provocado que la industria de la computación se encuentre en constante crecimiento. La evolución de la computación puede dividirse en dos eras:

- La era de la computación secuencial.
- La era de la computación paralela.

La computación paralela surge debido a que el paralelismo es la mejor solución para superar las limitaciones en velocidad de un procesador. El desarrollar y producir un sistema de velocidad moderada, utilizando arquitectura paralela, resulta más económico que desarrollar y producir un sistema secuencial con un rendimiento equivalente.



Fuente: [1], Capítulo 1, página 5

Figura 1-1. Eras de la Computación

Cada una de las eras de la computación empieza con el desarrollo de la arquitectura del hardware, luego se desarrolla el software del sistema, posteriormente se procede al desarrollo de las aplicaciones para luego centrarse

en los Ambientes de Resolución de Problemas[1] (PSE – *Problem Solving Environment*). Cada etapa que conforma la era de la computación se desarrolla en tres fases: una fase de investigación y desarrollo (R&D – *Research and Development*), una fase de comercialización y una fase de popularización (*commodity*[2]). En la Figura 1-1 se muestra una revisión de la evolución de las eras de la computación.

1.2. ARQUITECTURAS PARALELAS

En los últimos años han surgido nuevos sistemas de alto rendimiento; estos sistemas varían en la manera en la que usan tanto procesadores como memorias y en la forma de interconectarlos. Entre estos sistemas se pueden citar:

- Procesadores Masivamente Paralelos (MPP – *Massively Parallel Processors*).
- Multiprocesadores Simétricos (SMP – *Symmetric Multiprocessors*).
- Acceso No Uniforme a Memoria con Coherencia en Caché (CC-NUMA *Cache-Coherent Non-Uniform Memory Access*).
- Sistemas Distribuidos.
- *Clusters*.
- *Grids*.

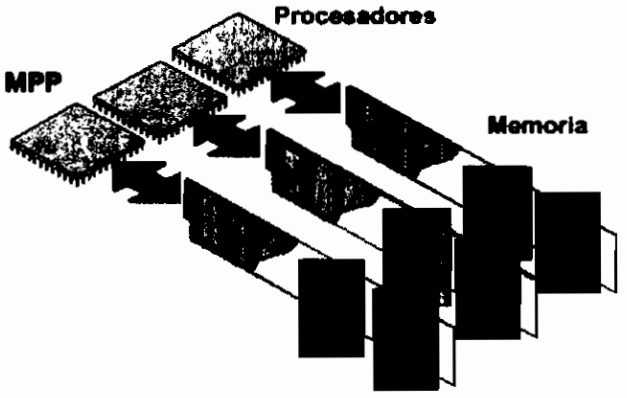
1.2.1. MPP

Es un sistema de procesamiento paralelo con una arquitectura que no comparte nada. Está compuesto por nodos independientes que están interconectados a través de una red de alta velocidad. Cada nodo posee memoria y uno o más procesadores, además ejecuta su propia copia del sistema operativo. Para la

[1] Un PSE es un sistema que provee las ventajas computacionales, tanto en hardware como en software, requeridas para resolver un problema específico. Dispone de métodos avanzados de solución, selección automática y semiautomática de métodos de solución e incorpora de forma fácil métodos novedosos de solución.

[2] *Commodity* es un término utilizado para describir un producto bastante conocido en el mercado. Un producto que se ha popularizado entre la población en general.

comunicación entre los nodos se utiliza paso de mensajes. En la Figura 1-2 se muestra una representación de la arquitectura MPP.

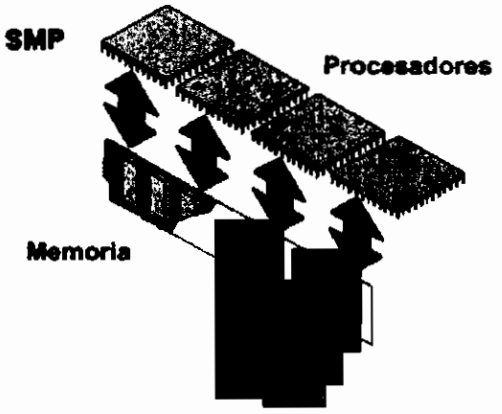


Fuente: http://www.answers.com/topic/massively-parallel

Figura 1-2. Arquitectura MPP

1.2.2. SMP

Es una arquitectura en la que se dispone de una memoria común compartida. Utiliza entre 2 y 64 procesadores [1]. Todos los procesadores comparten la memoria común y los dispositivos de entrada/salida. En todo el sistema sólo se ejecuta una única copia del sistema operativo. En la Figura 1-3 se muestra una representación de esta arquitectura.



Fuente: http://www.answers.com/topic/massively-parallel

Figura 1-3. Arquitectura SMP

1.2.3. CC-NUMA

Es un sistema multiprocesador escalable. Posee una arquitectura de acceso no uniforme a memoria con coherencia en caché. Debido a que la memoria está distribuida físicamente, no se garantiza que las operaciones de acceso a los datos siempre se satisfagan al mismo tiempo. El término coherencia en caché se refiere a que cualquier variable que vaya a ser usada tenga un valor consistente en todos los procesadores.

1.2.4. SISTEMAS DISTRIBUIDOS

Son redes convencionales de computadores independientes. Cada nodo ejecuta su propio sistema operativo. Para la comunicación entre los nodos utilizan paso de mensajes, IPC[3] y RPC[4].

1.2.5. *CLUSTERS*

Son una colección de computadores interconectados con alguna tecnología de red. Poseen una colección integrada de recursos. Para la comunicación entre los nodos utilizan paso de mensajes. En la Sección 1.4 se presenta una descripción más detallada de los *clusters.*

1.2.6. *GRIDS*

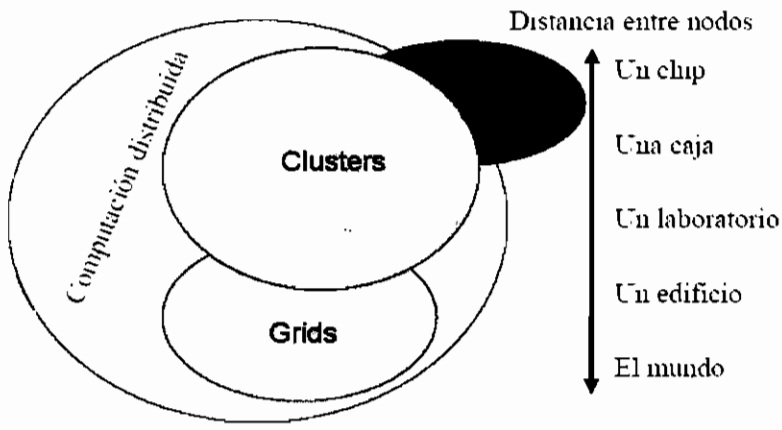
Un *grid* es la denominación que se le asigna a un conjunto de recursos computacionales heterogéneos distribuidos, pertenecientes a diferentes organizaciones. Un *grid* se refiere a un conjunto de computadores que trabajan en conjunto a través de una WAN o del Internet. Este conjunto puede encontrarse

[3] IPC (*InterProcess Communication*): Es la capacidad del sistema operativo que permite que un proceso se comunique con otro. El proceso puede estar corriendo sobre el mismo computador o en un computador diferente conectado a través de una red.

[4] RPC (*Remote Procedure Call*): Permite que los programas llamen a subrutinas que se ejecutan en un sistema remoto.

distribuido dentro de un edificio, a lo largo de un país o, incluso, por uno o varios continentes. La computación *grid* crea organizaciones virtuales que permiten compartir recursos distribuidos geográficamente, asumiendo la ausencia de una localidad central o un control central, cuyo objetivo es el permitir resolver problemas complejos. Las organizaciones virtuales pueden estar compuestas por departamentos de una corporación ubicados en una misma localidad física, o por grupos de personas de diferentes organizaciones que se encuentran distribuidas por todo el mundo.

En la Figura 1-4 se muestra una idea básica de la separación que puede existir entre los nodos que conforman algunas de las arquitecturas paralelas descritas en los párrafos anteriores.



Fuente: [2], página 9

Figura 1-4. Distancias entre los nodos que conforman las arquitecturas paralelas

1.3. MOTIVACIONES DE LA COMPUTACIÓN PARALELA DE BAJO COSTO

Durante la década de los años 80, se pensaba que para incrementar el rendimiento de un computador, se requería crear procesadores más eficientes y más veloces [1]. La introducción del procesamiento paralelo cambió esta concepción; en esencia, se puede conseguir procesamiento paralelo,

interconectando dos o más computadores. Desde principios de los años 90, muchas empresas se han dedicado a fabricar computadores paralelos especializados y costosos.

En la actualidad, la tendencia es comenzar a utilizar sistemas no muy costosos de propósito general conformados por computadores personales, en lugar de las costosas plataformas especializadas como Cray/SGI T3E [3].

Una alternativa muy popular, en lugar de utilizar plataformas de computación paralela costosas, es utilizar *clusters* para desarrollar, depurar y ejecutar aplicaciones paralelas. Uno de los factores que contribuyó al uso de *clusters* es la estandarización de la mayoría de herramientas usadas para desarrollar aplicaciones paralelas.

A continuación se menciona una serie de razones por las que se prefiere usar un *cluster* en lugar de computadores paralelos especializados [1,4]:

- En la actualidad, las estaciones de trabajo y los computadores personales[5] se han convertido en computadores muy poderosos, cuyo rendimiento se incrementa año tras año.
- Los nuevos protocolos y tecnologías de red han reducido los retardos y han incrementado el ancho de banda para las comunicaciones.
- Es más fácil integrar un *cluster* a una red que integrar un computador paralelo.
- El tiempo de utilización de un computador personal es relativamente bajo.
- Las herramientas de desarrollo para estaciones de trabajo son más maduras en comparación con las soluciones propietarias para computadores paralelos.

[5] Un computador personal es una máquina de propósito general diseñada para un amplio rango de tareas; mientras que una estación de trabajo suele ser diseñada para satisfacer los requerimientos de un número de aplicaciones más específicas.

- El crecimiento de un *cluster* es realmente sencillo, sólo basta incrementar la cantidad de memoria, o colocar más procesadores en un nodo, o incrementar el número de nodos.

1.4. *CLUSTERS*: BENEFICIOS, CLASIFICACIÓN Y ARQUITECTURA

Un *cluster* es una colección de computadores personales o de estaciones de trabajo, los cuales están interconectados a través de alguna tecnología de red. Un *cluster* trabaja como una colección integrada de recursos y provee una única imagen del sistema. Cada nodo posee un sistema formado por uno o más procesadores, memoria, dispositivos de entrada/salida y sistema operativo.
Un *cluster* puede estar conformado por nodos dedicados o por nodos no dedicados.

En un *cluster* con nodos dedicados, los nodos no disponen de teclado, mouse y monitor y su uso está exclusivamente dedicado a realizar tareas relacionadas con el *cluster*; en este *cluster*, el acceso se realiza mediante un nodo o un conjunto de nodos asignados a esta tarea, los cuales disponen de teclado, mouse y monitor.

En un *cluster* con nodos no dedicados, cada nodo dispone de teclado, mouse y monitor, y su uso no está limitado a realizar tareas relacionadas con el *cluster*. El *cluster* hace uso de los ciclos de reloj no utilizados por el usuario del nodo para realizar sus tareas. El acceso al *cluster* puede realizarse desde cualquier nodo.

Usualmente se puede hacer referencia al término *cluster* como una Red de Estaciones de Trabajo (NOW – *Network Of Workstations*) o un Conglomerado de Estaciones de Trabajo (COW – *Cluster Of Workstations*).

Debido al interés existente en la tecnología *cluster*, en 1999 se formó el TFCC [5] (*IEEE Computer Society Task Force on Cluster Computing*).

1.4.1. BENEFICIOS DE LA TECNOLOGÍA *CLUSTER*

Las aplicaciones paralelas escalables requieren: buen rendimiento, baja latencia, comunicaciones que dispongan de gran ancho de banda, redes escalables y acceso rápido a archivos. Un *cluster* puede satisfacer estos requerimientos usando los recursos que tiene asociados a él.

Los *clusters* ofrecen las siguientes características a un costo relativamente bajo:

- Alto Rendimiento (*High Performance*).
- Alta Disponibilidad (*High Availability*).
- Alta Eficiencia (*High Throughput*).
- Escalabilidad.

La tecnología *cluster* permite a las organizaciones incrementar su capacidad de procesamiento usando tecnología estándar, tanto en componentes de hardware como de software que pueden adquirirse a un costo relativamente bajo.

1.4.2. CLASIFICACIÓN DE LOS *CLUSTERS*

Los *clusters* pueden clasificarse en base a sus características. Se pueden tener *clusters* de alto rendimiento (HPC – *High Performance Clusters*), clusters de alta disponibilidad (HA – *High Availability*) o *clusters* de alta eficiencia (HT – *High Throughput*).

1.4.2.1. *High Performance*

Son *clusters* en los cuales se ejecutan tareas que requieren de gran capacidad computacional. El llevar a cabo estas tareas puede comprometer los recursos del *cluster* por largos periodos de tiempo.

1.4.2.2. *High Availability*

Son *clusters* cuyo objetivo de diseño es el de proveer disponibilidad y confiabilidad. Estos *clusters* tratan de brindar la máxima disponibilidad de los servicios que ofrecen. La confiabilidad se provee mediante software que detecta fallos y permite recuperarse frente a los mismos, mientras que en hardware se evita tener un único punto de fallos.

1.4.2.3. *High Throughput*

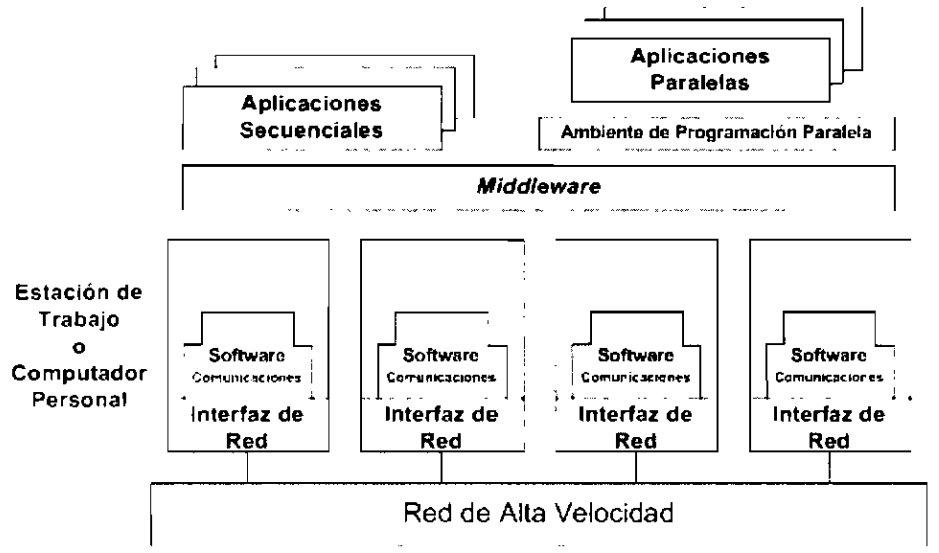
Son *clusters* cuyo objetivo de diseño es el ejecutar la mayor cantidad de tareas en el menor tiempo posible.

1.4.3. ARQUITECTURA DE LOS *CLUSTERS*

Un *cluster* está constituido por los siguientes componentes:

- Computadores.
- Sistemas operativos.
- Redes de alto rendimiento.
- Tarjetas de red.
- Servicios y protocolos de comunicación rápida.
- *Middleware*.
- Herramientas y ambientes de programación paralela.
- Aplicaciones:
  - Secuenciales.
  - Paralelas o distribuidas.

La arquitectura típica de un *cluster* se muestra en la Figura 1-5.

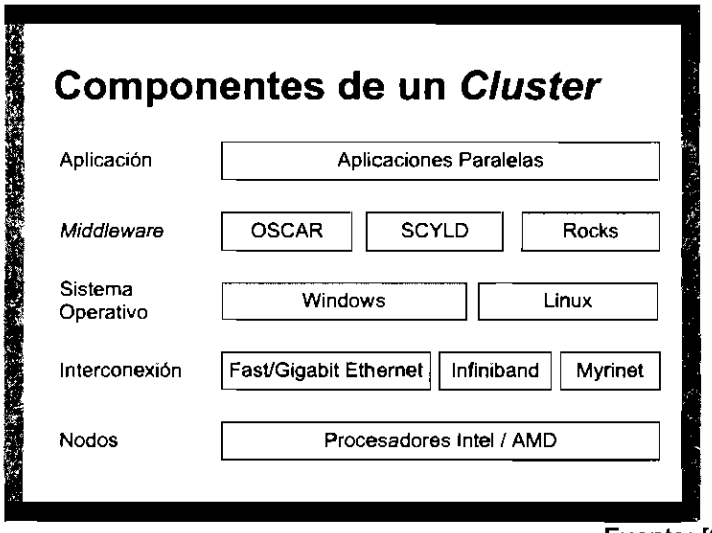


Fuente: [1], Capítulo 1, página 10

Figura 1-5. Arquitectura de un *cluster*

1.5. COMPONENTES DE LOS *CLUSTERS*

En la construcción de *clusters* se puede recurrir a varios componentes de hardware y software que se encuentran disponibles en el mercado. En la Figura 1-6 se muestran algunos ejemplos de componentes de una arquitectura típica de un *cluster*.



Fuente: [6], página 6

Figura 1-6. Componentes de un *cluster*

1.5.1. PROCESADORES

En la actualidad, varios fabricantes ofrecen diferentes tipos de procesadores. Entre los más utilizados en sistemas *clusters* se puede mencionar a: *Opteron* y *Athlon* de AMD, *Itanium* 2, *Xeon* y *Pentium* 4 de Intel, *Alpha EV8* de HP y Compaq, *PA-RISC* de HP, *PowerPC* de IBM, *UltraSPARC* de Sun, *G5* de Macintosh.

Los procesadores pueden estar basados en una de las siguientes arquitecturas: CISC (*Complex Instruction Set Computer*), RISC (*Reduced Instruction Set Computer*), Superescalares, VLIW (*Very Long Instruction Word*), Vectoriales, entre otros.

1.5.2. MEMORIA Y CACHÉ

La cantidad de memoria que un *cluster* requiere depende de las aplicaciones que se van a ejecutar en éste. Si no se dispone de la suficiente memoria por nodo, se puede impedir que algunas tareas se ejecuten completamente.

1.5.2.1. Memoria

Un computador utiliza la memoria (RAM) para mantener las instrucciones y los datos temporales necesarios para realizar tareas.

1.5.2.1.1. Tipos de RAM

En la Tabla 1-1 se muestran algunas memorias que se encuentran disponibles en el mercado.

	Nombre	Velocidad efectiva [MHz]	Rendimiento Máximo [Gbyte/s]
SDRAM133	PC133	133	1.06
DDR266	PC2100	133	2.1
DDR266-Dual	PC2100	133	4.2
DDR333	PC2700	166	2.7
DDR400	PC3300	200	3.3
RDRAM 400	PC800	400	1.6
RDRAM 400-Dual	PC800	400	3.2
RDRAM 533	PC1066	533	2.1
RDRAM 533-Dual	PC1066	533	4.2

Fuente: [7], página 57

Tabla 1-1. Memorias RAM disponibles

1.5.2.2. Caché

La memoria caché es una memoria relativamente pequeña (normalmente menor a 1MB) de alta velocidad, ubicada muy cerca del procesador. La memoria caché está diseñada para proporcionar al procesador los datos e instrucciones que se solicitan con más frecuencia. La recuperación de los datos en la memoria caché toma una fracción del tiempo que toma el acceder a los mismos desde la memoria principal, por lo que el tener una memoria caché puede ahorrar mucho tiempo. Si la información no está en la memoria caché, se debe recuperar de la memoria principal, pero la verificación en memoria caché toma tan poco tiempo que bien vale la comprobación de existencia de la información en la memoria caché.

El concepto detrás de la memoria caché es la regla "80/20" que establece que aproximadamente un 20% del código y los datos de los programas, se utiliza el 80% del tiempo. En forma inversa, el 80% restante de los datos en el sistema se utiliza aproximadamente el 20% del tiempo [7]. La memoria caché tiene sentido

debido a que hay una gran posibilidad de que el código y los datos que el procesador está utilizando ahora se vuelvan a requerir en el futuro cercano.

1.5.2.2.1. Niveles de Memoria Caché

Actualmente, la mayoría de memoria caché está incorporada en el circuito integrado del procesador; sin embargo, es posible encontrar otro tipo de configuraciones. Un sistema puede tener la memoria caché localizada dentro del circuito integrado del procesador, justo fuera del procesador en la tarjeta madre y/o puede tener una ranura de memoria caché cerca del procesador, que puede contener un módulo de memoria caché. Sin importar la configuración, cualquier memoria caché tiene un nivel asignado de acuerdo con su proximidad al procesador. La memoria caché que está más cercana al procesador se llama memoria caché de nivel uno (L1), el siguiente nivel de memoria caché se denomina L2.

1.5.3. DISCO Y DISPOSITIVOS DE ENTRADA/SALIDA (I/O[6])

Las aplicaciones que requieren procesar una gran cantidad de datos, requieren mayor velocidad del procesador y rapidez de acceso al disco duro. El cuello de botella está ubicado en los dispositivos de I/O. Una manera de obtener mejoras en el rendimiento de los dispositivos de I/O es el realizar las operaciones I/O en paralelo, para lo cual se requiere un sistema de archivos en paralelo basado en RAID[7] en hardware o software. Los dispositivos RAID pueden llegar a ser costosos, por lo que se puede recurrir al software RAID construido usando los discos asociados a cada nodo del *cluster*.

---

6 I/O: *Input / Output.*

[7] RAID (*Redundant Array of Inexpensive Disks*): Es una tecnología que forma un arreglo de dos o más discos duros para crear un dispositivo lógico que provea tolerancia a fallos y mejore el rendimiento.

1.5.4. BUS DEL SISTEMA

El bus de un computador es el mecanismo de transmisión de datos desde una parte del computador hacia otra. El bus conecta todos los dispositivos del computador con el procesador y la memoria. En la Tabla 1-2 se muestra un resumen de las especificaciones de algunos de los diferentes buses.

Bus	Cantidad de Bits	Velocidad [MHz]	Ancho de Banda [Mbyte/s]
ISA 8 bits	8	8.3	7.9
ISA 16 bits	16	8.3	15.9
EISA	32	8.3	31.8
VLB[8]	32	33	127.2
PCI	32	33	127.2
PCI 2.1 64 bits	64	66	508.6
PCI X	64	133	1017.3
AGP	32	66	254.3
AGP (2x)	32	66 x 2	508.6
AGP (4x)	32	66 x 4	1017.3

Fuente: http://www.computerhope.com/help/bus.htm

Tabla 1-2. Buses del Sistema

1.5.5. INTERCONEXIÓN DEL *CLUSTER*

Los nodos de un *cluster* se comunican mediante redes de alta velocidad.

1.5.5.1. Protocolos de Comunicación

Un protocolo de comunicaciones define las reglas y convenciones que serán usadas por dos o más computadores para intercambiar información en una red. Los protocolos de comunicación pueden clasificarse de acuerdo a varios criterios:

[8] VLB: *VESA Local Bus.*

- Orientados a Conexión y No Orientados a Conexión.
- Mediante varios niveles de confiabilidad, incluyendo garantía total de arribo (confiables) o ninguna garantía (no confiables).
- Sincrónicos o asincrónicos.
- Por el número de copias de datos intermediarios entre *buffers*, los cuales pueden ser cero, uno o más.

En los *clusters* pueden usarse varios protocolos: pueden usarse protocolos de red tradicionales que fueron diseñados originalmente para Internet o protocolos diseñados específicamente para la comunicación del *cluster.*

1.5.5.1.1. Protocolos de Internet

El Protocolo Internet (IP – *Internet Protocol*) es un estándar *de facto*. IP ofrece un servicio de entrega de mensajes del tipo mejor esfuerzo entre dos computadores que tengan una dirección IP. Sobre IP, en capa de transporte, se construyeron dos protocolos: el Protocolo de Control de Transmisión (TCP – *Transmission Control Protocol*) y el Protocolo de Datagramas de Usuario (UDP – *User Datagram Protocol*). TCP ofrece un servicio confiable orientado a conexión entre dos equipos en una red. UDP es un servicio no confiable no orientado a conexión. Las primeras librerías de mensajes usadas en *clusters* se basaron en los protocolos UDP y TCP y el estándar de *sockets*[9] para TCP y UDP.

Los protocolos TCP y UDP se implementan típicamente usando uno o más *buffers* y con la ayuda de los servicios del sistema operativo. Para enviar un mensaje, la aplicación de usuario construye el mensaje en la memoria de usuario, luego realiza un pedido al sistema operativo para copiar el mensaje en el *buffer* del sistema. Se requiere una interrupción del sistema para que el sistema operativo intervenga antes que el mensaje se copie al *buffer* de la tarjeta de red y pueda ser enviado fuera del hardware de red. Cuando se recibe un mensaje a través del hardware de red, el hardware copia el mensaje a la memoria de sistema. Una

[9] Un *socket* es una abstracción de software compuesto de una dirección de capa red y un puerto de capa transporte del modelo ISO/OSI (*International Standards Organization / Open Systems Interconnection*); sin embargo, fueron diseñados para utilizarse bajo arquitecturas TCP/IP.

protocolo pedido-respuesta. Este protocolo se denomina de cero copias, debido a que transfiere el mensaje desde la memoria de usuario de un proceso, a la memoria de usuario de otro proceso, sin realizar la copia a la memoria del sistema.

Mensajes Rápidos (*Fast Messages*):

Fueron desarrollados por la Universidad de Illinois. Es un protocolo similar a Mensajes Activos. Mensajes Rápidos [9] garantiza que todos los mensajes arriben y que el arribo sea ordenado, incluso si la red sobre la cual se ejecuta no dispone de esta característica. Además, realiza control de flujo para asegurar que un transmisor rápido no sature a un receptor lento, causando pérdida de mensajes.

Comunicación con Correspondencia a Memoria Virtual (VMMC – *Virtual Memory-Mapped Communications*):

Es el protocolo desarrollado por el proyecto SHRIMP [10] de la Universidad de Princeton. El objetivo de VMMC es tratar a los mensajes como lecturas y escrituras dentro del sistema de memoria virtual a nivel del usuario. VMMC usa hardware especialmente diseñado, que permite al interfaz de red el monitorear cambios en la memoria del equipo local y actualizar automáticamente la memoria de los equipos remotos. VMMC es un ejemplo de un paradigma conocido como Memoria Compartida Distribuida (DSM – *Distributed Shared Memory*). En sistemas DSM la memoria está distribuida entre los nodos del sistema, sin embargo, los procesos ven las localidades de memoria compartida como idénticas y realizarán lecturas y escrituras en dichas localidades.

Arquitectura de Interfaz Virtual (VIA – *Virtual Interface Architecture*):

VIA es un estándar de comunicaciones que combina muchas de las características de varios proyectos académicos. El consorcio que desarrolló el estándar está conformado por Intel, Compaq y Microsoft y otras industrias y universidades. VIA se basa en el concepto de una interfaz de red virtual. Antes de que un mensaje pueda ser enviado, los *buffers* de recepción y de transmisión se asignan y se asocian a localidades físicas de memoria. Después de la asignación y asociación de los *buffers* no se requieren llamadas al sistema. La aplicación

puede escoger esperar una confirmación de operación completa o puede continuar su funcionamiento mientras el mensaje empieza a ser procesado.

Las implementaciones de VIA pueden clasificarse como nativas o emuladas. Una implementación nativa se deshace de una porción del procesamiento requerido para enviar o recibir mensajes, colocando esta porción en un hardware especial ubicado en la tarjeta de red. Cuando un mensaje arriba, la tarjeta de red requiere menos trabajo para copiar el mensaje en la memoria del usuario. Con una implementación emulada, el procesador anfitrión realiza el procesamiento de enviar y recibir mensajes. A pesar de que el procesador se usa en ambos casos, la implementación emulada presenta menos sobrecarga que TCP/IP. Sin embargo, los servicios provistos por VIA son diferentes que los provistos por TCP/IP. VIA no garantiza un arribo confiable de datos.

Existe una versión de MPI para VIA llamada MVICH.

1.5.5.2. Productos de Hardware

1.5.5.2.1. Ethernet, Fast Ethernet y Gigabit Ethernet

Ethernet ha sido la tecnología más usada en redes de área local. La transmisión se realiza a 10 Mbps. El ancho de banda no es suficiente para usarlo en computación *cluster*, debido a que no es comparable con el poder computacional de los computadores actuales. Fast Ethernet provee una tasa de transmisión de 100 Mbps por un costo bajo. Gigabit Ethernet provee una tasa de transmisión de 1 Gbps. Ethernet y algunos productos de Fast Ethernet se basan en el concepto de un dominio de colisión. Dentro de un dominio de colisión, el canal de comunicaciones es compartido por lo que sólo un nodo puede transmitir datos al mismo tiempo. Si otro nodo trata de enviar datos al mismo tiempo, ocurre una colisión, los dos mensajes se pierden y ambos reintentan enviar sus datos después de un tiempo de espera aleatorio.

Si se usa un *hub* Ethernet o Fast Ethernet, sólo un nodo puede transmitir al mismo tiempo. Con un *hub switched*, cada nodo se encuentra en diferente dominio de colisión. Para computación *cluster* se prefieren los *switches*, debido a que más de un nodo puede transmitir al mismo tiempo. La gran mayoría de los productos Gigabit Ethernet, están basados en *switches* punto a punto de alta velocidad en los que cada nodo se encuentra en un dominio de colisión separado.

Los productos Fast Ethernet proveen la capacidad necesaria para varias aplicaciones de *clusters*. La mayoría de computadores nuevos vienen con un adaptador Fast Ethernet, y se pueden conseguir *switches* por menos de $30[10] USD. Los productos Gigabit Ethernet son más costosos que los Fast Ethernet, pero generalmente menos costosos que otros productos de red *gigabit*.

1.5.5.2.2. Modo de Transferencia Asíncrono (ATM – Asyncronous Transfer Mode)

ATM es una tecnología de conmutación de celdas. Usa la idea de circuitos virtuales para ofrecer un servicio orientado a conexión. ATM ofrece varias clases de entrega de datos. Algunas clases están diseñadas para comunicación por voz y por video en tiempo real, los cuales requieren garantía de entrega. Varias organizaciones usan ATM como tecnología de *backbone*[11]. Es utilizado en ambientes LAN y WAN. Se basa en la transferencia de paquetes de tamaño fijo denominados celdas[12]. El medio de transmisión puede ser cable de cobre o fibra óptica. Los productos ATM ofrecen un gran ancho de banda en comparación con tecnologías similares, sin embargo su costo es relativamente alto.

1.5.5.2.3. Interfaz Escalable Coherente (SCI – Scalable Coherent Interface)

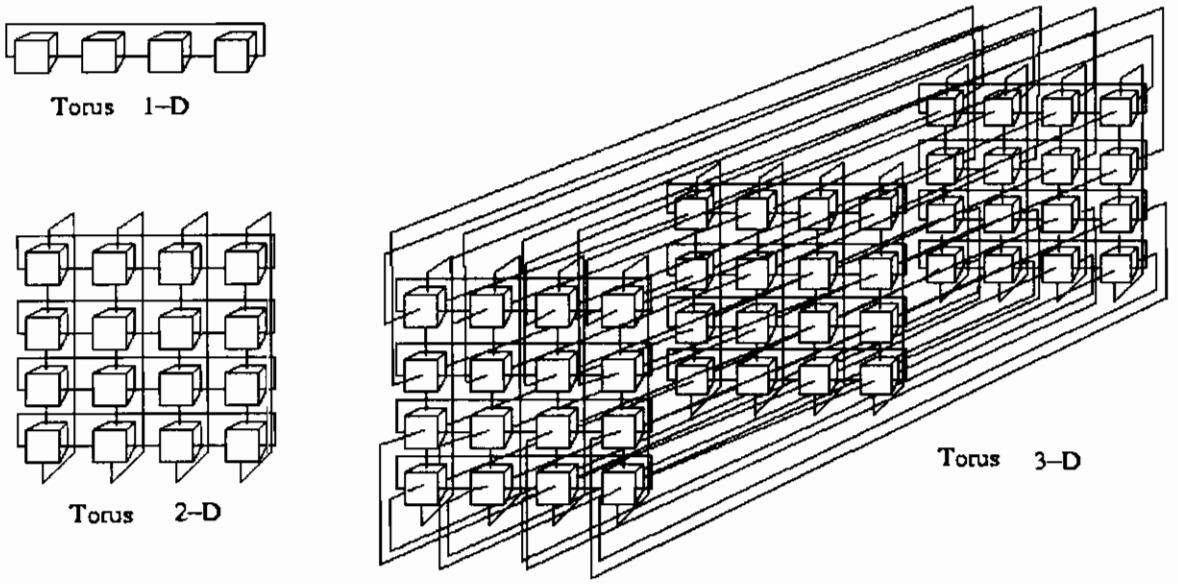
SCI se basa en un anillo que interconecta a los equipos. El anillo siempre está activo, si no hay transferencia de datos se envían mensajes sin información, con longitud cero. La longitud de la información en el paquete tiene tamaños fijos de 0,

[10] Dato tomado el 30 de marzo del 2005 de www.tecnomega.com.

[11] Parte de una red que actúa como ruta primaria para el tráfico.

[12] Una celda está conformada por 53 bytes.

16, 64 y 256 bytes y la cabecera es de 16 o 32 bytes; además se termina el paquete con un código CRC de 2 bytes. El tener un paquete de tamaño fijo permite que la recepción y verificación de los datos sea rápida. La red SCI puede arreglarse como un *torus* de 1, 2 ó 3 dimensiones; la Figura 1-8 muestra los tres diferentes arreglos de una red SCI. La red *torus* tiene sus inconvenientes, si un adaptador en un nodo falla, éste incapacita a todos los nodos que comparten el anillo. Cabe mencionar que debido al tipo de topología, no es posible agregar o remover un número arbitrario de nodos en el *cluster*. La tasa de transmisión está entre los 400 y 1000 Mbyte/s.



Fuente: http://www.top500.org/ORSC/2003/sci.html

Figura 1-8. Diferentes arreglos de una red SCI

1.5.5.2.4. cLAN

cLAN fue desarrollado por Giganet con el objetivo de soportar VIA en hardware. Giganet fue la primera industria que proveyó una implementación nativa en hardware del estándar VIA. Los productos cLAN incluyen: adaptadores PCI que soportan velocidades de 1.25 Gbps, *switches* con más de 30 puertos y software para administrar y probar la red *cluster*. Giganet fue adquirida por Emulex [11] en Marzo del 2001.

1.5.5.2.5. ServerNet

Desarrollado por Tandem Computers Incorporated [12]. ServerNet II ofrece interfaces con soporte para VIA en hardware y *switches* de 12 puertos. El soporte de software incluye controladores VIA para Windows y Linux. SeverNet II es un producto comercial que se ofrece en conjunto con los *clusters* de Compaq, por lo que su uso en *clusters* de propósito general está limitado.

1.5.5.2.6. Myrinet

Myrinet es el líder actual de redes de alta velocidad para interconexión de *clusters*. Myricom [13] empezó a vender Myrinet en 1994 con su primera implementación, como una alternativa a Ethernet para conectar los nodos de un *cluster*. Aparte de su gran tasa de transferencia, cerca de 500 Mbyte/s, la principal ventaja es que opera completamente en el espacio del usuario, evitando la interferencia del sistema operativo y los retardos que provienen de él. Myrinet ofrece *switches* de 8, 16, 32 a 128 puertos. Myrinet distribuye su propia versión de MPI. Las tarjetas de red se denominan Lanai. Su principal desventaja es el costo.

1.5.5.2.7. Infiniband

La especificación del estándar Infiniband [14] fue terminada en junio de 2001. Para el año 2002 ya se contaba con algunos productos comerciales basados en este estándar. Infiniband puede usarse para conectar varios componentes dentro de un sistema. Infiniband define un enlace básico de 1 Gbyte/s (Infiniband 1x), además de dos enlaces de 2 Gbyte/s (Infiniband 4x) y 6 Gbyte/s (Infiniband 12x). Se pueden usar dos tipos de conectores: Adaptador de Canal para *Host* (HCA – *Host Channel Adapter*) y Adaptador de Canal para Destino (TCA – *Target Channel Adapter*). Los TCA se usan para conectar subsistemas I/O y los HCA para comunicación entre procesadores. Los HCA y TCA disponen de múltiples puertos que son independientes, permitiendo alta confiabilidad y velocidad. Existen una gran cantidad de compañías que proveen equipos de interconectividad basados en tecnología Infiniband, como Voltaire Inc. [15],

Otro concepto interesante es el soporte de múltiples *threads* de control en un proceso. Este concepto agrega una nueva dimensión en el procesamiento en paralelo: el paralelismo dentro de un proceso. El programar un proceso con múltiples *threads* de control se conoce como *multithreading*.

1.5.6.1. Linux

Originalmente fue desarrollado por Linus Torvalds. Con la ayuda de varios colaboradores se convirtió en un sistema robusto y confiable. La ventaja de Linux es la cantidad de herramientas, librerías y utilidades de software existentes. Permite al programador acceder a la fuente del sistema operativo e implementar las características que necesite. Permite desarrollar controladores de hardware que luego pueden ser puestos a disposición de cualquier persona. Está disponible en Internet sin costo alguno. Los errores conocidos (*bugs*) y problemas que se presentan pueden solucionarse mediante ayuda en el Internet.

El sistema operativo más popular en *clusters* es Linux. Se puede decir que su popularidad en los ambientes *clusters* se debe a tres razones:

- Es gratuito.

- Es un sistema operativo de código abierto. Esta característica permite realizar cambios en el núcleo de ser necesario.

- Varios científicos de la NASA seleccionaron a Linux como el sistema operativo del *cluster Beowulf*[13]. Debido a esto, los *clusters* derivados del sistema *Beowulf* también usan Linux.

[13] Nombre del *cluster* desarrollado por investigadores de la NASA. El nombre de *Beowulf* se lo asignó en honor al héroe de las leyendas medievales, quien derrotó al monstruo *Grendel*.

1.6. *MIDDLEWARE* Y LA IMAGEN ÚNICA DEL SISTEMA

Se define la Imagen Única del Sistema (SSI – *Single System Image*) como una colección de computadores interconectados de tal manera que aparezcan como un recurso unificado. También se puede hacer referencia a la Imagen Única del Sistema como la propiedad de un sistema de ocultar la naturaleza heterogénea y distribuida de los recursos disponibles y presentarlos, tanto a los usuarios como a las aplicaciones, como un único recurso unificado.

El *middleware* se encuentra ubicado entre el sistema operativo y la capa de usuario. El *middleware* de un *cluster* está constituido por dos subcapas:

- La infraestructura SSI.
- La infraestructura de disponibilidad del sistema (SAI – *System Availability Infraestructure*).

La infraestructura SSI interconecta los sistemas operativos de todos los nodos para ofrecer un acceso unificado a los recursos del sistema. La infraestructura de disponibilidad del sistema brinda soporte para servicios como: punto de comprobación, recuperación de fallos y tolerancia a fallos.

1.6.1. CARACTERÍSTICAS DE SSI

- Cada SSI tiene un límite.
- El soporte de SSI puede existir a diferentes niveles dentro del sistema.

Un subsistema puede provocar que una colección de computadores interconectados aparezca como un gran computador. Si una operación se realiza dentro del límite del subsistema, éste provoca la ilusión de un supercomputador, pero si se realiza fuera del límite aparecerá como un conjunto de computadores conectados. Otro subsistema puede provocar que el mismo conjunto de computadores sea visto como un sistema de almacenamiento o base de datos.

1.6.2. NIVELES QUE SOPORTAN SSI

Se puede mencionar que el SSI es una ilusión, que presenta una colección de recursos como un único recurso de mayor capacidad. Tanto el SSI como los servicios de disponibilidad del sistema pueden ser ofrecidos por una o más de los siguientes niveles:

- Hardware.
- Núcleo del sistema operativo.
- *Middleware*.
- Aplicación.

1.6.2.1. Hardware

Existen sistemas como hardware DSM (*Distributed Shared Memory*) o Canal de Memoria [9] de Digital/Compaq que ofrecen SSI a nivel de hardware y permiten al usuario tener una vista del *cluster* como un sistema de memoria compartida.

1.6.2.2. Núcleo del Sistema Operativo

El sistema operativo del *cluster* debe soportar la ejecución eficiente de aplicaciones paralelas en un ambiente compartido con aplicaciones secuenciales. Además, el sistema operativo debe soportar planificación grupal (*gang-scheduling*) de programas paralelos, debe identificar los recursos inactivos dentro del sistema (procesadores, memorias y redes), debe ofrecer acceso global a todos ellos y debe dar soporte para migración de procesos con la finalidad de proveer balanceo dinámico de carga y una rápida comunicación entre procesos. El sistema operativo debe asegurar que estas características estén disponibles al usuario sin la necesidad de llamadas o comandos adicionales del sistema.

El SSI permite que todos los recursos físicos sean accesibles desde todos los nodos en el sistema. Cada nodo coopera para presentar la misma vista desde toda interfaz del sistema.

Un SSI a nivel de núcleo puede ahorrar tiempo y dinero debido a que las aplicaciones y los programas existentes no tienen que volver a reescribirse para trabajar en este nuevo ambiente. Adicionalmente, estas aplicaciones correrán sobre cualquier nodo sin requerir trabajo adicional por parte de los administradores, y los procesos puedan migrarse para balancear la carga entre los nodos, y también soportar tolerancia a fallos de ser necesario.

La mayoría de sistemas operativos que soportan SSI están construidos con una capa sobre el sistema existente y efectúan la asignación global de recursos. Esta estrategia consigue que el sistema sea portable, permita las actualizaciones del vendedor de software y reduzca el tiempo de desarrollo. Entre los sistemas operativos que soportan SSI se puede mencionar a MOSIX [20].

1.6.2.3. *Middleware*

Como ya se mencionó, el *middleware* es una capa que reside entre el sistema operativo y las aplicaciones. Este es uno de los métodos más comunes para implantar SSI en un *cluster*. El *middleware* puede incluir un sistema de archivos, ambientes de programación, sistemas de administración y planificación de tareas, puede habilitar la Máquina Virtual de Java[14] (JVM – *Java Virtual Machine*) para *clusters,* etc.

Mediante los sistemas de archivos se consigue proveer un sistema único de almacenamiento provisto por los discos de cada nodo. Además, se asegura que cada nodo tenga la misma vista de los datos.

Como parte del *middleware*, un sistema global de administración de tareas se encarga de la administración de recursos y permite planificar las actividades del sistema y la ejecución de aplicaciones mientras se ofrecen servicios de alta disponibilidad de forma transparente.

[14] Una máquina virtual de Java es aquella que permite utilizar código fuente en diferentes plataformas; mediante el uso de *bytecodes,* los cuales son generados mediante compiladores de Java. Los *bytecodes* se traducen a lenguaje de máquina permitiendo ejecutar las operaciones de bajo nivel.

Un *cluster* con una JVM habilitada permite la ejecución de aplicaciones Java basadas en *threads* sin ninguna modificación.

1.6.2.4. Aplicación

El nivel de aplicación SSI es el superior, y en algunos sentidos el más importante, debido a que esto es lo que el usuario ve. Una herramienta de administración del *cluster* ofrece un único punto de administración y control de los servicios SSI. La herramienta de administración puede construirse con herramientas basadas en GUI ofreciendo una interfaz única para monitorear y controlar el *cluster* como un todo, a los nodos de forma individual, o a componentes específicos del sistema.

Puntos a favor y puntos en contra de cada nivel:

- Un SSI a nivel de hardware puede ofrecer un alto nivel de transparencia, pero debido a su arquitectura rígida, no ofrece la flexibilidad que se requiere para extender y mejorar el sistema.

- Un SSI a nivel de núcleo puede ser ofrecido a todos los usuarios, tanto a desarrolladores de aplicaciones como a usuarios finales. Sin embargo, esta tecnología es costosa de desarrollar y mantener. Además, está limitada por el mercado y le es difícil convivir con las innovaciones tecnológicas del mercado de los sistemas operativos. Se debe considerar que a menos de que todos los componentes estén específicamente desarrollados para soportar SSI, no pueden usarse o ser liberados en el mercado. Debido a esto, un SSI a nivel del núcleo se muestra como riesgoso y como una solución económica no viable.

- Un nivel de aplicación ayuda a realizar SSI parciales. Requiere que cada aplicación sea desarrollada como un componente que conoce del SSI.

- A nivel de *middleware* se presentan problemas, por ejemplo con PVM, cada aplicación debe ser implementada usando un API (*Application*

Program Interface) especial, deber ser construida considerando la funcionalidad que tendrá cada nodo. Esto significa un alto costo tanto en la implementación como en el mantenimiento.

1.6.3. BENEFICIOS DE SSI

- Todos los nodos poseen una única vista de todos los recursos.
- El usuario no necesita conocer donde se ejecutará una aplicación.
- El operador no necesita conocer donde está localizado un recurso.
- El usuario puede trabajar con una interfaz familiar.
- El administrador puede gestionar a todo el *cluster* como una entidad única.
- Los posibles errores que un operador puede cometer se reducen, debido a que el operador usa comandos con sintaxis familiar.
- La administración y el control del sistema pueden realizarse de manera centralizada o descentralizada.
- La administración del sistema se simplifica notablemente; mediante un único comando se pueden realizar acciones que afecten a múltiples recursos, incluso cuando los recursos están dispersos en diferentes sistemas y en diferentes computadores.
- Reduce el tiempo, el esfuerzo y el conocimiento necesario para realizar tareas y permite manipular sistemas más complejos.

1.6.4. SERVICIOS PRINCIPALES DE SSI

SSI proporciona los siguientes servicios:

- Punto único de entrada: Un usuario puede conectarse al *cluster* como un sistema único en lugar de conectarse a los nodos de forma individual como en el caso de un sistema distribuido.

- Jerarquía de archivos única (SFH – *Single File Hierarchy*): Al entrar al sistema, el usuario ve al sistema como una única organización de archivos y directorios bajo el mismo directorio raíz.

- Punto único de administración y control: El *cluster* puede ser monitoreado y controlado desde una única ventana usando una herramienta GUI, de la misma forma como un equipo con Windows es administrado con la herramienta de Administración de Tareas (**taskmgr**).

- Red virtual única: Cualquier nodo puede acceder a cualquier conexión de red a través del dominio del *cluster,* incluso si todos los nodos del *cluster* no están dentro de la misma red física.

- Espacio de memoria único: Provoca la ilusión de memoria compartida de las memorias asociadas con cada nodo del *cluster*.

- Sistema único de administración de tareas: Un usuario puede exponer su trabajo desde cualquier nodo usando un mecanismo de presentación de tareas transparente. Se pueden planificar las tareas para ser ejecutadas en modo interactivo, paralelo o por lotes.

- Interfaz única de usuario: El usuario debería ser capaz de usar el *cluster* a través de un único GUI. La interfaz debe tener la misma vista desde cualquier nodo.

Entre las funciones de la infraestructura de disponibilidad se pueden mencionar las siguientes:

- Espacio único de I/O (SIOS - *Single I/O Space*): Permite que cualquier nodo realice operaciones de Entrada/Salida sobre cualquier periférico local o remoto. En el diseño SIOS, los discos asociados con los nodos del *cluster*, RAID, y los dispositivos periféricos forman un espacio de direcciones único.

- Espacio único de procesos: Los procesos poseen un identificador único dentro del *cluster*. Un proceso en cualquier nodo puede crear procesos hijos en el mismo nodo o en uno diferente o puede comunicarse con cualquier otro proceso en un nodo remoto. El *cluster* debe soportar una administración global de los procesos y permitir la administración y el control de los procesos como si estuvieran corriendo en una misma máquina.

- Migración de procesos y puntos de comprobación: El mecanismo de punto de comprobación permite almacenar el estado de un proceso, y los resultados de cálculos intermedios de forma periódica. Si existe un fallo en un nodo, los procesos en el nodo que falló pueden reiniciarse en otro nodo que se encuentre activo, sin la pérdida de los cálculos. La migración de procesos permite balancear la carga de forma dinámica.

1.6.5. OBJETIVOS DE DISEÑO DEL *MIDDLEWARE*

Los objetivos del diseño del *middleware* se enfocan principalmente en una completa transparencia en la administración de recursos, en rendimiento escalable y en disponibilidad del sistema en el soporte de las aplicaciones de los usuarios.

1.6.5.1. Transparencia completa

La capa SSI debe permitir que el usuario utilice el *cluster* de manera fácil y efectiva sin tener un conocimiento explícito de la arquitectura del sistema. El ambiente operativo se muestra familiar y conocido, además es conveniente de usar. El usuario dispone de una vista global del sistema de archivos, de los procesos y la red.

Los detalles de la administración de recursos y del control de actividades, así como la asignación, el retiro de la asignación y la replicación de recursos son invisibles a los procesos del usuario. Esto permite a los usuarios acceder de

forma transparente a los recursos del sistema como la memoria, los procesadores y la red, sin importar donde se encuentren ubicados.

Ejemplos:

- El usuario puede conectarse en cualquier nodo.
- El administrador del sistema puede instalar o cargar software en cualquier nodo y disponer del software desde cualquier parte del *cluster*.

1.6.5.2. Rendimiento Escalable

Debido a que un *cluster* puede expandirse fácilmente, su rendimiento también debería escalar de manera fácil. Esta escalabilidad debe lograrse sin la necesidad de nuevos protocolos o APIs. Para conseguir un máximo rendimiento, el servicio SSI debe soportar balanceo de carga y paralelismo, mediante la distribución de pedidos a cualquier nodo ligeramente cargado. El *cluster* debe ofrecer estos servicios con una sobrecarga pequeña y también debe asegurar que el tiempo requerido para ejecutar esta operación en el *cluster* no sea mayor que el necesario para ejecutarla sobre un sólo computador.

1.6.5.3. Disponibilidad Mejorada

Debe existir una alta disponibilidad de los servicios del *middleware* todo el tiempo. En cualquier instante, una falla debería ser recuperable sin afectar a las aplicaciones del usuario. Esto puede conseguirse mediante el empleo de tecnología de puntos de verificación y tolerancia a fallos y habilitar la recuperación tipo *rollback*[15].

Cuando los servicios SSI se ofrecen usando la disponibilidad de recursos sobre múltiples nodos, una falla en cualquier nodo no debería afectar la operación del sistema, y un servicio particular debería soportar uno o más de los objetivos de diseño. Por ejemplo, cuando un sistema de archivos está distribuido a través de

[15] *Rollback* se refiere al proceso de restaurar una transacción desde un estado consistente previamente almacenado.

varios nodos con un cierto grado de redundancia, si un nodo falla, esta porción del sistema de archivos podrá migrarse a otro nodo de forma transparente.

1.7. PLANEACIÓN Y ADMINISTRACIÓN

La Planeación y Administración de Recursos (RMS – *Resource Management and Scheduling*) es la actividad de distribuir aplicaciones a través de computadores para maximizar su rendimiento. Adicionalmente, permite un uso eficiente y efectivo de los recursos disponibles. El software que realiza las funciones de RMS consta de dos componentes: un administrador de recursos y un planificador de recursos. El componente de administración de recursos se encarga de los problemas concernientes con la autenticación, y la creación y migración de procesos. El componente de planificación de recursos se encarga de tareas como encolamiento de aplicaciones, localización y asignación de recursos.

La existencia de RMS se justifica por un número de razones, como balanceo de carga, utilización de ciclos (*idle-cycles*) del CPU, proveer sistemas de tolerancia a fallos, administrar el acceso a sistemas, etc. Pero la principal razón para su existencia es la habilidad de proveer un rendimiento eficiente y confiable de las aplicaciones del usuario sobre el sistema que el RMS administra.

La arquitectura básica de RMS es un sistema cliente-servidor. En su forma más simple, cada computador que comparte recursos computacionales ejecuta un demonio[16] servidor. Estos demonios mantienen tablas actualizadas, las cuales almacenan información acerca del ambiente RMS en el cual residen.

Un usuario interactúa con el ambiente RMS a través de un programa cliente, el cual puede ser un explorador Web o una interfaz personalizada de ventanas. La aplicación puede ejecutarse en modo interactivo o por lotes. En el modo por lotes,

[16] En UNIX, se utiliza el término demonio (DAEMON – *Disk And Execution MONitor*) para referirse a un proceso que nunca debería terminar. Puede ser un proceso encargado de administrar un servicio de red o un proceso encargado de administrar un recurso del sistema.

una aplicación que se ejecuta llega a ser una tarea que se publica en el sistema RMS para su procesamiento. Para entregar (*submit*) una tarea al *cluster*, un usuario necesita proveer los detalles de la tarea a través del cliente RMS. Entre estos detalles pueden estar incluidos la localización del ejecutable y los datos de entrada a utilizar, donde colocar la salida, el tipo de sistema, la longitud máxima de la ejecución, si el trabajo requiere recursos secuenciales o paralelos, etc. Una vez que la tarea ha sido asignada al ambiente RMS, éste usa los detalles de la tarea para su colocación, planificación y ejecución en la vía apropiada.

Entre los servicios que el ambiente RMS provee se incluyen:

- Migración de procesos: Permite que un proceso pueda ser suspendido, movido y reiniciado en otro computador dentro del ambiente RMS. Por lo general, la migración de un proceso sucede debido a que un recurso computacional tiende a estar demasiado cargado y existen otros que están libres.

- Punto de verificación: Permite almacenar el estado de un programa y puede usarse para reiniciarlo desde ese punto de ser necesario. Provee confiabilidad en el sistema.

- Buscar ciclos desocupados: Se conoce que entre un 70% y un 90% del tiempo, la mayoría de computadores están desocupados. El sistema RMS puede configurarse para utilizar estos ciclos de CPU.

- Tolerancia a fallos: Mediante monitoreo de tareas y recursos, un sistema RMS puede proveer varios niveles de tolerancia a fallos. En su forma más simple, el soporte de tolerancia a fallos permite que una tarea fallida pueda ser reiniciada después o pueda migrarse a otros recursos del sistema.

- Balanceo de carga: Las tareas pueden distribuirse a través de todos los nodos disponibles. Esto permite un uso eficiente y efectivo de todos los recursos. La migración de procesos puede ser parte de la estrategia de

balanceo de carga, donde se puede obtener un beneficio al mover los procesos de sistemas sobrecargados a otros sistemas que estén ligeramente cargados.

- Colas de aplicación múltiple: Las colas de tareas pueden configurarse para ayudar en la administración de recursos de una organización. Cada cola puede configurarse con atributos particulares. Por ejemplo, ciertos usuarios pueden tener prioridad al ejecutar tareas cortas en lugar de tareas largas. Las colas de tareas pueden configurarse para administrar el uso de recursos especiales como una estación de trabajo de alto rendimiento para gráficos. Las colas en los sistemas RMS pueden ser transparentes a los usuarios; las tareas son asignadas al sistema mediante claves específicas al asignar el trabajo.

1.8. HERRAMIENTAS Y AMBIENTES DE PROGRAMACIÓN

1.8.1. HILOS (*THREADS*)

Los *threads* son un paradigma popular para programación concurrente en máquinas con uno o varios procesadores. En sistemas con múltiples procesadores se utilizan *threads* para usar simultáneamente todos los procesadores disponibles. En sistemas con un procesador, los *threads* se utilizan para usar de forma eficiente los recursos del sistema. Contrario a la creación de procesos, la creación de *threads* no es costosa y es fácil de administrar.

Existe un estándar del IEEE (*Institute of Electrical and Electronics Engineers*) para la interfaz POSIX (*Portable Operating System Interface based on uniX*) de *threads* llamado *pThreads*. La interfaz estándar POSIX para múltiples *threads* se encuentra disponible para computadores personales, estaciones de trabajo, SMPs y *clusters*. El lenguaje de programación Java está construido con soporte para *multithreading*, lo que permite un desarrollo fácil de aplicaciones con múltiples

threads. Los *threads* son extensamente usados en el desarrollo tanto de aplicaciones como de sistemas.

1.8.2. SISTEMAS DE PASO DE MENSAJES

Las librerías para paso de mensajes permiten escribir programas paralelos eficientes para sistemas de memoria distribuida. Estas librerías proveen rutinas para inicializar y configurar el ambiente de mensajes, así como para enviar y recibir paquetes de datos. Los sistemas más populares de paso de mensajes de alto nivel para aplicaciones científicas y de ingeniería son: Máquina Virtual Paralela (PVM – *Parallel Virtual Machine*) del Laboratorio Nacional Oak Ridge [21] e Interfaz de Paso de Mensajes (MPI – *Message Passing Interface*) definido por el Foro MPI [22].

PVM es una librería de paso de mensajes. Puede usarse para desarrollar y ejecutar aplicaciones paralelas en sistemas que están dentro del rango que va desde supercomputadores hasta *clusters* de estaciones de trabajo.

MPI es una especificación de paso de mensajes, diseñada para ser el estándar de computación paralela de memoria distribuida usando paso de mensajes. Esta interfaz intenta establecer un estándar práctico, eficiente, portátil y flexible para el paso de mensajes.

MPI puede considerarse como la amalgama de lo que puede considerarse los mejores aspectos de la mayoría de sistemas de paso de mensajes al tiempo de su concepción[17]. Es el resultado del trabajo desarrollado por el Foro MPI, un comité compuesto por vendedores y usuarios, conformado con la finalidad de definir un estándar para paso de mensajes. El estándar solamente define una librería de paso de mensajes y deja que el desarrollador defina la inicialización y el control de los procesos, entre otras cosas. MPI está disponible para un amplio

[17] MPI fue concebido en el año 1992.

rango de plataformas. Las librerías de MPI y PVM están disponibles para Fortran 77, Fortran 90, ANSI C y C++. También existen interfaces para otros lenguajes.

1.8.3. SISTEMAS DE MEMORIA COMPARTIDA DISTRIBUIDA (DSM)

El paradigma más eficiente y ampliamente usado en sistemas de memoria distribuida es el paso de mensajes. Un problema con este paradigma es la complejidad y dificultad de su programación en comparación con los sistemas de programación de memoria compartida. Un sistema de memoria compartida ofrece un modelo de programación simple y general, pero carece de escalabilidad. Una solución alternativa de costo efectivo es construir un sistema DSM sobre un sistema de memoria distribuida, el cual exhibe un modelo de programación simple y general, y la escalabilidad de los sistemas de memoria distribuida.

DSM permite programar variables compartidas y puede implementarse usando soluciones en hardware o software. Dentro de las implementaciones de DSM en hardware se puede mencionar a Canal de Memoria [9], DASH [1] y Merlin [1]; en software se puede mencionar a TreadMarks [24], DiscoLab [24] y KDSM (*KAIST Distributed Shared Memory*) [25].

1.8.4. DEPURADORES PARALELOS

Para desarrollar aplicaciones de alto rendimiento, correctas y eficientes, es recomendable disponer de algunas herramientas de fácil uso para descripción y depuración.

La cantidad de depuradores paralelos que se pueden usar en plataformas heterogéneas es muy limitada. En 1996 se formó el *High Performance Debugging Forum* (HPDF) [26] como el proyecto *Parallel Tools Consortium*. Este foro desarrolló la especificación de la versión HPD, la cual define la funcionalidad, la semántica y la sintaxis para un depurador paralelo de línea de comandos.

Idealmente, un depurador paralelo debería ser capaz de:

- Administrar múltiples procesos y múltiples *threads* dentro de un proceso.
- Mostrar código fuente y permitir rastrear la pila (*stack*).
- Permitir monitorear objetos, subrutinas y funciones.
- Colocar puntos de parada a nivel de código fuente y de máquina.
- Compartir puntos de parada en un grupo de procesos.
- Definir puntos de evaluación y de observación.
- Mostrar arreglos y sus elementos.
- Manipular variables y constantes.

Existen algunos depuradores paralelos, de entre los cuales se puede citar: Total VIEW [27], GDB (GNU DeBuger) [28] y Eclipse [29].

1.8.5. HERRAMIENTAS DE ANÁLISIS DE RENDIMIENTO

El propósito básico de las herramientas de análisis de rendimiento es ayudar al programador a entender el rendimiento de una aplicación. Estas herramientas permiten localizar y analizar las partes de una aplicación que presenten un rendimiento pobre y creen cuellos de botella. Son muy útiles para entender el comportamiento de aplicaciones secuenciales normales y pueden ser de enorme ayuda cuando se trata de analizar las características de rendimiento de aplicaciones paralelas.

Existen algunas herramientas de análisis de rendimiento, de entre las cuales se puede citar: PAPI [30], Guide [31], SvPablo [32], Vampir [33].

1.8.6. HERRAMIENTAS DE ADMINISTRACIÓN

El monitoreo de un *cluster* es una tarea que puede realizarse mediante herramientas que permiten observar al *cluster*, usando un GUI. Un buen software de administración es crucial para explotar un *cluster* como una plataforma de computación de alto rendimiento.

Existen algunos sistemas de administración de *clusters* que soportan computación paralela. Entre estos se puede mencionar a C3 [34] y Ganglia [35].

1.9. SISTEMAS *CLUSTERS* IMPLEMENTADOS

1.9.1. *BEOWULF*

Beowulf fue construido por Donald Becker y Thomas Sterling en 1994. Fue construido con 16 computadores personales con procesadores Intel DX4 de 200 MHz, que estaban conectados a través de un *switch* Ethernet. El rendimiento teórico era de 3.2 GFlops.

1.9.2. Berkeley NOW

El sistema NOW de Berkeley [36] estuvo conformado por 105 estaciones de trabajo Sun Ultra 170, conectadas a través de una red Myrinet. Cada estación de trabajo contenía un microprocesador Ultra1 de 167 MHz, caché de nivel 2 de 512 KB, 128 MB de memoria, dos discos de 2.3 GB, tarjetas de red Ethernet y Myrinet. En la red se utilizaban múltiples etapas de *switches* Myrinet, cada *switch* con 8 puertos bidireccionales de 160 MByte/s. En abril de 1997, NOW logró un rendimiento de 10 GFlops.

1.9.3. AVALON

Avalon [37] fue construido en 1997 por el Laboratorio Nacional de los Álamos de la Universidad de California. Inicialmente estaba constituido por 70 computadores conectados mediante un *switch* Fast Ethernet. Cada computador estaba conformado por un procesador Alpha de 533 MHz, 128 MB de memoria RAM y 3.2 GB de disco duro. Fue el primer *cluster* en ingresar en la lista de los 500

computadores más veloces del mundo ("TOP 500"[18]), donde ocupó el lugar 315 en el primer semestre de 1998. Posteriormente, Avalon creció a 140 procesadores, llegando a tener 36 GB de memoria RAM, consiguiendo ocupar la posición 114 del "TOP 500" durante el segundo semestre de 1998.

1.9.4. HIDRA

Hidra [38] es un *cluster Beowulf* desarrollado a finales del 2002 y principios del 2003 por el Laboratorio de Computación y Visualización Avanzada del Centro de Apoyo Tecnológico de la Universidad Rey Juan Carlos de España. Está conformado por 40 computadores.

El *cluster* posee las siguientes características:

- Procesadores AMD Athlon de 2 GHz.
- Cada nodo dispone de una memoria DDR-RAM de 512 MB.
- Cada nodo dispone de un disco duro de 40 GB.
- Posee una doble conexión de red, usando Myrinet y Fast Ethernet.
- Utiliza el sistema operativo Linux Red Hat 7.3.
- Utiliza las librerías de comunicación mediante paso de mensajes MPICH y LAM-MPI.

1.9.5. *CLUSTER* GOOGLE

Durante el año 2003, el *cluster* Google [39] llegó a estar conformado por más de 15.000 computadores personales. En promedio, una consulta en Google lee cientos de megabytes y consume algunos billones de ciclos del CPU.

[18] http://top500.org

1.9.6. *CLUSTER* PS2

En el año 2004, en la Universidad de Illinois en Urbana-Champaign, Estados Unidos, se exploró el uso de consolas *Play Station 2* (PS2) en cómputo científico y visualización de alta resolución [40]. Se construyó un *cluster* conformado por 70 PS2; utilizando *Sony Linux Kit* (basado en Linux Kondora y Linux Red Hat) y MPI.

1.9.7. *CLUSTER* X

En la lista "TOP 500" de noviembre de 2004 fue considerado el séptimo sistema más rápido del mundo; sin embargo, para julio de 2005 ocupa la posición número catorce. *Cluster* X [41] fue construido en el Tecnológico de Virginia en el 2003; su instalación fue realizada por estudiantes del Tecnológico. Está constituido por 2200 procesadores Apple G5 de 2.3 GHz. Utiliza dos redes: Infiniband 4x para las comunicaciones entre procesos y Gigabit Ethernet para la administración. *Cluster* X posee 4 Terabytes de memoria RAM y 176 Terabytes de disco duro, su rendimiento es de 12.25 TFlops. Se lo conoce también como Terascale.

1.9.8. MareNostrum

En julio de 2004 se creó el Centro de Supercomputación de Barcelona (BSC), de la Universidad Técnica de Cataluña, España. El BSC creó el *cluster* MareNostrum [42]. En noviembre de 2004 MareNostrum se ubicó en el "TOP 500", como el primer *cluster* más veloz y el cuarto sistema más rápido del mundo; sin embargo, para julio de 2005 se ubicó en la quinta posición. Está conformado por 3564 procesadores PowerPC970 de 2.2 GHz. Utiliza una red Myrinet. Su rendimiento es de 20.53 TFlops.

1.9.9. THUNDER

Thunder [43] fue construido por el Laboratorio Nacional Lawrence Livermore de la Universidad de California. Está conformado por 4096 procesadores Intel Itanium2

Tiger4 de 1.4GHz. Utiliza una red basada en tecnología Quadrics. Su rendimiento es de 19.94 TFlops. Se ubicó en la segunda posición del "TOP 500" durante junio de 2004, luego en la quinta posición en noviembre de 2004 y en la lista de julio de 2005 se ubicó en la séptima posición.

1.9.10. ASCI Q

ASCI Q [44] fue construido en el año 2002 por el Laboratorio Nacional Los Álamos, Estados Unidos. Está constituido por 8192 procesadores AlphaServer SC45 de 1.25 GHz. Su rendimiento es de 13.88 TFlops. Se ubicó en la segunda posición del "TOP 500" durante junio y noviembre de 2003, luego en la tercera posición en junio de 2004, en la sexta posición en noviembre de 2004 y en la doceava posición en julio de 2005.

1.10. REFERENCIAS

[1] BUYYA, Rajkumar: *High Performance Cluster Computing – Volume 1*. Prentice Hall, Primera Edición, Estados Unidos, 1999.

[2] LOPEZ, Guillermo: *Linux Cluster Computing.* http://www.des.udc.es/~gltaboada Último acceso: 27/12/2004

[3] www.cray.com/products/systems/crayt3e/ Último acceso: 27/12/2004

[4] BAKER, Mark: *Cluster Computing White Paper*. Reino Unido, 2000.

[5] TFCC – http://www.ieeetfcc.org/ Último acceso: 25/05/2005

[6] HARGAUGH, Logan: *Building High-Performance Linux Clusters White Paper*. 2004

[7] KINGSTON TECHNOLOGY: *La guía completa de Memoria*.
www.kingston.com/latinoamerica
Último acceso: 12/01/2005

[8] http://now.cs.berkeley.edu/Case
Último acceso: 27/12/2004

[9] http://www.cc.gatech.edu/classes/AY2000/cs4230_spring/FM/userdoc.html
Último acceso: 27/12/2004

[10] http://www.cs.princeton.edu/shrimp
Último acceso: 27/12/2004

[11] http://www.emulex.com
Último acceso: 26/12/2004

[12] http://www.tandem.com
Último acceso: 26/12/2004

[13] http://www.myri.con
Último acceso: 12/05/2005

[14] http://www.infinibandta.org
Último acceso: 12/05/2005

[15] http://www.voltaire.com
Último acceso: 14/06/2005

[16] http://www.topspin.com
Último acceso: 14/06/2005

[17] http://www.mellanox.com
Último acceso: 14/06/2005

[18] http://www.quadrics.com
Último acceso: 12/05/2005

[19] http://www.hp.com/techservers/systems/symc.html
Último acceso: 26/12/2004

[20] http://www.mosix.cs.huji.ac.il/
Último acceso: 14/06/2005

[21] http://www.ornl.gov
Último acceso: 14/06/2005

[22] http://mpi-forum.org
Último acceso: 14/06/2005

[23] http://www.cs.rice.edu/~willy/TreadMarks/overview.html
Último acceso: 14/06/2005

[24] http://discolab.rutgers.edu/dsm/
Último acceso: 14/06/2005

[25] http://camars.kaist.ac.kr/~nrl/team/dsm.html
Último acceso: 14/06/2005

[26] http://www.ptools.org/hpdf
Último acceso: 14/06/2005

[27] http://www.etnus.com/index.html
Último acceso: 14/06/2005

[28] http://www.ncsa.uiuc.edu/UserInfo/Resources/Software/Tools/GDB/
Último acceso: 25/11/2004

[29] http://www.eclipse.org/ptp/
Último acceso: 14/06/2005

[30] http://icl.cs.utk.edu/projects/papi/
Último acceso: 25/11/2004

[31] http://developer.intel.com/software/products/guide/
Último acceso: 25/11/2004

[32] http://www-pablo.cs.uiuc.edu/Project/Pablo/ScalPerfToolsOverview.htm
Último acceso: 25/11/2004

[33] http://www.ncsa.uiuc.edu/UserInfo/Resources/Software/Math/Vampir/
Último acceso: 25/11/2004

[34] http://www.csm.ornl.gov/torc/C3
Último acceso: 14/06/2005

[35] http://ganglia.sourceforge.net
Último acceso: 14/06/2005

[36] CULLER, David; ARPACI-DUSSEAU, Andrea; ARPACI-DUSSEAU Remzi: *Parallel Computing on the Berkeley NOW*. Universidad de California, Berkeley.

[37] http://cnls.lanl.gov/avalon/
Último acceso: 26/12/2004

[38] http://www.urjc.es/cat/hidra
Último acceso: 14/06/2005

[39] BARROSO, Luiz André; DEAN, Jeffrey; HÖZLE, Urs: *Web search for a planet: The Google cluster Architecture.* IEEE Computer Society. 2003.

[40] LIZÁRRAGA, Carlos: *Cómputo de Alto Rendimiento en Clusters de Play Station 2.* http://www.fisica.uson.mx/carlos/PS2Cluster/PS2Cluster.ppt
Último acceso: 12/05/2005

[41] http://computing.vt.edu/research.computing/terascale
Último acceso: 07/07/2005

[42] http://www.bsc.org.es/
Último acceso: 07/07/2005

[43] http://www.llnl.gov/linux/thunder/
Último acceso: 07/07/2005

[44] http://www.llnl.gov/asci/platforms/lanl_q/
Último acceso: 07/07/2005

CAPÍTULO 2. HERRAMIENTAS PARA INSTALACIÓN AUTOMÁTICA DE *CLUSTERS*

En este capítulo se realiza una breve descripción de los *clusters Beowulf*; se presentan dos herramientas que permiten instalar, configurar y administrar *clusters*: OSCAR y NPACI *Rocks*; también se describen los componentes que conforman las herramientas mencionadas, acompañados del proceso de su instalación. Además se describen brevemente otras herramientas existentes.

2.1. *CLUSTERS BEOWULF*

"Existen tantas definiciones de *Beowulf* así como personas que han construido o usado supercomputadores *Beowulf*"[1]. Algunos autores afirman que se debe llamar *Beowulf* sólo aquellos computadores construidos de la misma forma que el computador original de la NASA. Otros autores, un tanto extremistas, llaman *Beowulf* a cualquier sistema de computadores que ejecute código paralelo. Sin embargo, hay autores que coinciden con la siguiente definición: "*Beowulf* es una arquitectura conformada por múltiples computadores que puede usarse para computación paralela".

En 1994 bajo el patrocinio del Proyecto de la Tierra y Ciencias del Espacio (ESS – *Earth and Space Sciences*) del Centro de Excelencia en Datos del Espacio y Ciencias de la Información (CESDIS - *Center of Excellence in Space Data and Information Sciences*), Thomas Sterling y Don Becker crearon el primer *cluster Beowulf* con fines de investigación. Thomas Sterling nombró *Beowulf* al proyecto en honor al héroe de un cuento inglés, quien liberó a "*Danes of Heorot*" del monstruo opresivo *Grendel*. A manera de metáfora, se nombró *Beowulf* a esta nueva estrategia en computación de alto rendimiento

que hace uso de tecnología bien difundida en el mercado, para derrotar a los costos opresores en tiempo y dinero de la supercomputación.

Un sistema *Beowulf* usualmente consiste de un nodo servidor (maestro) y uno o más nodos clientes (esclavos), interconectados a través de una red Ethernet u otro tipo de red. Un *cluster Beowulf* se construye usando componentes de hardware y de software "bien conocidos" (*commodities*).

En la mayoría de los casos, los *clusters Beowulf* están compuestos por nodos clientes que no disponen de monitores ni teclados; se accede a cada nodo mediante una conexión remota o a través del puerto serial. El nodo servidor controla a todo el *cluster* y presta servicios de sistemas de archivos a los nodos clientes; también puede ser la consola de configuración del *cluster* y la puerta de salida (*gateway*) al mundo exterior. El nodo servidor controla y configura a los nodos clientes, e indica las tareas que éstos deben ejecutar. En una configuración con nodos clientes sin disco, éstos ni siquiera conocen su dirección IP hasta que el nodo servidor les asigna una. Los grandes *clusters Beowulf* pueden tener más de un nodo servidor y otros nodos dedicados a diversas tareas específicas; como por ejemplo, consolas de supervisión. En la Figura 2-1 se muestra la arquitectura genérica de un *cluster Beowulf.*

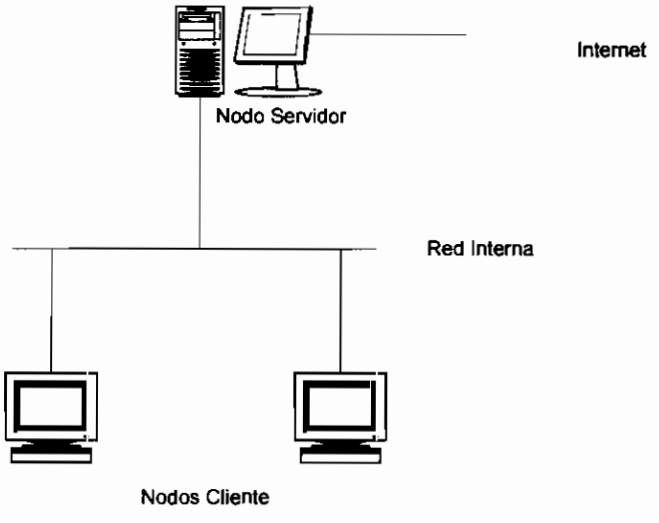


Figura 2-1. Arquitectura genérica de un *cluster Beowulf*

2.1.1. CLASIFICACIÓN

Para establecer las diferencias entre los distintos tipos de sistemas *Beowulf,* se presenta la siguiente clasificación:

- Clase I. Son sistemas compuestos por computadores cuyos componentes cumplen con la prueba de certificación "*Computer Shopper*", lo que significa que sus elementos son de uso común, y pueden ser adquiridos muy fácilmente en cualquier tienda distribuidora.

- Clase II. Son sistemas compuestos por computadores cuyos componentes no pasan la prueba de certificación "*Computer Shopper*", lo que significa que sus componentes no son de uso común y por tanto no pueden encontrarse con la misma facilidad que los componentes de sistemas de la clase anterior. Los equipos ubicados en esta categoría pueden presentar un nivel de prestaciones superior al de la Clase I.

Por lo general, los *clusters Beowulf* son *clusters* basados en hardware bien conocido, con una infraestructura de software de código abierto (Linux). Los nodos que conforman el *cluster* se encuentran dedicados únicamente a tareas del *cluster.* Los programas desarrollados para *clusters Beowulf* usualmente están escritos en C o en Fortran, utilizando librerías de paso de mensajes para realizar operaciones paralelas.

Existen algunas herramientas que permiten instalar, configurar y administrar *clusters Beowulf* de forma automática, como es el caso de OSCAR y NPACI *Rocks.* Las dos herramientas incluyen las mejores prácticas de computación *cluster* tipo *Beowulf* de alto rendimiento. OSCAR permite implementar *clusters Beowulf* Clase I; mientras que NPACI *Rocks* permite implementar *clusters Beowulf* Clase I o Clase II.

2.2. OSCAR

OSCAR es una colección de software de código abierto para crear un *cluster* sobre Linux desarrollada por el Grupo de *Clusters* Abiertos (OCG – *Open Cluster Group*).

El OCG se formó en abril del 2000, es un grupo formado por personas dedicadas a realizar prácticas de computación *cluster* de alto rendimiento y que últimamente desarrollan *clustering* en general (alta disponibilidad, sin discos duros (*diskless*)). El grupo de trabajo OSCAR está conformado por miembros de la industria y de investigación académica. Entre los miembros que pertenecen al grupo de trabajo OSCAR se encuentran: *Bald Guy Software* (BGS), Dell, IBM, Intel, MSC.Software, la Universidad de Indiana, el Centro Nacional para Aplicaciones de Supercomputadores (NCSA – *National Center for Supercomputing Applications*), el Laboratorio Nacional Oak Ridge (ORNL – *Oak Ridge National Laboratory*) y la Universidad Sherbrooke de Québec, Canadá.

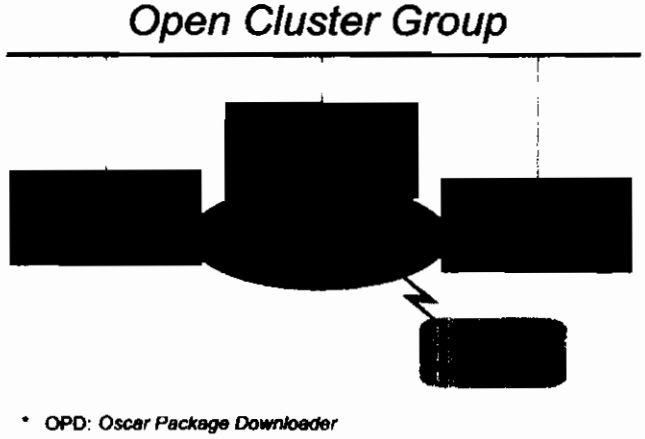
OSCAR se instala sobre un computador con una distribución Linux previamente instalada. La lista de las distribuciones que OSCAR soporta se presenta en la Sección 2.2.4. OSCAR realiza la instalación y/o configuración de los paquetes necesarios para el *cluster* de forma automática.

El objetivo primario del grupo de trabajo OSCAR es conseguir que la instalación, la configuración y la administración de un *cluster* de tamaño modesto, sea fácil. Cualquier cosa necesaria para un *cluster* – instalación y configuración, mantenimiento, programación [paralela], sistemas de encolamiento, programación de tareas – está incluida en OSCAR.

La primera versión de OSCAR (OSCAR v1.0) se liberó en abril de 2001. Desde su primera versión, OSCAR ha ido madurando hasta incluir funciones de instalación, mantenimiento y operación, dando como resultado uno de los paquetes de computación *cluster* ampliamente difundido. En la actualidad, OSCAR incluye dos

paradigmas adicionales: una solución de *clusters* sin discos llamada Thin-OSCAR [2] y funcionalidad de alta disponibilidad denominada HA-OSCAR [3].

La relación entre los grupos de trabajo del OCG y la infraestructura de OSCAR se muestra en la Figura 2-2.

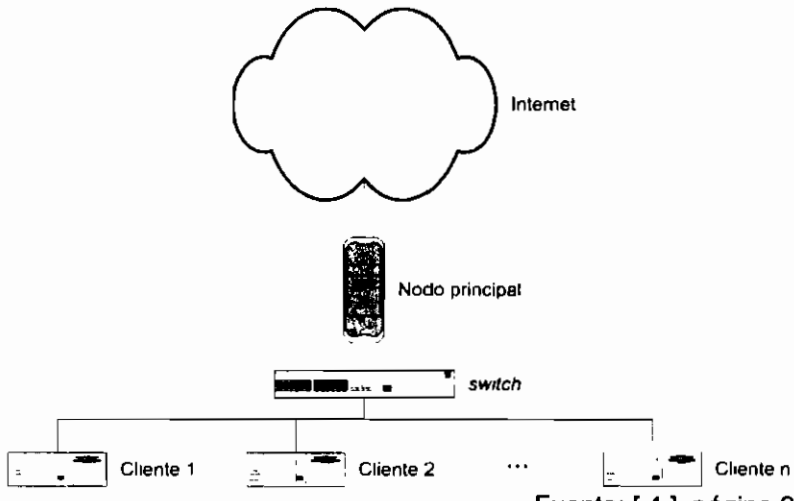


Fuente: [1], página 2

Figura 2-2. Relación entre los grupos de trabajo del OCG

2.2.1. ARQUITECTURA DE OSCAR

En la Figura 2-3 se muestra la arquitectura de OSCAR. Los computadores que conforman el *cluster* se denominan nodos, existen dos tipos de nodos: un nodo principal (nodo servidor) y varios nodos de cómputo (nodos clientes).



Fuente: [4], página 2

Figura 2-3. Arquitectura de OSCAR

El nodo principal provee respuestas a los pedidos de servicio y asigna las tareas apropiadas a cada nodo cliente. Los nodos clientes se dedican a realizar tareas de cómputo, poseen hardware homogéneo y disponen de una copia completa del sistema operativo y de otros programas. Los nodos se comunican a través de una red Ethernet que no está expuesta al mundo exterior; el nodo servidor posee dos interfaces de red: una interfaz de red conectada al mundo externo y una interfaz conectada a la red interna.

2.2.2. DISEÑO DE OSCAR

OSCAR fue diseñado para proporcionar un sistema capaz de soportar frecuentes modificaciones, permitiendo que la actualización de los paquetes que incluye sea fácil de realizar; además para realizar una nueva versión de OSCAR, se toma todo lo mejor que la versión anterior tiene y se realizan las modificaciones necesarias.

OSCAR es flexible, permite descargar e instalar paquetes para proveer mayor funcionalidad al *cluster* mediante el Gestor de Descargas de Paquetes OSCAR (OPD – *OSCAR Package Downloader*).

2.2.2.1. Gestor de descarga de Paquetes OSCAR (OPD)

OPD permite descargar e instalar software OSCAR desde contenedores de paquetes remotos. Un contenedor de paquetes puede ser un sitio Web o un sitio FTP. No existe un contenedor central; el cliente OPD descarga una lista[1] inicial de contenedores desde el sitio Web del grupo de trabajo OSCAR.

Debido a que los contenedores son sitios Web o FTP, se puede usar cualquier cliente tradicional FTP o explorador Web para obtener paquetes OSCAR. Sin embargo, los clientes OPD proveen mayor funcionalidad que los clientes tradicionales. OPD ofrece dos interfaces: una interfaz gráfica, excelente para uso

[1] El grupo de trabajo de OSCAR mantiene esta lista. Se puede realizar un pedido a los encargados del mantenimiento de la lista para agregar un nuevo sitio contenedor.

interactivo, y una interfaz de línea de comandos, excelente para usarlo mediante herramientas de alto nivel (o mediante *scripts* automatizados).

OPD provee las siguientes funcionalidades:

- Permite acceder de forma automática a una lista central de contenedores.
- Permite explorar los paquetes disponibles en cada contenedor.
- Provee información detallada acerca de cada paquete.
- Permite descargar, verificar y extraer los paquetes.

La interfaz gráfica se denomina **OPDer**, esta interfaz presenta un listado de paquetes con información específica de cada uno, además informa sobre su funcionalidad, posibles conflictos con versiones anteriores, paquetes adicionales que puede requerir e información acerca de la fuente que proporciona dicho paquete. En la Figura 2-4 se muestra **OPDer**.

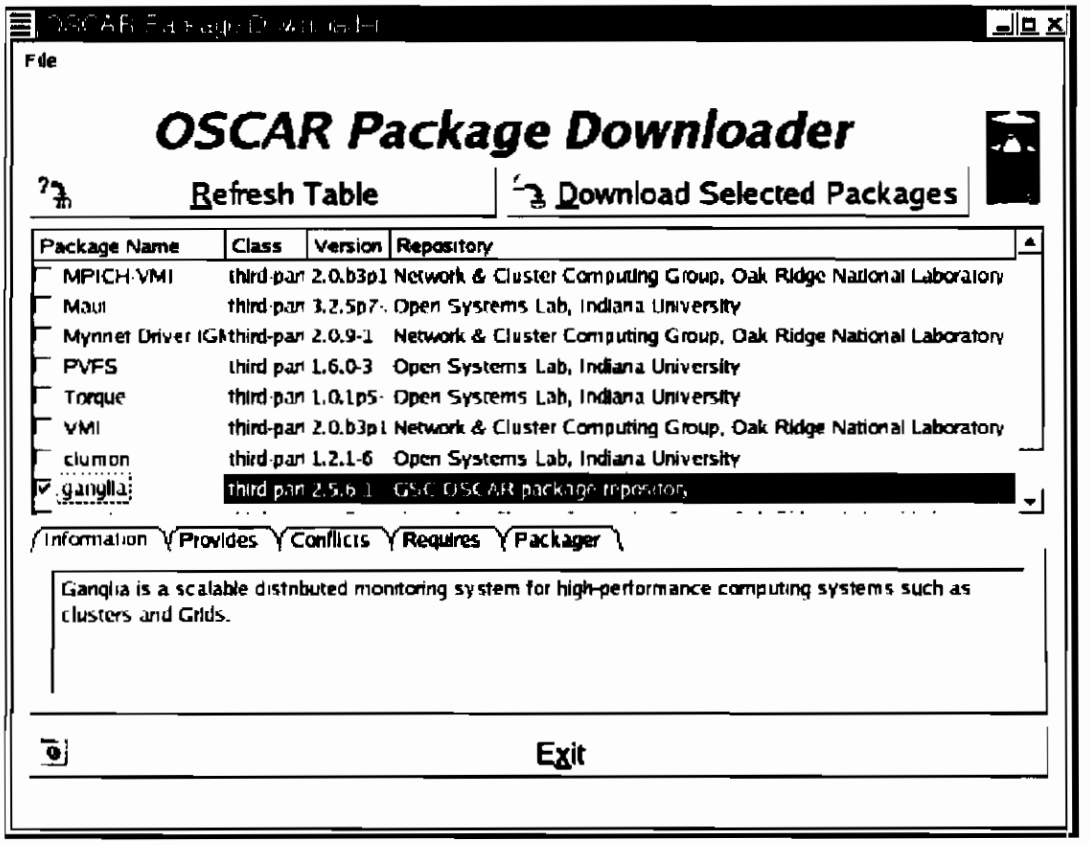


Figura 2-4. Interfaz gráfica del Gestor de Descargas OSCAR - OPDer

2.2.2.2. Empaquetamiento Modular

El empaquetamiento modular permite instalar y configurar software liberado con un formato prescrito. Los paquetes OSCAR se basan en el Sistema de Administración de Paquetes de Red Hat (RPM[2] – *Red Hat Package Manager*). Los paquetes OSCAR son software en formato RPM con meta archivos XML (*eXtended Markup Language*) que los describen y definen criterios para su instalación. Esta modularidad permite que la adición de nuevo software sea sencilla y fácil.

En la Tabla 2-1 se muestra el contenido y la estructura de los directorios de un paquete de OSCAR.

Archivo o Directorio	Descripción
config.xml	Meta archivo que contiene la descripción del paquete.
RPMS/	Directorio que contiene los binarios del(os) RPM(s) para el paquete.
SRPMS/	Directorio que contiene los archivos fuente del(os) RPM(s) para construir el paquete.
scripts/	Conjunto de *scripts* que se ejecutan durante la instalación y/o configuración del *cluster*.
testing/	*Scripts* de prueba.
doc/	Documentación y/o información de licencia del paquete.

Tabla 2-1. Directorios y archivos de un paquete OSCAR

2.2.3. COMPONENTES DE OSCAR

OSCAR contiene una serie de paquetes que se instalan en la mayoría de *Clusters* de Computación de Alto Rendimiento (HPCC – *High Performance Computing Cluster*). El "núcleo" de la infraestructura OSCAR está conformado por un pequeño conjunto de paquetes y se utiliza para instalar y configurar el *cluster*.

[2] Debido a que otras distribuciones de Linux hacen uso de los RPMs, en la actualidad el nombre oficial de RPM es *RPM Package Manager*, un acrónimo SRA (*self-referencing acronym*).

Los componentes de OSCAR se pueden dividir de acuerdo a la funcionalidad que proveen en el *cluster*. Se pueden mencionar los siguientes componentes:

- Núcleo de la Infraestructura/Administración.
- Administración/Configuración.
- Herramientas y servicios HPC.
- Seguridad.

2.2.3.1. Núcleo de la Infraestructura y Administración

El conjunto núcleo está conformado por los siguientes paquetes:

- Serie de programas para la Instalación del Sistema (SIS – *System Installation Suite*).
- Base de datos OSCAR (ODA - *OSCAR DAtabase*).
- Control y Comando del *Cluster* (C3 - *Cluster, Command and Control).*
- Administrador de Ambientes[3] (*switcher*).

Los componentes del núcleo permiten que el usuario construya una imagen[4] virtual del equipo destino usando SIS. La información del *cluster* se almacena en ODA. Se dispone de un *shell*[5] distribuido conocido como C3 y un administrador de ambientes denominado *switcher*.

2.2.3.1.1. Serie de programas para la Instalación del Sistema (SIS)

OSCAR utiliza SIS para realizar la instalación inicial de los nodos de cómputo. SIS se basa en la herramienta **SystemImager**. **SystemImager** se utiliza para construir una imagen virtual; ésta imagen se define como un árbol de directorios que comprende un sistema completo de archivos de un computador; la imagen se utiliza para instalar los nodos de cómputo del *cluster*. SIS incluye dos componentes adicionales: **System Installer** y **System Configurator**. Estos dos

[3] En Linux, el Ambiente se refiere al conjunto de información (variables de ambiente, alias) que el usuario utiliza para personalizar su sesión.

[4] En OSCAR, el término *imagen* se define como un árbol de directorios que comprenden un sistema de archivos completo de un computador.

[5] Un *shell* es un intérprete de comandos. Sirve de interfaz entre el usuario y el sistema.

componentes amplían la funcionalidad de **SystemImager**. **System Installer** permite construir la imagen en el nodo principal, mediante una descripción del objetivo. Esta imagen tiene ciertos aspectos generalizados que luego se adecuan al momento (*customize on-the-fly*) de la instalación mediante **System Configurator**. Esta fase de configuración dinámica permite que la imagen sea lo más general posible, por lo que aspectos como la interfaz de red no se encuentran definidos en la imagen SIS. Esta funcionalidad permite cierta heterogeneidad dentro de los nodos del *cluster*.

SIS se utiliza para realizar tareas iniciales (*bootstrap*) en la instalación de los nodos, es decir: inicio del *kernel*[6], particionamiento del disco, formateo del sistema de archivos e instalación del sistema operativo base. La imagen que se utiliza durante la instalación puede usarse para realizar mantenimiento de los nodos del *cluster*. Este método puede usarse para instalar y administrar un *cluster* completo. Modificar la imagen es tan fácil como modificar un sistema de archivos locales. Si se actualiza la imagen, se utiliza el comando **rsync**[7] para actualizar el sistema de archivos local en cada nodo del *cluster*.

2.2.3.1.2. Base de datos OSCAR (ODA)

El asistente de instalación OSCAR y los paquetes OSCAR utilizan la Base de Datos OSCAR para almacenar y recuperar datos.

Cada paquete de OSCAR tiene un meta archivo XML denominado `config.xml` que se usa para ingresar su información en la base de datos. Este archivo XML contiene una descripción del paquete y el nombre del RPM asociado a este paquete. Esto permite que la búsqueda de información de un paquete sea rápida y eficiente.

---

[6] El *kernel* es el núcleo de todo sistema operativo. Se carga al iniciar un sistema y permanece en memoria durante todo el tiempo en que el sistema permanece encendido. Administra los dispositivos como memoria y disco duro.

[7] **rsync** es una utilidad de Linux que provee transferencia de archivos de forma rápida. Sólo envía los bytes que han sido modificados no el archivo completo, por lo que es utilizado para sincronizar datos entre equipos.

ODA provee un sitio central de información del *cluster*. ODA dispone de siete tablas en una base de datos de MySQL denominada **oscar**, la cual se encuentra en el nodo principal.

2.2.3.1.3. Control y Comando del Cluster (C3)

La naturaleza distribuida del *cluster* introduce la necesidad de ejecutar comandos e intercambiar archivos a través del *cluster*. La herramienta C3 ofrece una interfaz de línea de comandos para administrar el *cluster*. Este conjunto fue desarrollado por ORNL.

C3 ofrece un conjunto de comandos para realizar operaciones de recolección (*gather*) y distribución (*scatter*) de archivos. Esta herramienta es útil tanto en el ámbito administrativo como en el ámbito de usuario.

El conjunto de herramientas OSCAR utiliza C3 internamente para distribuir archivos y realizar operaciones paralelas sobre el *cluster*. C3 permite la ejecución en paralelo de los comandos estándar de Linux, por lo que los administradores pueden usar estas herramientas para realizar operaciones de mantenimiento del *cluster*.

2.2.3.1.4. Switcher

Un problema común de la administración de un sistema es el administrar el ambiente de usuario. Típicamente, la administración del ambiente se realiza modificando de forma manual archivos de tipo "*dot*"[8], como por ejemplo: `.bashrc`, `.cshrc`, `.profile`. Muchos administradores de *clusters* proveen ciertos *scprits* que crean y/o aumentan las variables de ambiente del `$PATH`[9], de

[8] En Linux, los archivos de configuración del ambiente de usuario son archivos simples de texto, que se encuentran ocultos por lo cual su nombre empieza con un punto.

[9] Variable del sistema Linux que contiene la ubicación de los ejecutables.

$LIBRARY_PATH[10] y de $MANPAGE[11]. Por lo general, estas dos soluciones conllevan errores humanos, algunas veces con resultados desastrosos, como por ejemplo imposibilidad de ingresar al sistema debido a errores en los archivos "*dot*".

OSCAR provee un paquete denominado *switcher*, que provee un mecanismo simple para manipular ambientes tanto a nivel de sistema como a nivel de usuario. Los cambios en el ambiente se realizan a través de la línea de comandos y no requieren que el usuario edite de forma manual ningún archivo. *switcher* contiene dos herramientas: **env-switcher** y **modules**.

El paquete **modules** contiene un conjunto de *scripts* que proveen un método eficiente de manipulación de ambiente. Se proveen primitivas básicas para acciones, por ejemplo: adición de un directorio a las variables de ambiente del PATH, despliegue de información básica sobre un paquete, y configuración de variables arbitrarias de ambiente.

switcher instala y configura el paquete **modules**. Además, crea dos tipos de módulos: aquellos que se cargan de manera incondicional y aquellos que son sujetos a niveles de usuario o de sistema.

Algunos paquetes de OSCAR usan módulos no condicionales para incluir su información en el PATH, configurar ciertas variables arbitrarias de entorno, etc. Esto permite que los usuarios del *cluster* tengan automáticamente estas configuraciones en sus ambientes.

Otros módulos son opcionales, es decir, proveen una metodología de selección de uno entre varios paquetes equivalentes. Esto permite que el sistema provea un conjunto de aplicaciones por defecto, que opcionalmente pueden ser cambiados

[10] Variable del sistema Linux que contiene la ubicación de las librerías necesarias para que los programas funcionen correctamente.

[11] Variable del sistema Linux que contiene la ubicación de las páginas de ayuda utilizadas por el programa **man**.

por el usuario. OSCAR instala dos implementaciones de MPI: LAM/MPI y MPICH. Algunos usuarios prefieren una de estas implementaciones en particular. Otros usuarios prefieren el uso de ambas implementaciones, por lo que constantemente realizan cambios entre dichas implementaciones.

env-switcher proporciona un ambiente persistente basado en el ambiente configurado por **modules**. **env-switcher** carga el conjunto de módulos específicos cada vez que se necesita invocar al *shell*. **env-switcher** permite al usuario del *cluster* el manipular su ambiente a través de la línea de comandos. Además, permite que los cambios realizados sean persistentes; los cambios realizados se mantienen incluso en *shells* remotos, como por ejemplo **rsh** o **ssh**.

2.2.3.2. Administración y Configuración

OSCAR provee varias herramientas que asisten en la administración del *cluster*, incluyendo las herramientas ya mencionadas SIS y C3. Además, OSCAR ayuda en la configuración de servicios que el *cluster* requiere como el Sistema de Archivos de Red (NFS – *Network File System*) y el Protocolo de Tiempo de Red (NTP – *Network Time Protocol*). NFS permite compartir el sistema de archivos de los usuarios (`/home`) a través del *cluster*; esto permite que los usuarios puedan acceder fácilmente a sus archivos desde cualquier nodo del *cluster*. El servicio NTP permite tener el tiempo consistente a lo largo del *cluster* y puede ser sincronizado con un reloj atómico del Internet.

2.2.3.2.1. OPIUM - OSCAR Password Installer and User Management

OPIUM se encarga de sincronizar los archivos de las cuentas de usuarios por todo el *cluster*. Los comandos para agregar y eliminar usuarios están inmersos en **OPIUM** para ser utilizados en el *cluster*.

2.2.3.2.2. Kernel-Picker

Kernel-Picker permite escoger el *kernel* de la imagen SIS antes de instalar los nodos.

2.2.3.2.3. Loghost

El paquete **Loghost** envía información de los registros de eventos (*logs*) del sistema (syslog[12]) desde los nodos de cómputo al nodo principal. La centralización es muy útil cuando se trata de dar mantenimiento al *cluster*.

2.2.3.3. Herramientas y Servicios HPC

2.2.3.3.1. Sistema de Monitoreo Ganglia

Ganglia [5] ofrece un sistema de monitoreo escalable mediante línea de comandos o mediante una herramienta de acceso basada en GUI. OSCAR configura el sistema base con un demonio monitor en cada nodo y un servidor en el nodo principal que recopila información. El sistema se comunica usando un canal *multicast* y utiliza XML como el lenguaje de formato de datos. El flujo de datos contiene información de métricas (uso de CPU, tiempo de funcionamiento), así como información de identificación (versión del sistema operativo). Se puede ampliar las métricas si se desea obtener otro tipo de información. Ganglia se describe con más detalle en la Sección 4.2.

2.2.3.3.2. Formato de Datos Jerárquicos ver. 5 (HDF5 – Hierarchical Data Format version 5)

HDF5 [6] es una librería de propósito general que define un formato de datos de tipo científico. Estos datos pueden agruparse dentro de estructuras jerárquicas definidas por el usuario. La librería fue desarrollada para usuarios que trabajan

[12] **syslog** es el demonio (syslogd) encargado de recoger todos los *logs* que otros demonios generan dentro de un ambiente Linux y procesarlos de acuerdo a criterios definidos en cada sistema.

con archivos de gran tamaño, teóricamente en rangos que incluyen o superan los terabytes.

2.2.3.3.3. Librerías Paralelas

Las librerías de paso de mensajes de PVM y las de MPI (MPICH y LAM/MPI), proveen la funcionalidad requerida por el ambiente OSCAR que permite tomar ventaja de los ambientes de computación distribuida para la creación de programas paralelos distribuidos. La distribución de OSCAR dispone de las versiones más recientes de estos paquetes y provee la configuración necesaria para las librerías mencionadas. OSCAR no realiza ninguna modificación de estos paquetes.

En la Sección 3.1 se describen las librerías de paso de mensajes PVM y MPI.

2.2.3.3.4. OpenPBS / MAUI

El Sistema Portable por Lotes (OpenPBS – *Portable Batch System*) [7] define un sistema flexible de administración de carga y planificación de tareas. Este sistema opera sobre ambientes UNIX multiplataforma, incluyendo *clusters* heterogéneos de estaciones de trabajo, supercomputadores y sistemas masivamente paralelos. El paquete OSCAR incluye varias mejoras muy útiles en OpenPBS, como soporte para fallos de nodos, soporte para más de 500 nodos, entre otros.

El planificador de tareas que se utiliza en OpenPBS es del tipo Primero en Entrar, Primero en Salir (FIFO[13] – *First In First Out*). Sin embargo, OSCAR incluye un planificador de tareas más avanzado. Este planificador de tareas se denomina Maui [8]. Maui ofrece numerosas características avanzadas, muchas de las cuales van más allá de la configuración por defecto de OSCAR. Maui dispone de varias políticas de programación de tareas como priorización dinámica, reserva de nodos y compartición justa (*fairshare*).

[13] Se conoce a FIFO como una cola de entrada y salida de datos hacia adelante, es decir los datos que entran primero son los datos que salen primero.

Algunas características adicionales de PBS se presentan en la Sección 4.4 y en la Sección 4.3.3 se describe al planificador Maui.

2.2.3.4. Seguridad

Los *clusters* están sujetos a ataques, ataques que van en aumento. El proveer seguridad es especialmente importante para investigaciones privadas. La mayoría de herramientas de OSCAR no fueron concebidas teniendo en cuenta cuestiones de seguridad. El uso de una red privada limita el acceso desde el mundo exterior al *cluster*; el uso de encripción para comunicaciones internas y externas, así como el uso de *firewalls* locales tanto en el nodo principal como en los nodos de cómputo, incrementan la seguridad del *cluster*. OSCAR utiliza **ssh** y configura los paquetes del núcleo para que usen **ssh** para la comunicación cuando sea posible, proveyendo conexiones seguras al *cluster* desde el mundo exterior. Cada nodo dispone de un *firewall* con filtrado de paquetes provisto por **Pfilter**. Los *firewalls* se configuran para permitir comunicaciones sin restricciones entre los nodos de cómputo y el nodo principal, para prevenir todo excepto unos cuantos métodos autorizados de acceso externo hacia el *cluster*, y para permitir acceso sin restricciones desde el *cluster* hacia los recursos del mundo externo.

2.2.3.4.1. OpenSSH

Típicamente se accede al *cluster* desde localidades remotas. El *Shell* Seguro (SSH – *Secure SHell*) permite a los usuarios conectarse de forma segura al *cluster*. OSCAR utiliza la implementación de código abierto de **ssh**: OpenSSH. Los esquemas de autenticación y encripción incorporados pueden modificarse para satisfacer necesidades más exigentes o menores.

2.2.3.4.2. Pfilter

Pfilter provee un *firewall* con filtrado de paquetes. **Pfilter** se instala en cada nodo como un servicio del sistema que puede ser inicializado, detenido y reinicializado. **Pfilter** puede manipular múltiples interfaces de red, permitiendo a una máquina el

reenvío de paquetes para los nodos que se encuentra en la red privada, permitiendo proteger los nodos de accesos desde la red externa de una manera segura.

2.2.4. INSTALACIÓN DE OSCAR

Para hacer uso de OSCAR, se requiere una instalación de tipo "*workstation*" de alguna de las distribuciones de Linux soportada por OSCAR. La lista de distribuciones de Linux que OSCAR soporta se muestra en la Tabla 2-2.

Distribución	Versión	Fecha	Estado
Red Hat 8.0	OSCAR 3.0	11 - 2003	Totalmente soportado
Red Hat 9.0	OSCAR 3.0	11 - 2003	Totalmente soportado
	OSCAR 4.0	12 - 2004	Totalmente soportado
	OSCAR 4.1	04 - 2005	Totalmente soportado
Mandrake Linux 9.0	OSCAR 3.0	11 - 2003	Totalmente soportado
Mandrake Linux[14] 10.0	OSCAR 4.1	04 - 2005	Totalmente soportado
Fedora Core 2	OSCAR 4.0	12 - 2004	Totalmente soportado
	OSCAR 4.1	04 - 2005	Totalmente soportado
Fedora Core 3	OSCAR 4.1	06 - 2005	Experimental
Red Hat Linux Enterprise 3 (x86)	OSCAR 4.0	12 - 2004	Totalmente soportado
	OSCAR 4.1	04 - 2005	Totalmente soportado
Red Hat Linux Enterprise 3 (IA64)	OSCAR 4.0	12 - 2004	Totalmente soportado
	OSCAR 4.1	04 - 2005	Totalmente soportado

Tabla 2-2. Distribuciones que OSCAR soporta

Los RPMs de la distribución deben copiarse al directorio `/tftpboot/rpm` para ser usados durante la fase de construcción de la imagen de los nodos. El usuario puede descargar, extraer e iniciar el asistente de instalación de OSCAR.

[14] Mandriva Linux es el nuevo nombre de la distribución Mandrake Linux.

La instalación se realiza a través de una interfaz de usuario. La interfaz gráfica del asistente de instalación de OSCAR se muestra en la Figura 2-5. El asistente dispone de pasos que el usuario debe seguir en secuencia para obtener un nodo principal y un conjunto de nodos de cómputo. Los servicios que el nodo principal necesita se configuran durante este proceso, así como el número de nodos y el software que se instalará en el *cluster*.

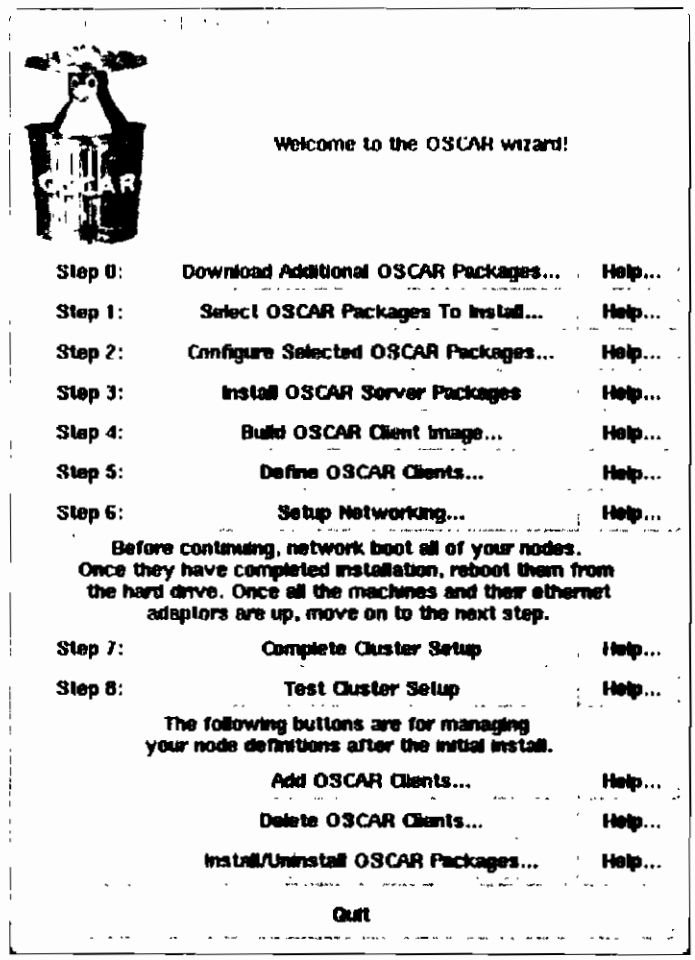


Figura 2-5. Interfaz gráfica del Asistente de OSCAR

El corazón de la instalación de OSCAR es la construcción de una "imagen", un nodo lógico que contiene el sistema operativo básico e incluye los paquetes de software seleccionados. Los nodos de cómputo se definen y asocian con la imagen. Cada nodo arranca por red (mediante un disquete o de PXE[15]), intercambia información de identificación (dirección MAC), es formateado de

[15] PXE (*Pre eXecution Environment*) es un programa que permite iniciar un computador desde la red, está disponible en el ROM de algunas tarjetas de red.

forma remota y se le instala el software basándose en la imagen. OSCAR administra todo este proceso de instalación y reduce significativamente la experiencia y el tiempo requerido para construir y configurar un *cluster*. En la Figura 2-6 se presentan los pasos del proceso de instalación del nodo maestro de OSCAR mediante el *Wizard* de instalación de OSCAR.

El proceso detallado de instalación y configuración de OSCAR se presenta en el Anexo A.

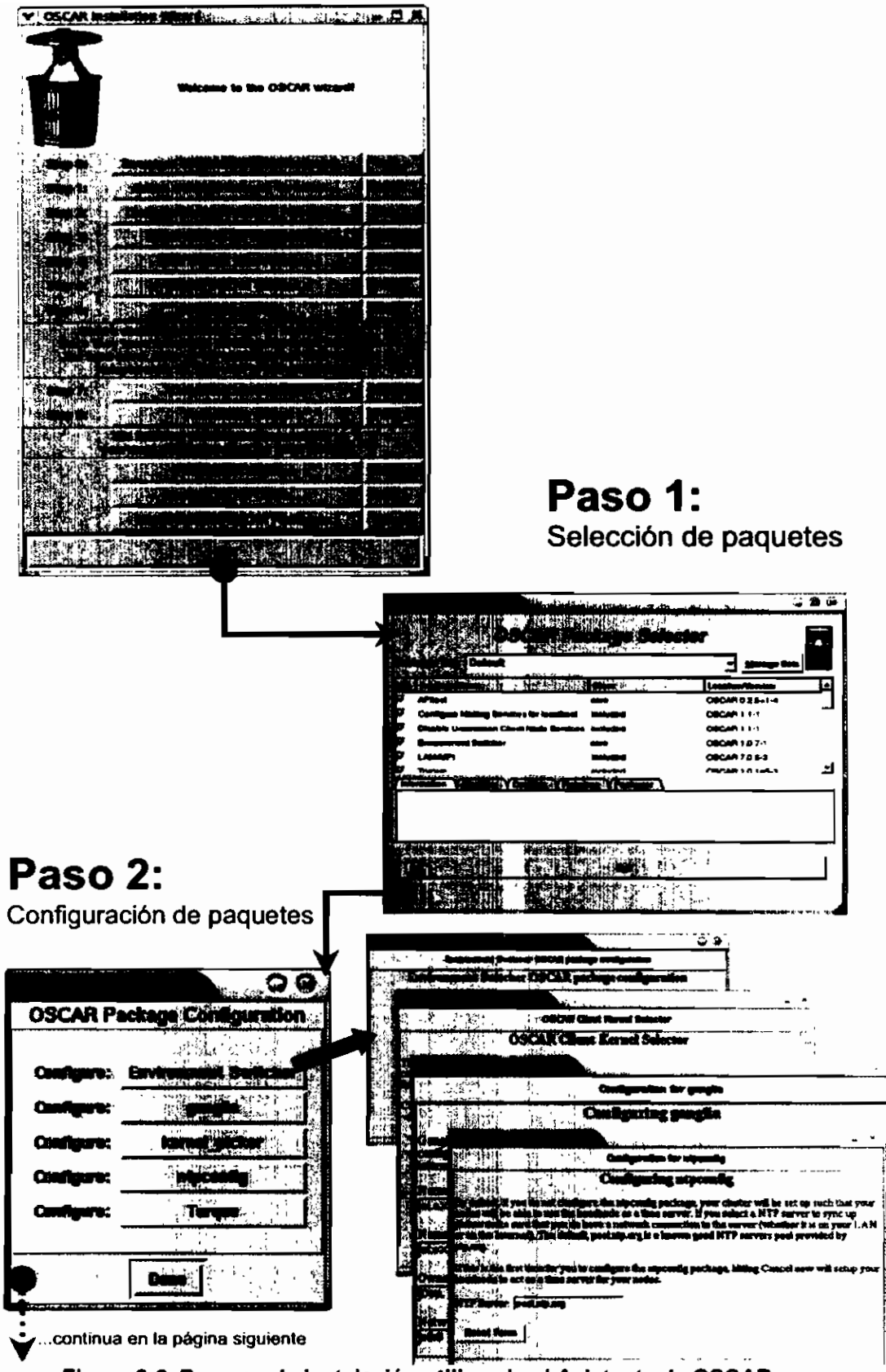


Figura 2-6. Proceso de instalación utilizando el Asistente de OSCAR

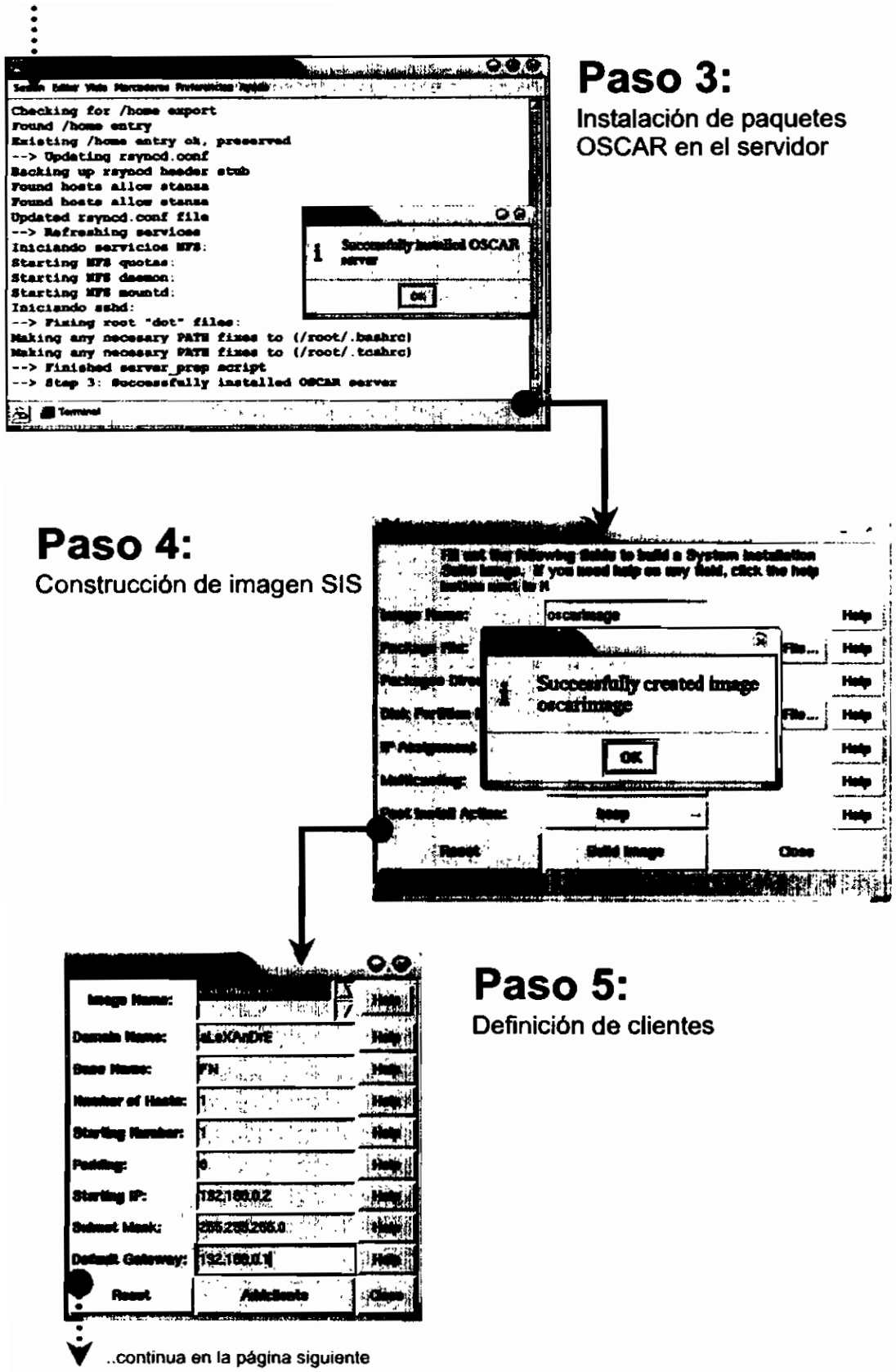


Figura 2-6 (continuación). Proceso de instalación utilizando el Asistente de OSCAR

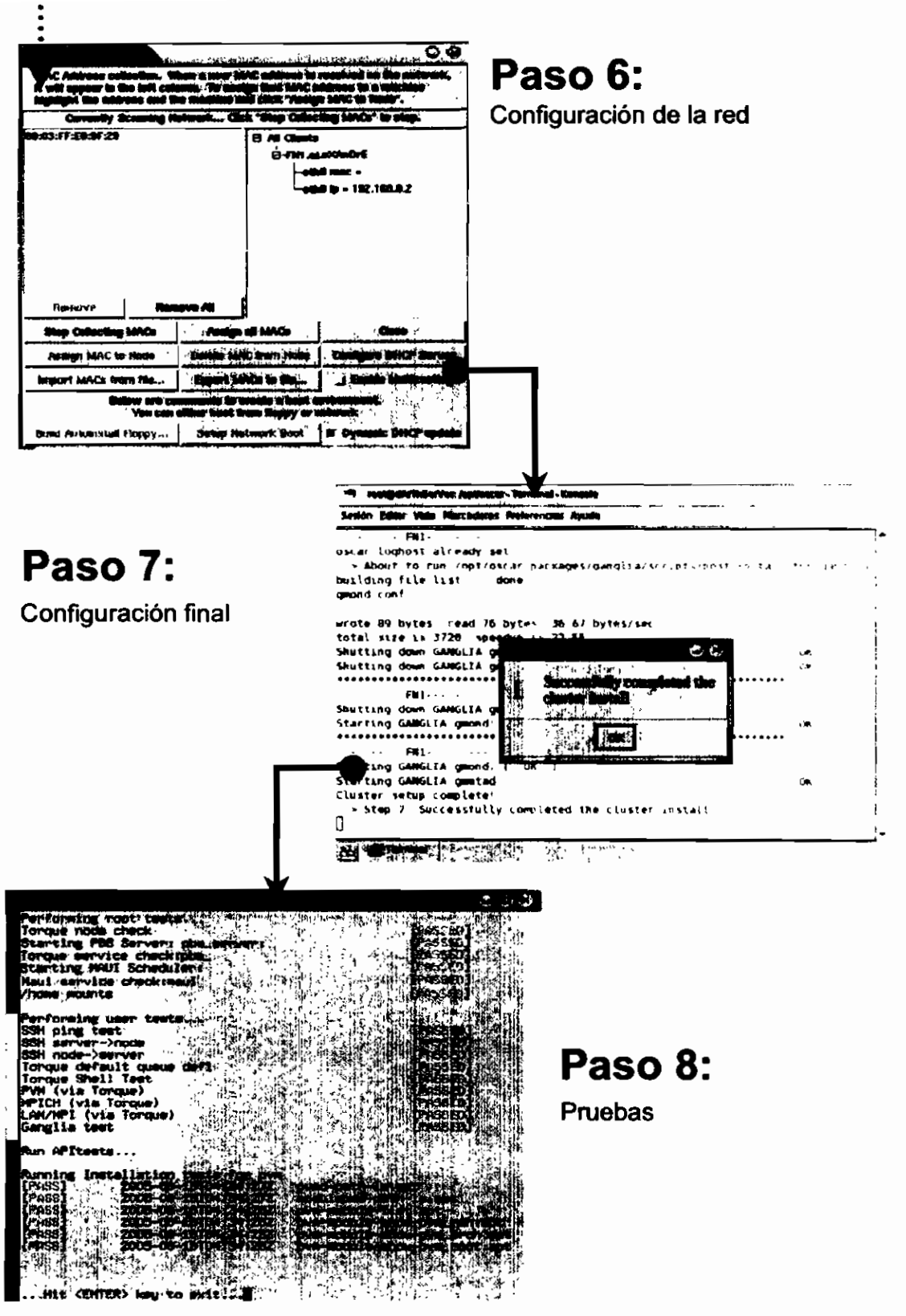


Figura 2-6 (continuación). Proceso de instalación utilizando el Asistente de OSCAR

2.2.5. Thin-OSCAR

El grupo de trabajo Thin-OSCAR fue creado en julio de 2001. Este grupo de trabajo, específicamente analiza y resuelve problemas en los que se requiere soporte sin disco de los *cluster* OSCAR. La implementación actual usa la imagen SIS y crea un disco de inicio que contiene un sistema Linux mínimo[16]. Los recursos del sistema (`/usr`), los binarios de OSCAR (`/opt`) y los directorios de usuarios (`/home`) se exportan mediante NFS.

Los objetivos a largo plazo del grupo de trabajo Thin-OSCAR son soportar nodos sin sistema (nodos sin sistema operativo en el disco), nodos sin disco y *clusters* heterogéneos que estén constituidos de nodos con disco y sistema.

Thin-OSCAR define tres clases de nodos: sin disco (*diskless*), sin sistema operativo (*systemless*) [se dispone de disco en el nodo pero sin ningún sistema operativo], discos completos (*diskfull*) [nodo regular de OSCAR]. Por el momento Thin-OSCAR solo soporta los nodos completos y sin disco.

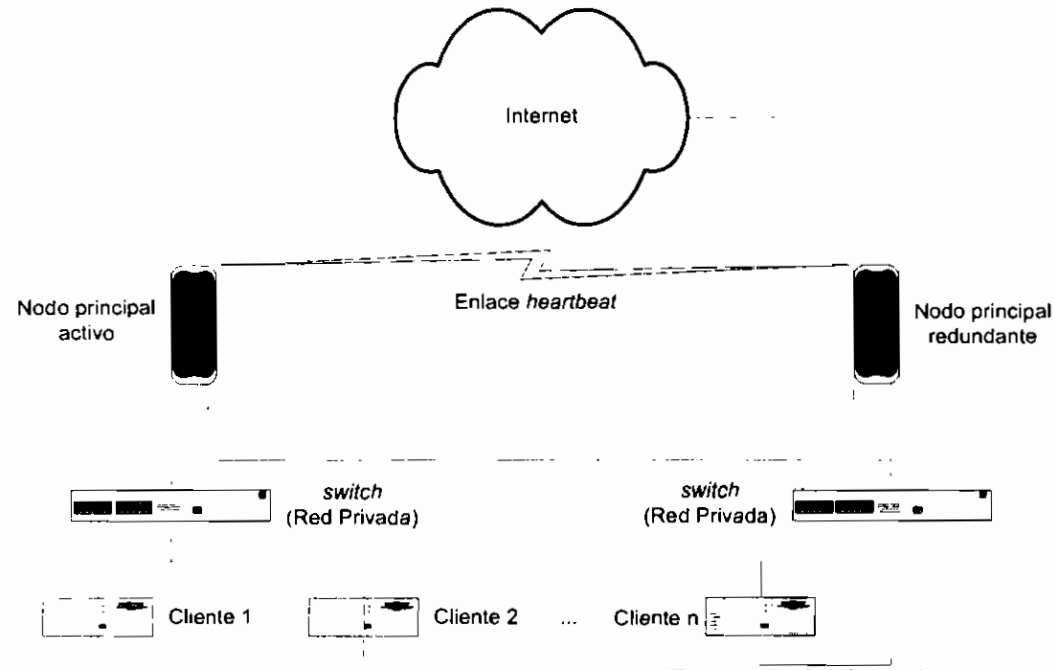
2.2.6. HA-OSCAR

El grupo de trabajo de Alta Disponibilidad de OSCAR (HA-OSCAR – *High Availability* OSCAR) se estableció en julio del 2002. El objetivo de este grupo es el desarrollar una solución completa, de código abierto y de costo efectivo para problemas de alta disponibilidad asociadas con computación *cluster* de alto rendimiento.

HA-OSCAR provee formas para eliminar puntos únicos de fallo. Algunas técnicas que proveen confiabilidad y disponibilidad de sistemas de computación son la duplicación de hardware y la redundancia de red. El *cluster* HA-OSCAR provee un duplicado del nodo principal. Para implementar esta arquitectura se presentan algunas configuraciones: *Active-Active*, *Active-Hot Standby* y *Active-Cold*

[16] El disco de inicio se trasfiere a través de la red o mediante un disquete y permite iniciar los nodos, los cuales no requieren tener instalado ningún sistema operativo (sin disco).

Standby. Actualmente solo se dispone de la configuración *Active-Hot Standby*. En la Figura 2-7 se muestra la arquitectura de un cluster HA-OSCAR.



Fuente: [4], página 3

Figura 2-7. Arquitectura de HA-OSCAR

En la configuración *Active-Active* se dispone de una mejor utilización de recursos, debido a que se dispone de dos nodos principales activos y proveyendo servicios. Los nodos maestros ejecutan servidores redundantes de PBS, Maui, DHCP, NTP, TFTP (*Trivial File Transfer Protocol*), NFS, **rsync** y SNMP. Si algo le llega a pasar a uno de los servidores, todas las funciones que éste proveía son recuperadas en el otro servidor y todos los pedidos continúan siendo servidos, a cambio de una pérdida en el rendimiento debido a la caída de uno de los servidores.

En la configuración *Active-Hot Standby* se dispone de un servidor maestro y de un servidor suplente. El servidor suplente es un "clon" del servidor maestro. El nodo maestro está activo y provee servicios, mientras que el servidor suplente monitorea la actividad (*heartbeat*) del nodo maestro buscando un posible fallo (de un servicio o del equipo), si descubre que alguna función está fallando, toma el control del *cluster* y empieza a dar servicio. El servidor suplente es un servidor de imágenes, el cual utiliza **SystemImager** para construir y restaurar imágenes del

sistema del servidor primario, así como para proveer respaldos para recuperarse ante fallos.

La configuración *Active-Cold Standby* es similar a la configuración *Active-Hot Standby,* con la diferencia de que el servidor suplente se activa desde un estado frío (sin conocimiento previo del estado del maestro).

HA-OSCAR también provee una red de alta disponibilidad mediante puertos Ethernet redundantes en cada máquina, además de funcionalidad de red duplicada (cables, *switches*). Adicionalmente el hecho de tener dos redes funcionando, permite un incremento en el rendimiento de las comunicaciones usando técnicas como *channel bonding*[17] o mensajes a través de caminos redundantes de comunicación.

En marzo de 2004, el grupo de Investigación de Computación Extrema (XCR – *eXtreme Computing Research*) de la Universidad Tech de Louisiana, liberó la versión 1.0 beta de HA-OSCAR. HA-OSCAR 1.0 incorpora mecanismos para detección y recuperación de fallos, y soporte para el modo *Active-Hot Standby* para los nodos principales. Esta versión se instala únicamente sobre Red Hat 9.0 con OSCAR 3.0 y dispone de un asistente para su instalación y una herramienta Web que permite crear y configurar *clusters Beowulf* de "múltiples cabezas" (múltiples nodos principales).

2.3. NPACI *Rocks*

NPACI (*National Partnership for Advanced Computational Infrastructure*) *Rocks* es una colección de software de código abierto para crear un *cluster* sobre Linux Red Hat, desarrollada por el Grupo de Computación *Cluster* del Centro de Supercomputación de San Diego, California.

[17] *Channel bonding* es una técnica que permite agregar múltiples interfaces de red dentro de un interfaz lógico para proveer balanceo de carga, por lo que los datos enviados pueden distribuirse entre las múltiples interfaces.

Rocks es desarrollado por grupos de investigación, varias compañías y científicos, así como desarrolladores e investigadores de varias partes del mundo. Entre los grupos de investigación se encuentran: el Centro de Supercomputadores de San Diego (SDSC – *San Diego Supercomputer Center*) de la Universidad San Diego de California, el Grupo Millennium de la Universidad Berkeley de California, la Compañía de Sistemas Escalables (SCS) en Singapur, el grupo de Ambientes de *Clusters* Escalables Abiertos (OpenSCE) de Tailandia. Entre las compañías que apoyan al grupo *Rocks*, se encuentran: Sun Microsystems, HP, Dell, AMD, Infinicom Systems, entre otros.

El objetivo principal de *Rocks* es permitir que la instalación de un *cluster* sea lo más fácil posible. Para cumplir con este objetivo, la instalación realiza una serie de suposiciones acerca del software que será incluido y cómo el *cluster* será configurado.

Rocks realiza una instalación completa del sistema operativo sobre un nodo. *Rocks* incorpora una versión de la distribución de Red Hat con software adicional específico para *clusters*. Adicionalmente, *Rocks* configura correctamente varios servicios. Al instalar *Rocks*, éste instalará Linux, por lo que no es posible agregar *Rocks* sobre un servidor existente o usarlo con alguna distribución de Linux diferente.

Rocks permite la creación de *clusters* más complejos que los creados con OSCAR. Tiene soporte para sistemas Itanium (tecnología Intel de 64 bits) y Opteron (tecnología AMD de 64 bits), así como sistemas basados en la tecnología Intel de 32 bits.

Una de las principales características de *Rocks* es un mecanismo robusto para producir distribuciones personalizadas (con parches de seguridad preaplicados tanto a nivel del *kernel* como a nivel de aplicación), el mismo que define un conjunto completo de software para un nodo en particular.

Un *cluster* puede requerir varios tipos de nodos: nodos de cómputo, nodos servidores de archivos y nodos de monitoreo. Cada uno de estos roles requiere un conjunto especializado de software. Dentro de una distribución, se pueden definir diferentes tipos de nodos basándose en un archivo *kickstart* de Red Hat.

Un archivo *kickstart* es una descripción basada en texto, de los paquetes de software y de la configuración que será desplegada en un nodo. El *Kickstart Rocks Graphs* es una estructura de árbol basada en XML utilizada para definir archivos *kickstart* de Red Hat. Mediante el uso de un grafo, *Rocks* puede definir de forma eficiente tipos de nodos, sin duplicar componentes compartidos. Los varios nodos de *Rocks* comparten varios paquetes del conjunto de software.

Mediante esta tecnología de instalación, se puede abstraer muchas de las diferencias del hardware y permitir que el proceso **Kickstart** auto detecte los módulos de hardware correctos a cargarse (por ejemplo, tipos de discos: SCSI, IDE; adaptadores integrados RAID; interfaces Ethernet; interfaces de red de alta velocidad).

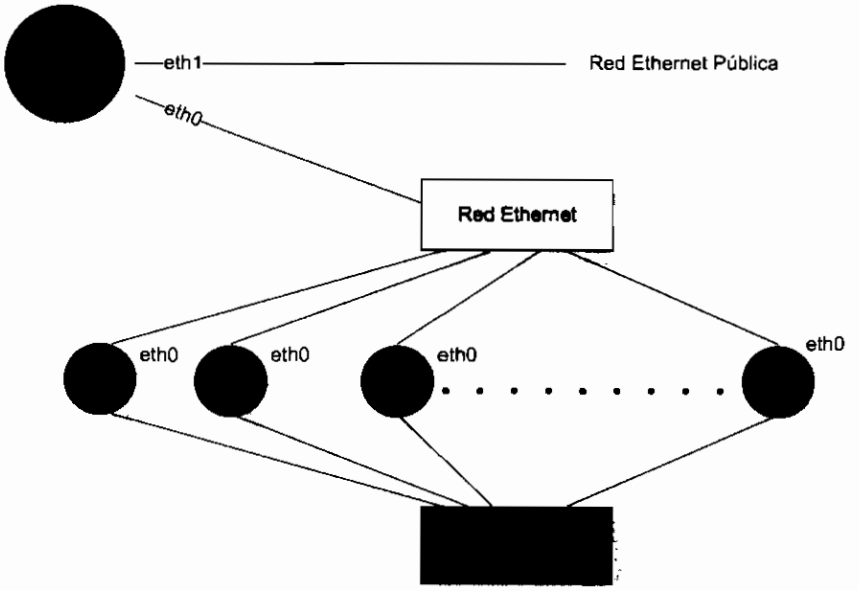
Rocks utiliza una base de datos de MySQL para almacenar las configuraciones globales y luego genera reportes para crear archivos de configuración de servicios específicos; como por ejemplo, archivos de configuración de DHCP, archivos de nodos PBS, archivos `/etc/hosts`[18].

Las instalaciones por defecto tienden a ser bastante rápidas. La estrategia de administración de *Rocks* permite arreglar los problemas de software sobre un nodo mediante la reinstalación del sistema sobre el mismo, en lugar de tratar de diagnosticar y resolver el problema. Dependiendo del hardware del nodo, es posible reinstalarlo en menos de 10 minutos [9].

[18] El archivo `/etc/hosts` contiene información de todos los equipos que conforman el *cluster Rocks*, motivo por el cual cada computador puede comunicarse utilizando el nombre del equipo en lugar de su dirección IP sin hacer uso de un servidor DNS.

2.3.1. ARQUITECTURA DE *Rocks*

En la Figura 2-8 se muestra la arquitectura tradicional de un sistema *Rocks*, la cual está compuesta por un nodo servidor (*frontend*), varios nodos cliente, una red Ethernet y una red opcional de alto rendimiento (Myrinet o Gigabit Ethernet).



Fuente: [9], página 19

Figura 2-8. Arquitectura de *Rocks*

2.3.1.1. Nodo *Frontend*

Los nodos *frontend* se instalan con software estándar de amplio uso, para soportar el desarrollo de aplicaciones *cluster* y la ejecución de aplicaciones paralelas. Este nodo se usa para instalar los nodos de cómputo.

Estos nodos están expuestos al mundo exterior. Sobre estos nodos se ejecutan algunos servicios como NFS, NIS, DHCP, NTP, MySQL, HTTP. En los nodos *frontend* los usuarios se conectan al sistema, presentan tareas, compilan código, etc. Estos nodos pueden actuar como enrutadores para otros nodos usando NAT (*Network Address Translation*).

Por lo general, los nodos *frontend* tienen las siguientes características:

- Dos interfaces de red: una pública y la otra privada.
- Discos de gran tamaño para almacenar archivos.

La interfaz pública está conectada a la red pública, mientras que la interfaz privada está conectada a la red del *cluster*. En *Rocks*, la primera interfaz (`eth0`[19]) se conecta a la red privada, y la segunda interfaz (`eth1`) a la red pública.

2.3.1.2. Nodos de Cómputo

Los nodos de cómputo son nodos esclavos. Son los que ejecutan todas las tareas. Las herramientas de *Rocks* permiten que un sistema operativo completo sea reinstalado en cada nodo de cómputo en una cantidad pequeña de tiempo (~10 minutos). Para la instalación de los nodos de cómputo, se puede usar el CD de *Rocks*, un disquete de arranque o hacerlo desde la red. Los nodos inician utilizando un kernel básico que contiene un conjunto de programas que les permite usar HTTP para obtener el sistema operativo y los paquetes del *cluster* desde el *frontend*. *Rocks* soporta hardware heterogéneo en los nodos de cómputo, gracias al uso de **Kickstart** y de **Anaconda**[20].

Los nodos de cómputo pueden disponer de las siguientes características:

- Conexión Ethernet para la administración.
- Discos de tamaño suficiente para contener tanto el sistema operativo como librerías.
- Conexión de alto rendimiento (opcional).

Los *clusters* sin disco duro no son una opción de *Rocks*. *Rocks* asume que se dispone de discos duros para todos los nodos. Los nodos de cómputo pueden configurarse para que arranquen sin teclado, con un teclado o conectados a un *switch* KVM (*Keyboard Video Mouse*).

---

[19] Nombre del interfaz Ethernet asignado por el sistema operativo UNIX. La primera interfaz se denomina `eth0`, la segunda `eth1`.

[20] Anaconda es el demonio encargado de la instalación y configuración de los diferentes dispositivos de un sistema Linux basándose en archivos *kickstart*.

2.3.1.3. Otros tipos de Nodos

Rocks permite instalar otro tipo de nodos de acuerdo a las necesidades que se tenga. Es posible instalar nodos para administración, nodos I/O[21] de PVFS, nodos con NAS (*Network Attached Storage*). Estos son nodos de cómputo pero que disponen de servicios adicionales. El tipo de nodo se escoge mediante **insert-ethers**[22]. En la Figura 2-9 se puede apreciar la interfaz de **insert-ethers**. Además es posible definir nuevos tipos de nodos, por ejemplo, se puede crear un nodo que tenga soporte X11 para aplicaciones de visualización.

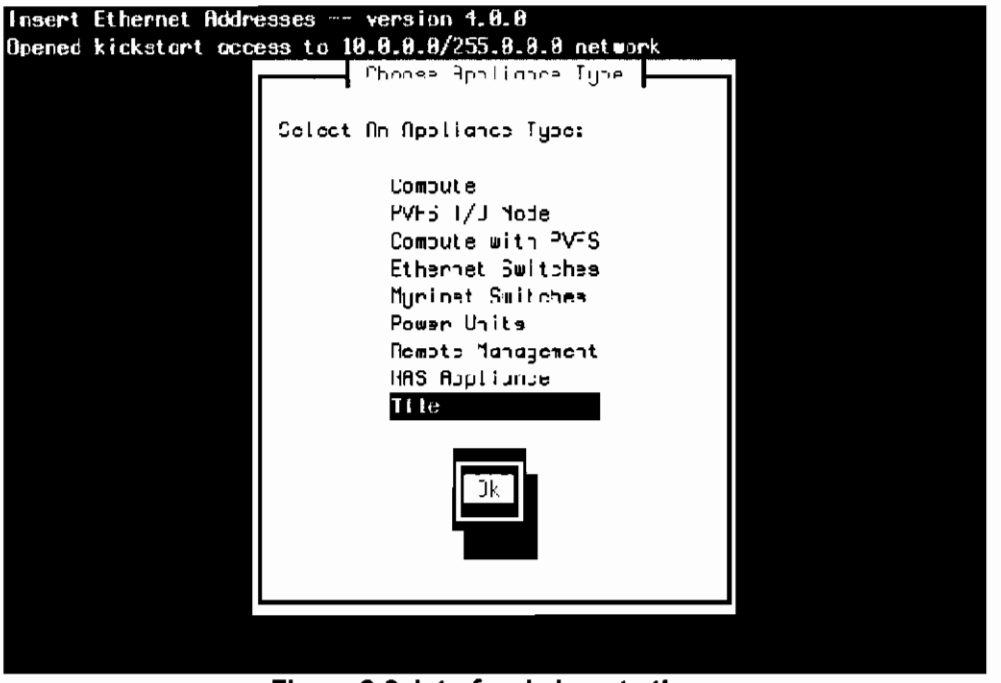


Figura 2-9. Interfaz de insert-ethers

2.3.1.4. Red Ethernet

Todos los nodos de cómputo utilizan Ethernet como red privada. Esta red se usa para administración y monitoreo, además permite compartir archivos.

---

[21] Los nodos I/O de PVFS se presentan en la Sección 4.5.5.1.1

[22] **insert-ethers** es un programa que se ejecuta en el *frontend* para capturar las direcciones MAC de los nodos y asignarles el tipo de dispositivo que se les instalará. El tipo de dispositivo es la funcionalidad que el nodo tendrá.

2.3.1.5. Red de Paso de Mensajes para las Aplicaciones

Todos los nodos pueden conectarse con redes clase Gigabit. Estas son redes de baja latencia y gran ancho de banda que permiten paso de mensajes con alto rendimiento para programas paralelos.

2.3.2. ESTRATEGIA DE ADMINISTRACIÓN

La estrategia de administración de *Rocks* se basa en la siguiente filosofía:

> "El desarrollo de cualquier versión de software en cualquier nodo del *cluster* debe ser trivial, sin importar el tamaño del *cluster*". [10]

Para implementar esta visión, se definen las siguientes características:

- Todo el software desarrollado para *clusters Rocks* está en RPMs.
- Se requiere una configuración 100% automática para los nodos de cómputo.
- Es esencial el uso de servicios escalables.

Red Hat ha desarrollado dos tecnologías principales que soportan directamente esta filosofía: los RPMs y la herramienta de instalación **Kickstart**.

Un paquete RPM contiene todos los archivos (binarios, archivos de cabecera, *scripts* iniciales, páginas de ayuda) para desplegar un módulo particular de software. Los RPMs se instalan usando la línea de comandos lo que permite usar métodos de programación para agregar o actualizar los paquetes.

Red Hat desarrolló un programa que automatiza la instalación de paquetes, denominado **Kickstart**. Los nodos instalados usando **Kickstart** se administran mediante un archivo de configuración creado por el usuario, que esencialmente contiene las respuestas a todas las preguntas, contestadas basándose en una instalación estándar. Los archivos *kickstart* pueden contener *scripts* que se

ejecutan durante la instalación. *Rocks* mantiene esta característica de *scripting*[23] para conseguir 100% de configuración automática de los nodos de cómputo.

En la Figura 2-10 se muestra un ejemplo de un archivo *kickstart*. En el archivo de ejemplo la sentencia `mouse genericps/2`, es la respuesta de un usuario a la pregunta del asistente de instalación de Red Hat acerca del tipo de mouse que se dispone.

```
url --url http://10.1.1.1/install/i386
zerombr yes
clearpart --all
part / --size 4096
lang en_US
keyboard us
mouse genericps/2
timezone --utc GMT
skipx
install
reboot

%packages
@Base
pdksh

%post
cat > /etc/motd << 'EOF'
Kickstarted on `date`
EOF
```

Fuente: [11], página 3

Figura 2-10. Ejemplo de un archivo *kickstart*

Para la instalación, se inicia los nodos de cómputo mediante un disquete o un CD o a través de PXE. Los nodos de cómputo cargan un *kernel* (que se encuentra en el medio desde el cual arrancaron) y se comunican con el *frontend*. Los nodos de cómputo envían su dirección MAC al *frontend*, el cual almacena esta información en una base de datos de MySQL. El *frontend* le asigna una dirección IP al nodo de cómputo, para lo cual se utiliza DHCP. Los nodos de cómputo, mediante Anaconda, usan HTTP para recibir los RPMs a través de la red. Las cuentas de usuario (claves y localizaciones de los directorios `/home`) se sincronizan desde el nodo *frontend* hacia los nodos de cómputo mediante el Servicio de Información de Red (NIS - *Network Information Service*) o mediante el Servicio de Información 411 (*411 Information Service*). El nodo *frontend* exporta todos los directorios de

[23] *Scripting* es un término utilizado para referirse a la actividad de realizar *scripts* para automatizar tareas.

los usuarios a los nodos de cómputo mediante el Sistema de Archivos de Red (NFS- *Network File System*).

2.3.3. HERRAMIENTAS PARA LA INSTALACIÓN Y CONFIGURACIÓN DE *Rocks*

La distribución de NPACI *Rocks* se instala y configura con la ayuda de algunas herramientas.

2.3.3.1. Administración activa de la configuración de los nodos

Los nodos se instalan usando la herramienta **Kickstart** de Red Hat. En *Rocks*, los archivos *kickstart* son dinámicos, son administrados de forma activa mediante su generación al momento (*on-the-fly*) con un *script* CGI, este *script* CGI se ejecuta en el *frontend*. Las funciones del *script* son:

- Construir un archivo de configuración general a partir de un conjunto de archivos de configuración basados en XML.

- Aplicar parámetros específicos de cada nodo mediante la consulta a la base de datos local.

Existen dos tipos de archivos de configuración basados en XML:

- *Nodes*: especifican los paquetes por módulo y los comandos de post configuración por paquete, para un servicio específico.

- *Graphs*: Enlazan los módulos definidos usando arcos (*edges*) directos, donde un arco representa una relación entre dos módulos.

Las raíces del grafo representan dispositivos (*appliances*), como: cómputo y *frontend*. Esta infraestructura de instalación basada en XML describe el comportamiento total de todos los nodos. En la Figura 2-11 se encuentra una sección de un archivo grafo, y la Figura 2-12 permite visualizar los componentes del grafo: dispositivos, módulos y relaciones entre módulos.

```
<?XML VERSION="1.0" STANDALONE="no"?>
<GRAPH>
        <DESCRIPTION>Default Graph for NPACI Rocks</DESCRIPTION>

        <EDGE FROM="frontend" TO="mpi-devel"/>
        <EDGE FROM="frontend" TO="dhcp-server"/>

        <EDGE FROM="compute" TO="mpi"/>
        <EDGE FROM="compute" TO="c-development"/>

        <EDGE FROM="mpi-devel" TO="c-development"/>
        <EDGE FROM="mpi-devel" TO="fortran-development"/>
</GRAPH>
```

Fuente: [10], página 11

Figura 2-11. Sección de un archivo grafo

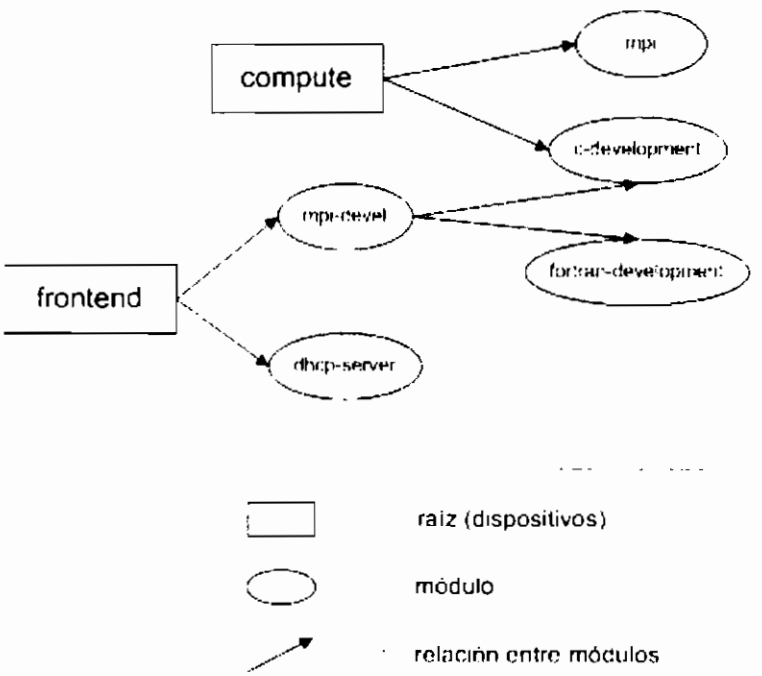


Figura 2-12. Visualización de la descripción de un grafo

El módulo es un indicativo de que software instalar, mientras que la raíz indica sobre cual nodo instalar el software; las relaciones indican que módulos se requieren para que un dispositivo sea funcional. Es posible incrementar los dispositivos haciendo modificaciones a los archivos grafo.

El procedimiento de instalación involucra los siguientes componentes:

- **Anaconda**: **Anaconda** es el nombre del proceso UNIX encargado de la instalación de un equipo basándose en un archivo *kickstart*. Cada máquina en el *cluster* ejecuta este proceso, iniciándolo desde una unidad de

disquete, una unidad de CD, un disco rígido o utilizando PXE. El proceso **Anaconda** realiza el requerimiento de un archivo *kickstart* (localizado en el medio de la instalación o en la red, en un servidor HTTP o NFS), analiza gramaticalmente las palabras del archivo y ejecuta los comandos apropiados para completar la configuración de la instalación del software en un computador. Una vez que **Anaconda** termina, el computador reinicia el sistema y se convierte en un miembro funcional del *cluster*.

- **CGI**: *Rocks* utiliza un *script* CGI, que se ejecuta en el *frontend*, para despachar los archivos *kickstart* al proceso **Anaconda**. Cuando se instala un nodo, **Anaconda** realiza un pedido del archivo *kickstart* utilizando HTTP. El nodo construye el URL combinando la información que obtiene de la respuesta DHCP enviada por el nodo *frontend* y su información específica (como nombre del disco duro[24] y tipo de arquitectura). Por ejemplo, un URL para un nodo con tecnología x86 y con dos discos SCSI es:

  http://frontend-0/install/kickstart.cgi?devname=sda,sdb&arch=i386

  El *script* CGI coordina la creación del archivo *kickstart* mediante la extracción de los campos específicos del nodo de los parámetros del URL, consultando la base de datos SQL y pasando estos valores a los programas subsecuentes: **KPP** y **KGen**. El *script* CGI toma la salida de **KGen** y lo envía de vuelta al nodo que realizó el pedido.

- **Base de Datos SQL**: La base de datos almacena información de configuración del *cluster* e información específica de los computadores y de los grupos de computadores.

- **KPP**: El Pre Procesador *Kickstart* (*Kickstart Pre-Processor*) atraviesa los grafos de configuración, realiza un pedido en base a la información del

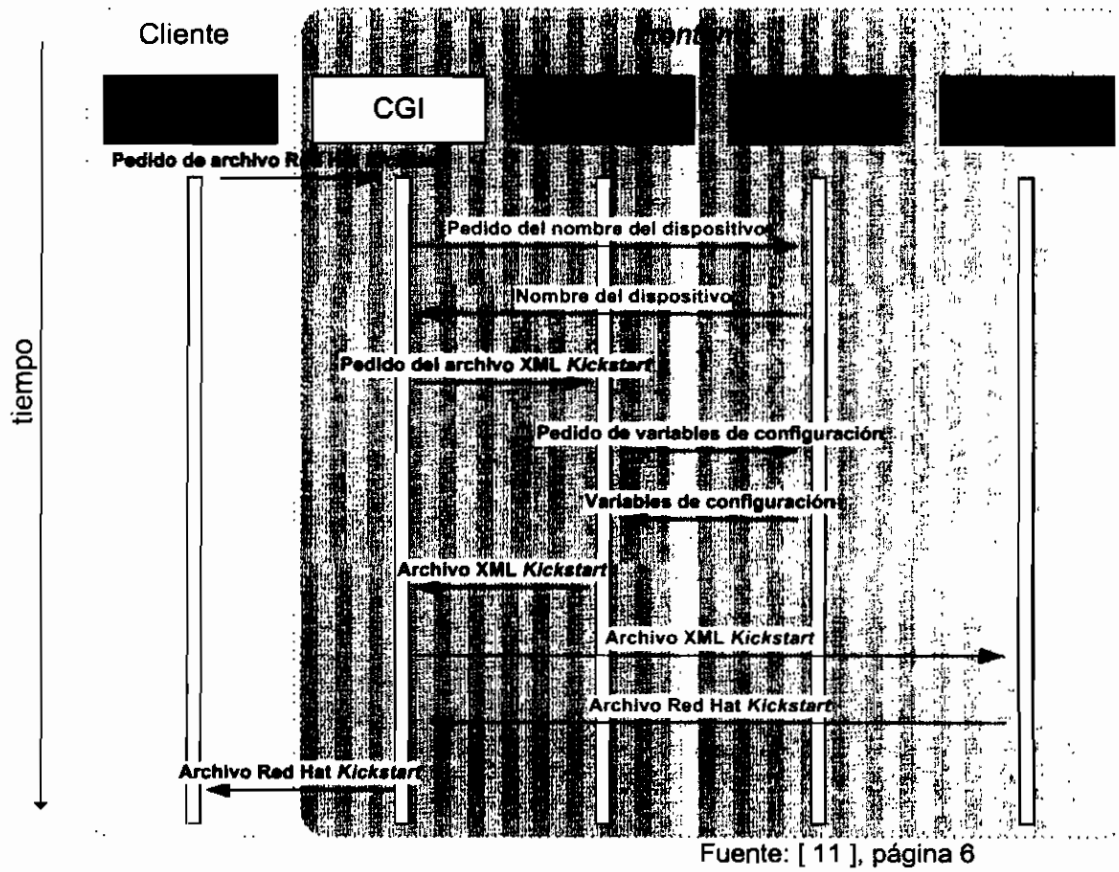
[24] En Linux, los dispositivos reciben un nombre para hacer referencia a los mismos, por ejemplo, un disco duro SCSI ubicado en la ranura 0 recibirá el nombre `sda`; un disco duro IDE ubicado como maestro en la primera ranura recibirá el nombre `hda`.

grafo a la base de datos de configuración de SQL y construye un archivo *kickstart* monolítico basado en XML para un nodo específico.

- **KGen:** El Generador *Kickstart* (*Kickstart Generator*) transforma un archivo *kickstart* XML a la sintaxis de un archivo Red Hat *kickstart*. Este paso adicional existe para permitir utilizar nuevos formatos. Por ejemplo, es posible utilizar otro generador para producir archivos *JumpStart*[25].

Este método es sumamente flexible, permitiendo soportar tanto hardware homogéneo como heterogéneo.

En la Figura 2-13 se muestra un diagrama espacio-tiempo de la generación de un archivo *kickstart*.



Fuente: [11], página 6

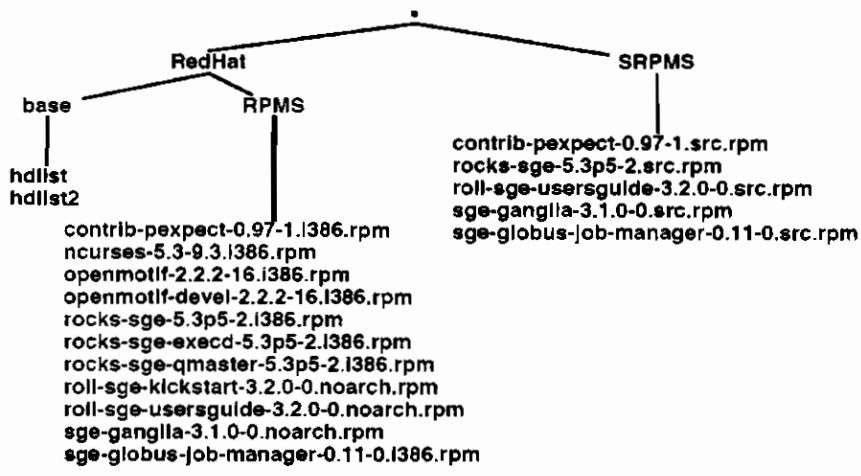
Figura 2-13. Diagrama espacio-tiempo de generación de un archivo *kickstart*

[25] Un *JumpStart* es un archivo de configuración para plataformas Solaris.

2.3.4. *ROLLS*

La mayoría de *Rolls* son sistemas opcionales de configuración automática. La mayoría de *Rolls* son desarrollados por grupos ajenos al grupo núcleo de *Rocks*. Un *Roll* provee tanto la arquitectura como los mecanismos que permiten al usuario de *Rocks*, el incrementar y modificar la descripción del grafo de todos los tipos de dispositivos. Se puede agregar nueva funcionalidad, y cualquier software proporcionado con *Rocks* puede ser sobrescrito o removido simplemente insertando el CD del *Roll* deseado al momento de la instalación.

Un *Roll* es una imagen ISO independiente que contiene paquetes y sus respectivos *scripts* de configuración. En la Figura 2-14 se muestra el contenido del *Roll* SGE [12]. Este diseño es casi idéntico a la imagen ISO de Red Hat, y permite usar los *scripts* proporcionados con Red Hat para asistir en la construcción del *Roll* y resulta familiar para aquellos desarrolladores que tienen experiencia en la construcción de distribuciones de Red Hat.



Fuente: [13], página 5

Figura 2-14. Contenido del *Roll* SGE

El directorio `/base` contiene dos archivos que pueden ser usados por las utilidades de Red Hat para consultar sobre el contenido del *Roll*. El directorio `/RPMS` contiene los paquetes binarios que contiene el *Roll*, y el directorio `/SRPMS`

contiene los archivos fuentes de los paquetes binarios incluidos en el directorio `/RPMS`.

Rocks también ha denominado *Rolls* al software necesario para poder instalar *Rocks.* Los Rolls requeridos para instalar *Rocks* son *Rocks Base* y HPC. El *Roll Rocks Base* comprende los RPMs del sistema operativo Linux, así como las herramientas de instalación; mientras que el *Roll* HPC contiene varias versiones de MPICH, Ganglia y PVFS [14]. Si se requiere software adicional que no es parte de la instalación núcleo, se deben descargar los *Rolls* específicos.

Los *Rolls* que actualmente están disponibles son:

- Motor *Grid* de Sun (SGE – *Sun Grid Engine*): Es un sistema de encolamiento de tareas para *grids.* Es una alternativa para *grids* de openPBS. Es un software de código abierto de administración distribuida.

- Intel *Roll* [16]: Este paquete instala y configura el compilador de C de Intel y el compilador FORTRAN de Intel. (Se requieren las licencias de Intel). También incluye los ambientes MPICH construidos para estos compiladores.

- Ambiente *Cluster* Escalable (SCE - *Scalable Cluster Environment*) [17]: Incluye el software OpenSCE que fue desarrollado en la Universidad Kasetstart de Tailandia.

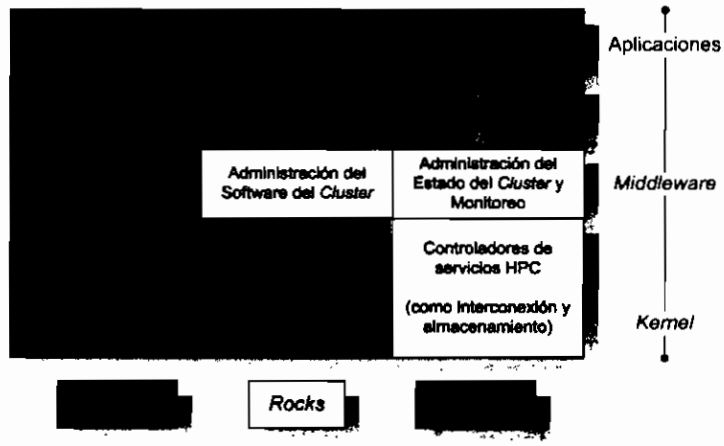
- Java *Roll* [18]: Incluye la Máquina Virtual de Java.

- PBS *Roll*: Incluye OpenPBS y Maui, además de software para encolamiento y planificación de tareas.

- Condor *Roll* [19]: Incluye el software de administración de carga Condor. Condor provee encolamiento de tareas, planificación y administración de prioridades además de administración y monitoreo de recursos.

2.3.5. COMPONENTES DE *Rocks*

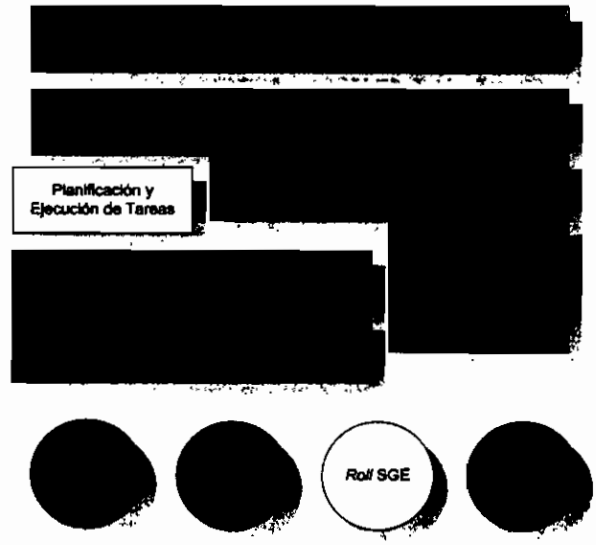
En la Figura 2-15 se muestra la pila de software de *Rocks*. Las capas en color azul son comunes a todo *cluster*. Las capas en color amarillo son propias de *Rocks* y la capa de color verde es específica para cada *cluster* y depende de la aplicación del mismo.



Fuente: [20], página 24

Figura 2-15. Componentes de *Rocks*

La Figura 2-16 muestra algunos *Rolls* que conforman los componentes de la pila de software de *Rocks*. Cada color define un conjunto de aplicaciones que viene en un *roll* determinado.



Fuente: [20], página 13

Figura 2-16. Arquitectura basada en *rolls* de *Rocks*

2.3.5.1. Paquetes comunes en todo *cluster*

En todo *cluster* existe un conjunto de paquetes que son comunes. En la Figura 2-17 se muestra parte de la pila de componentes de *Rocks* que son comunes en todo *cluster*.

Figura 2-17. Componentes comunes de *Rocks* en todo *cluster*

2.3.5.1.1. El Kernel de Linux y el Entorno Linux

Rocks escogió como base a Red Hat debido a su excelente soporte para instalación automática, a su buena detección de hardware y debido a que la instalación puede realizarse automáticamente mediante *scripts* (*kickstart*). En las Versiones 3.0.0 y 2.3.3 de *Rocks* se utilizó Red Hat 7.3 como entorno Linux y la Versión 2.4 del *kernel* de Linux. En las Versiones de 3.2.0 y 3.3.0 de *Rocks* se utilizó Red Hat Enterprise Linux 3.0 (RHEL 3.0) tanto en el entorno como en el *kernel*. Las versiones basadas en RHEL son recompiladas a partir de archivos SRPMS, disponibles en la página de Red Hat Linux, corrigiendo los errores que pueden tener hasta la fecha de la publicación de la versión de *Rocks*. En la última versión (4.0.0) liberada en junio de 2005 se utiliza CentOS[26] [21] como entorno Linux y como *kernel*. El *kernel* ha sido recompilado para varios tipos de procesadores (*Pentium, Athlon, Opteron, Itanium, Xeon*), pero sin agregar ninguna modificación o parche. Por el momento, *Rocks* no soporta ninguna otra

[26] CentOS (*Community ENTerprise Operating System*) es un sistema operativo desarrollado por el Equipo de Desarrollo CentOS (CentOS *Development Team*). Este sistema operativo fue construido a partir de los SRPMs de Red Hat Enterprise 4.0 disponibles de manera gratuita en el sitio Web de Red Hat. CentOS incluye versiones de paquetes comerciales, que han sido modificados para incluir licencias libres.

distribución de Linux; sin embargo, se está investigando soporte para sistemas basados en SuSe.

En la Tabla 2-3 se muestra un resumen de las versiones de *Rocks*, con sus diferentes *kernels* y entornos utilizados.

Versión	Fecha	*kernel*	Entorno
Rocks 2.3.3 (Annapurna)	08 - 2003	2.4	RH 7.3
Rocks 3.1.0 (Matterhorn)	12 - 2003	2.4	RH 7.3
Rocks 3.2.0 (Shasta)	05 - 2004	RHEL 3.0	RHEL 3.0
Rocks 3.3.0 (Makalu)	10 - 2004	RHEL 3.0	RHEL 3.0
Rocks 4.0.0 (Whitney)	06 - 2005	CentOS	CentOS

Tabla 2-3. Entornos y *kernels* utilizados por *Rocks*

2.3.5.1.2. Planificación y Ejecución de Tareas

En *Rocks* se pueden encontrar varias aplicaciones que permiten la planificación y ejecución de tareas.

PBS y MAUI

PBS es un sistema desarrollado por la NASA a mediados de los 90s. Es un sistema flexible de encolamiento por lotes. Opera en ambientes multiplataforma UNIX. Por largo tiempo ha sido un estándar para sistemas HPC de encolamiento. Maui es un planificador de tareas, establece las políticas que permiten definir que tarea se ejecutará en cuál cola y en qué momento. Existen múltiples implementaciones de PBS: OpenPBS, PBSPro [22].

PBS es analizado con más detalle en la Sección 4.4 y se presentan algunas características de Maui en la Sección 4.3.3.

***Sun Grid Engine* (SGE)**

Está convirtiéndose rápidamente en un nuevo estándar. Fue integrado en *Rocks* por SCS. SGE es el sistema de lotes por defecto en los *clusters Rocks* desde la Versión 3.2.0 y permite disponer de un conjunto de nodos para ejecutar una tarea en paralelo. SGE es un software de administración de recursos distribuidos y permite el uso eficiente de los recursos de un *cluster*.

Las tareas se envían al Motor *Grid* a través de *scripts*, para esto se utiliza el comando:

```
$ qsub [nombre_script]
```

Se puede investigar el estado de la cola ejecutando el siguiente comando:

```
$ qstat -f
```

La salida de la tarea se coloca mediante el Motor *Grid* en cuatro archivos. Uno de los archivos contiene los mensajes de salida y el otro contiene mensajes de error. Los otros dos archivos contienen mensajes de estado del Motor *Grid*.

mpirun

mpirun se utiliza para ejecutar programas desarrollados con MPI. Para ejecutar un programa desarrollado con MPI se utiliza el siguiente comando:

```
$ mpirun -np <numero_de_procesos> <nombre_programa argumentos >
```

MPD

MPD[27] permite ejecutar tareas. Es un programa de alto rendimiento desarrollado por el Laboratorio Nacional Argonne. Sirve como reemplazo a **mpirun**, y puede

[27] MPD es el nombre del programa desarrollado por el Laboratorio Nacional Argonne. No son siglas.

usarse para ejecutar tareas paralelas. **MPD** puede iniciar aplicaciones paralelas MPI y no MPI.

Como ventajas de **MPD** se pueden mencionar las siguientes:

- Ejecución rápida. **MPD** puede iniciar una tarea en 100 nodos en menos de un segundo [9].

- Limpieza después de realizar la tarea. **MPD** propaga adecuadamente las señales ^C[28] (SIGTERM) y ^Z[29] (SIGINT), permitiendo parar y reiniciar una tarea mediante un solo comando en el nodo *frontend*.

- Tolerancia a fallos. **MPD** puede iniciar tareas y entregar señales incluso frente a fallos en los nodos.

Como inconvenientes de **MPD** se pueden mencionar:

- Compatibilidad. Las aplicaciones de MPI deben recompilarse para usar el lanzador de tareas **MPD**.

- Seguridad. **MPD** no usa **ssh** para lanzar tareas o repartir señales, y no intenta encriptar comandos. Este nivel de seguridad es solo apropiado para *clusters* con una red interna protegida.

- Complejidad. **MPD** se basa en un anillo[30] de demonios en los nodos del *cluster*, los cuales deben ser creados y mantenidos.

---

[28] ^C es la señal de terminación de un proceso en UNIX.

[29] ^Z es la señal para pausar un proceso en UNIX.

[30] El anillo de demonios es un conjunto de demonios mpd conectados, cada demonio tiene una conexión TCP con otros dos computadores.

- Velocidad. Todos los nodos que ejecutan la tarea piden una copia del ejecutable de la aplicación casi al mismo tiempo, causando inconvenientes al servidor NFS.

cluster-fork

Usualmente se requiere ejecutar programas paralelos que consisten de comandos estándar UNIX, donde paralelos se refiere a ejecutar el mismo comando sobre múltiples nodos. Estas tareas pueden usarse para mover archivos, ejecutar pequeñas pruebas y realizar varias tareas administrativas. Para realizar esto *Rocks* provee la herramienta denominada **cluster-fork**.

Por defecto, **cluster-fork** usa una serie de conexiones **ssh** para ejecutar las tareas serialmente en cada nodo de cómputo en el *cluster*. **cluster-fork** puede detectar nodos caídos. Usualmente la tarea es "bloqueante"; es decir, **cluster-fork** espera que la tarea inicie en un nodo antes de moverse al siguiente.

cluster-fork permite seleccionar los nodos en los que la tarea será ejecutada. Para ésto se puede utilizar una sentencia SQL o especificar los nodos utilizando una forma especial rápida (*shorthand*).

Para usar la sentencia SQL se usa la base de datos SQL del *frontend*. Se requiere una sentencia SQL que retorne una columna con los nombres de los nodos. Por ejemplo, para ejecutar un comando en los nodos de cómputo del primer gabinete[31], se podría ejecutar:

```
$ cluster-fork -query="SELECT name FROM nodes WHERE name LIKE \
  'compute-1-%'" [comando]
```

Con la forma especial rápida se especifica un rango de nodos. Esta forma funciona cuando los nombres comparten un prefijo común y las variaciones entre

[31] *Rocks* asigna los nombres de los nodos de cómputo con el prefijo `compute-1-`, por este motivo se habla de los nodos de cómputo del primer gabinete.

los nombres son números. Por ejemplo, para ejecutar un comando en los nodos del 0 al 4 del primer gabinete, se puede ejecutar:

```
$ cluster-fork -nodes=compute-1-%d:0-4 [comando]
```

2.3.5.1.3. Capa de Comunicación

En la capa de comunicación se pueden encontrar varios métodos para la comunicación entre nodos:

- Ninguno (Embarazosamente Paralelo, *Embarrassingly Parallel*).
- *Sockets* (Modelo cliente-servidor, utiliza comunicación punto a punto).
- MPI.
- PVM.

En la Figura 2-18 se presenta la comunicación con *sockets* y la comunicación con MPI/PVM. En la Figura 2-18 (a) se puede observar que la comunicación con *sockets* es punto a punto, lo que significa que si el *cluster* está conformado por ***n*** nodos, se requieren ***(n^2-n) / 2*** conexiones para establecer comunicación con todos los nodos. Por otro lado, la Figura 2-18 (b) muestra la comunicación utilizando MPI o PVM, en las cuales se dispone de un canal virtual compartido para las comunicaciones. El canal virtual de MPI o PVM no es más que un conjunto de llamadas para paso de mensajes que facilitan el escribir programas; sin embargo, MPI y PVM implementan este canal virtual mediante *sockets*.

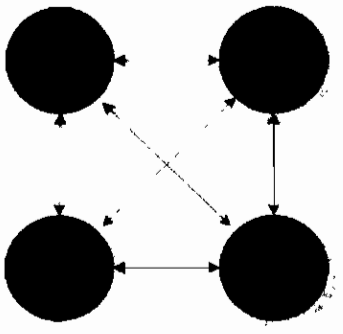
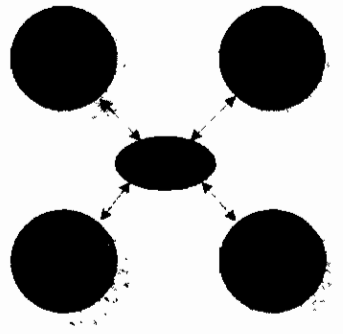


Figura 2-18. (a) Comunicación utilizando *sockets*



Fuente: [23], página 29

Figura 2-18. (b) Comunicación utilizando MPI o PVM

PVM

Rocks, desde la Versión 3.1.0, incluye la versión 3.4.3 de PVM. Esta versión tiene soporte en 60 variantes de UNIX y soporte en Windows NT/2000/XP/2003. Permite control de recursos y dispone de tolerancia a fallos.

MPI

Rocks incluye la Versión 1.1 de MPI. Esta versión tiene soporte para GNU C, Fortran 77, Intel C, Fortran 90, Portland Group C, C++, pero requiere las licencias correspondientes pasa su uso.

2.3.5.1.4. NFS

En *Rocks*, las cuentas de los usuarios se despachan utilizando NFS. NFS funciona bien en *clusters* pequeños (con menos de 128 nodos); sin embargo, tiene problemas de escalamiento en *clusters* de mayor tamaño.

2.3.5.1.5. OpenSSH

En *Rocks* se utiliza **ssh** para reemplazar a **telnet** y **rsh**. **ssh** permite autenticar y encriptar. En *Rocks* se utiliza **ssh** al lanzar aplicaciones MPI o al utilizar **cluster-fork**. Dispone además de **ssh-agent** para administrar las claves SSH.

2.3.5.1.6. NIS y DHCP

NIS se utiliza para administrar la información de las cuentas de los usuarios, así como información de los equipos. Es la alternativa a copiar los archivos `/etc/hosts`, `/etc/passwd`. Los clientes se conectan con el servidor más cercano. NIS estaba incluido en las primeras versiones de *Rocks*; sin embargo, fue reemplazado desde la Versión 3.1.0 por el Servicio 411.

DHCP administra información de red. Es la alternativa a tener direcciones IP estáticas. Es la base de **insert-ethers**[32].

2.3.5.1.7. SNMP

SNMP está habilitado en todos los nodos de cómputo. Excelente para uso punto a punto. Soporta la MIB (*Management Information Base*) para Linux, la cual permite obtener información como: tiempo de funcionamiento (*uptime*), carga, estadísticas de red, procesos de ejecución, software disponible. No ofrece un buen escalamiento si se usa para obtener información de todo el *cluster*.

2.3.5.1.8. Syslog

Es el encargado de monitorear la actividad en todas las implementaciones de sistemas UNIX. Cada nodo de cómputo almacena los eventos locales en `/var/log/message`, para luego enviarlos al *frontend*.

2.3.5.1.9. Servicio de Información Segura 411

El Servicio de Información Segura 411 provee funcionalidad similar a la ofrecida por NIS. Su nombre se debe al código 411 usado por los sistemas telefónicos para proveer información. 411 se usa para distribuir archivos de claves y archivos de configuración de usuarios y grupos de manera segura.

411 utiliza Criptografía de Llave Pública (PKC – *Public Key Cryptography*) para proteger el contenido de los archivos. Opera a nivel de archivo. 411 no se construyó sobre RPC como NIS, y en lugar de eso distribuye los archivos usando HTTP a través de un servicio Web. Su tarea central es el mantener seguros archivos críticos de *login/password* sobre los nodos de cómputo del *cluster*. Esto lo realiza mediante la implementación de una base de datos distribuida basada en

[32] **insert-ethers** es un programa incluido en *Rocks* que se encarga de obtener las direcciones MAC de los nodos cliente, asignarles un nombre y guardar esta información en la base de datos de MySQL.

archivos. Los objetivos de diseño de 411 incluyen escalabilidad, seguridad, baja latencia cuando ocurren cambios, y resistencia a fallos.

2.3.5.2. Software del *cluster Rocks*

Rocks incluye un conjunto de componentes propios[33]; en la Figura 2-19 se muestran estas partes de la pila de componentes de *Rocks*.

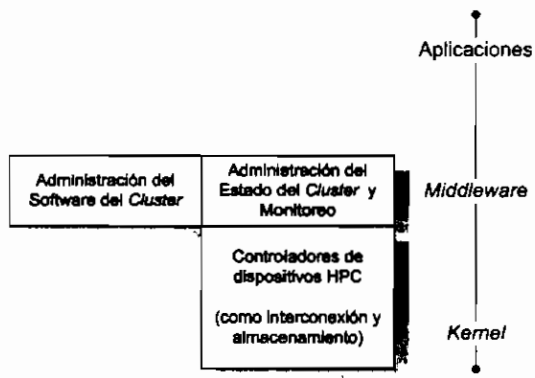


Figura 2-19. Componentes propios de *Rocks*

2.3.5.2.1. Administración del Estado del Cluster

Rocks permite obtener información estática o dinámica del *cluster*. Dentro de la información estática que *Rocks* provee se encuentra la dirección del nodo, el tipo de nodo o la configuración específica. Dentro de la información dinámica que *Rocks* provee se encuentra la utilización del CPU, la utilización del disco e información sobre la disponibilidad de los nodos (nodos en línea). Para esto se utiliza tanto MySQL como Ganglia.

Ganglia

Ganglia es un sistema de monitoreo de *clusters*. Fue desarrollado por Matt Massie de la UCB (*University of California Berkeley*). En cada nodo se ejecuta el demonio **gmom**. **gmom** escucha a los demonios de los otros nodos. Todos los datos se representan utilizando XML. Para extender Ganglia se puede utilizar **gmetric**, que es una interfaz de línea de comandos que permite recuperar métricas.

[33] El nombre de componentes propios es asignado por *Rocks* para referirse a paquetes, que no son desarrollados por *Rocks*, pero que han sido incluidos en el *Roll* HPC.

Ganglia es un sistema de monitoreo distribuido y escalable para sistemas de computación de alto rendimiento como *clusters* y *grids*. Se basa en un protocolo *multicast* basado en escuchar y anunciar para monitorear el estado en el *cluster*.

Ganglia utiliza tecnologías ampliamente usadas como XML para la representación de los datos, **XDR** (XML *Data Reduce*) para transportar datos compactados, y **RRDtool** (*Round Robin Database Tool*) para el almacenamiento y la visualización de datos.

En la Sección 4.2 se presentan mayores detalles de Ganglia.

phpMyAdmin

Es una herramienta escrita en PHP que trata de manejar la administración de MySQL en WWW. Actualmente puede crear y eliminar bases de datos, crear/borrar/alterar tablas, eliminar/editar/agregar campos, ejecutar cualquier sentencia SQL, administrar claves principales en las tablas, administrar privilegios, exportar datos en varios formatos y está disponible en 47 idiomas.

Particionamiento de Discos Duros y eKV

El particionamiento de los discos de los clientes puede realizarse desde el *frontend* de forma automática o manual. Si se selecciona el formato automático, se realizarán particiones / (*root*[34]) de 4 GB en el disco duro. Cuando un nodo es reinstalado, esta partición es formateada.

eKV (Ethernet *Keyboard and Video*) es un programa que permite observar e interactuar con **Kickstart** a través de Ethernet. Permite interactuar de forma remota con la instalación (inicial o reinstalación).

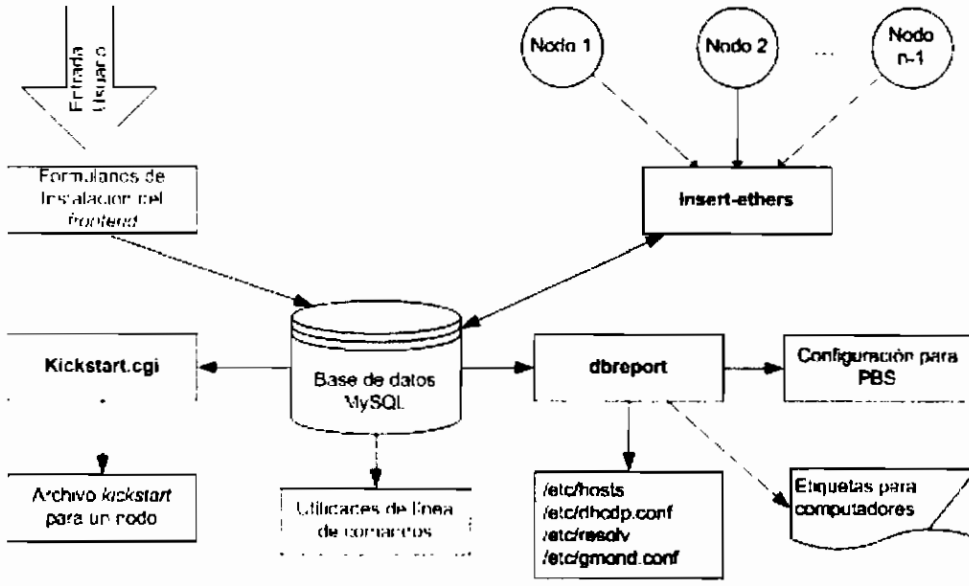
Base de Datos del *Cluster*

Rocks utiliza una base de datos desarrollada en MySQL. La base de datos de *Rocks* permite obtener la configuración de todo el *cluster* a través de consultas SQL. Esto permite escribir *scripts* para recuperar datos y asignarles un formato

[34] */* o *root* es el nombre de la partición raíz del sistema de archivos Linux.

específico, así como administrar los archivos de configuración a través de una herramienta específica.

La base de datos también permite a las aplicaciones almacenar (**insert-ethers**) y extraer información dinámica (**makehosts**, **makedhcp**, **gmom**), como se muestra en la Figura 2-20.



Fuente: [24], página 32

Figura 2-20. Base de datos *Rocks* y programas que interactúan con la misma

2.3.5.2.2. Administración del Software del Cluster

Los paquetes de software de *Rocks* se administran utilizando la herramienta **rpm**. Para administrar el repositorio de RPMs, se dispone de la herramienta **Rocks-dist**, la cual permite definir una nueva distribución de *Rocks* que se ajuste a los requerimientos de los desarrolladores. Para configurar el software del cluster se puede utilizar **rpm** o utilizar **Kickstart**.

Rocks-dist
Es una herramienta que permite construir una distribución de *Rocks*. Además permite desarrollar CDs de tipo *bootable*.

Red Hat *Kickstart*

Un archivo *kickstart* está formado por cuatro secciones:

- Sección principal: Contiene información sobre el particionamiento del disco, el *password* del usuario *root*, la dirección URL del repositorio de paquetes RPM, entre otros.

- Sección de Paquetes: Define la lista de RPMs. Los RPMs se agrupan en conjuntos específicos de paquetes. El repositorio es el que determina las versiones de los RPMs que serán instalados.

- Sección de pre-instalación: Son un conjunto de *scripts* que se ejecutan antes de la instalación de los RPMs.

- Sección de post-instalación: Son un conjunto de scripts que se ejecutan después de la instalación de los RPMs, permiten corregir errores conocidos (*bugs*) en los paquetes así como agregar información local.

Rocks* XML *kickstart

Permite procesar y descomponer un archivo *kickstart* en nodos y grafos. Los nodos especifican un servicio y su configuración, mientras que los grafos definen el marco de trabajo. Utilizan la información existente en la base de datos del *cluster* para generar el archivo *kickstart* específico, basándose en la información que obtienen a través de un *script* CGI.

2.3.5.2.3. Módulos HPC

PVFS

El objetivo del Sistema de Archivos Virtual Paralelo (PVFS – *Parallel Virtual File System*) es explorar el diseño, la implementación y el uso de I/O paralelo.

PVFS soporta una interfaz I/O UNIX y permite que programas UNIX usen los archivos PVFS sin tener que ser recompilados. Las herramientas para archivos de UNIX (**cp**, **ls**, **rm**) operan sobre archivos y directorios PVFS.

PVFS puede separar (*stripe*) los datos de los archivos en múltiples discos, en diferentes nodos de un *cluster*.

Rocks realiza la instalación de PVFS como un módulo del *kernel* en todos los nodos. *Rocks* incluye un soporte inicial, pero no incluye toda la funcionalidad. Se espera tener un soporte total de PVFS en versiones futuras de *Rocks*. Por defecto *Rocks* no habilita PVFS.

En la Sección 4.5 se presentan características adicionales de PVFS.

Myrinet

Rocks incorpora módulos del *kernel* para computadores con tarjetas Myrinet; estos módulos habilitan las aplicaciones para que utilicen MPI/GM. Para la administración de los puertos se utiliza el demonio **Usher**.

GM es el sistema de paso de mensajes para Myrinet. El paquete GM contiene *drivers*, API y librerías.

Usher

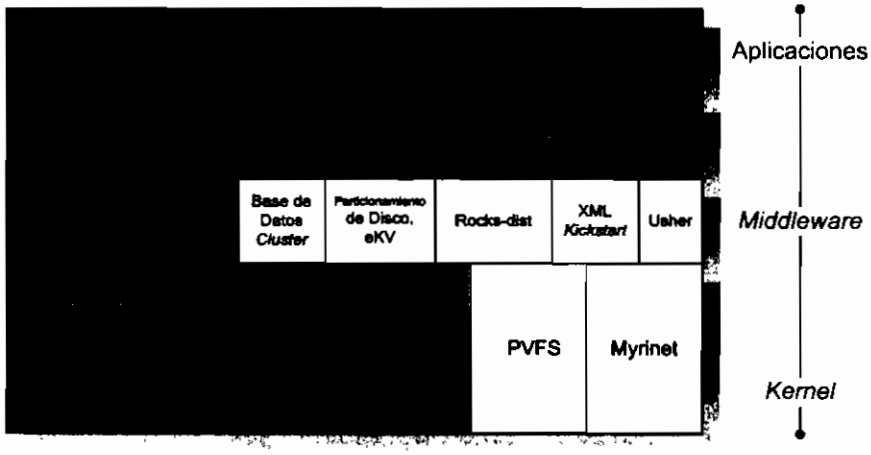
Usher es el administrador de recursos distribuidos para puertos GM. Se basa en RPC. Utiliza un sistema de reservaciones y permite ejecutar múltiples tareas GM por nodo. Es transparente y se integra en las aplicaciones para ejecutar tareas **mpi-launch** y **rexec**.

2.3.5.3. Aplicaciones del *Cluster*

Hay aplicaciones libres que han sido desarrolladas para ser integradas en *Rocks,* como **GAMESS** [25], **NAMD** [26] y **NWChem** [27]; u otras que requieren licencia, como **AMBER** [28]. También existen aplicaciones que se basan en *Rocks* como

BioBrew [29], Las aplicaciones que se pueden ejecutar en el *cluster* pueden ser provistas por vendedores o por los mismos usuarios y deben ser compiladas para poder ser ejecutadas. Estas aplicaciones se describen con más detalle en la Sección 5.1.

La Figura 2-21 muestra las diferentes aplicaciones que se pueden tener en cada sección de la pila de software de *Rocks*.



Fuente: [24], página 57

Figura 2-21. Aplicaciones y programas que conforman la pila de software de *Rocks*

2.3.6. INSTALACIÓN DE *Rocks*

Para la instalación de *Rocks* se requiere los CDs apropiados. Típicamente, se pueden descargar las imágenes ISO de *Rocks* del sitio Web http://rocks.npaci.edu/Rocks/ y proceder a quemar CDs con estas imágenes.

La ventaja de usar *Rocks* al construir y mantener un *cluster* es la simplicidad. En general, el construir un *cluster* es simple, pero el administrar su software puede ser complejo.

Esta complejidad llega a ser inmanejable durante el proceso de instalación del *cluster* y su expansión. *Rocks* provee mecanismos para controlar la complejidad en el proceso de instalación y expansión, y provee herramientas de monitoreo del rendimiento.

Rocks utiliza una estructura denominada grafo para crear archivos de descripción y utiliza una base de datos de MySQL para almacenar toda la información del *cluster*. Cuando un nodo inicia, envía su dirección MAC a la dirección MAC de *broadcast*. El *frontend* captura la dirección MAC del nodo mediante **insert-ethers** y la almacena en la base de datos de MySQL. El *frontend* le informa al nodo de su existencia. El nodo procede a solicitar una dirección IP, entonces, el servidor DHCP del nodo *frontend*, replica una etiqueta que contiene el URL del archivo *kickstart* que debe generarse para este nodo. El nodo se contacta con el servidor Web y un *script* CGI se ejecuta buscando el tipo de nodo en la base de datos y a traviesa el grafo para crear dinámicamente el archivo *kickstart*. Una vez que la descripción es descargada, el instalador empieza a descargar desde la localidad especificada en el archivo *kickstart*, los paquetes que serán instalados, luego realiza tareas específicas de post-instalación y reinicia el computador correspondiente.

Primero se procede a instalar el nodo *frontend* utilizando el CD de *Rocks*. El instalador se encargará de configurar automáticamente los servicios que ejecutará el nodo *frontend*.

Luego, el asistente procede a definir la configuración de los nodos de cómputo. Se inician los nodos mediante un disquete, un CD o utilizando PXE. Los nodos cargan un *kernel* en memoria, el cual contiene todas las herramientas necesarias para proceder con la instalación. Para cada nodo de cómputo se detecta la dirección MAC utilizando la herramienta **insert-ethers**, se asigna una dirección IP al nodo, se realiza una instalación completa del sistema operativo y se procede a configurar el nodo basándose en la información del archivo *kickstart*.

Una vez que todos los nodos de cómputo han sido configurados, se procede a reiniciar los servicios del nodo *frontend*; los servicios se reinician de forma automática para tener consistencia con la nueva configuración.

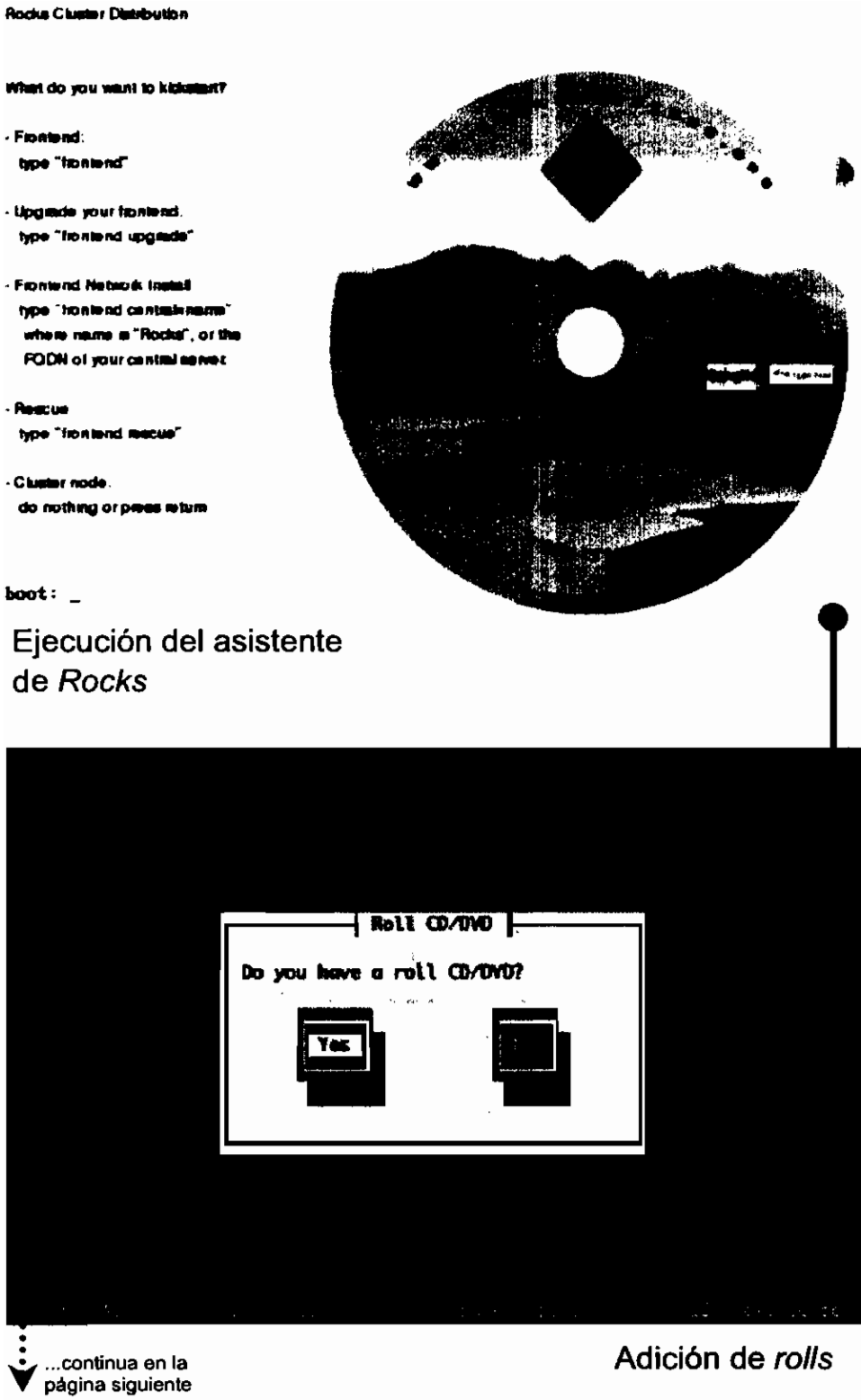
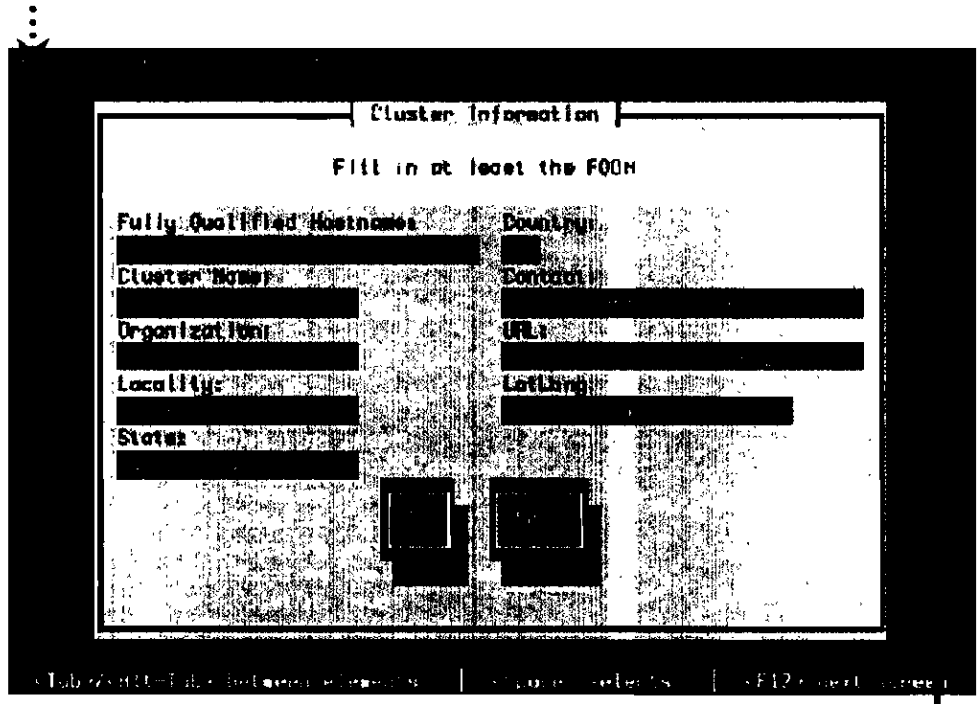
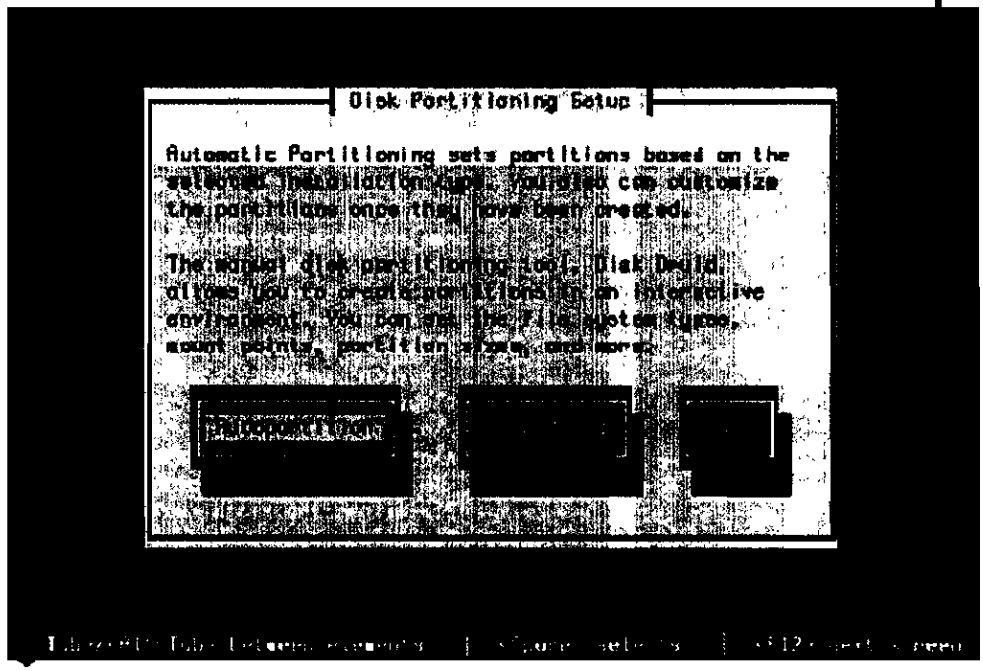


Figura 2-22. Proceso de instalación del *frontend* utilizando *Rocks*



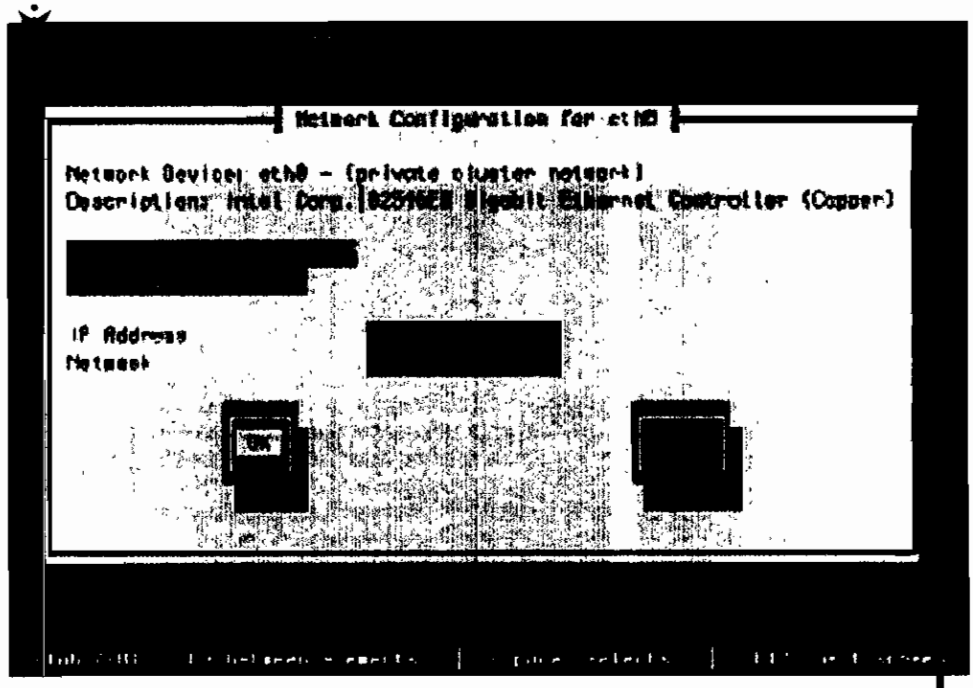
Información del *cluster*



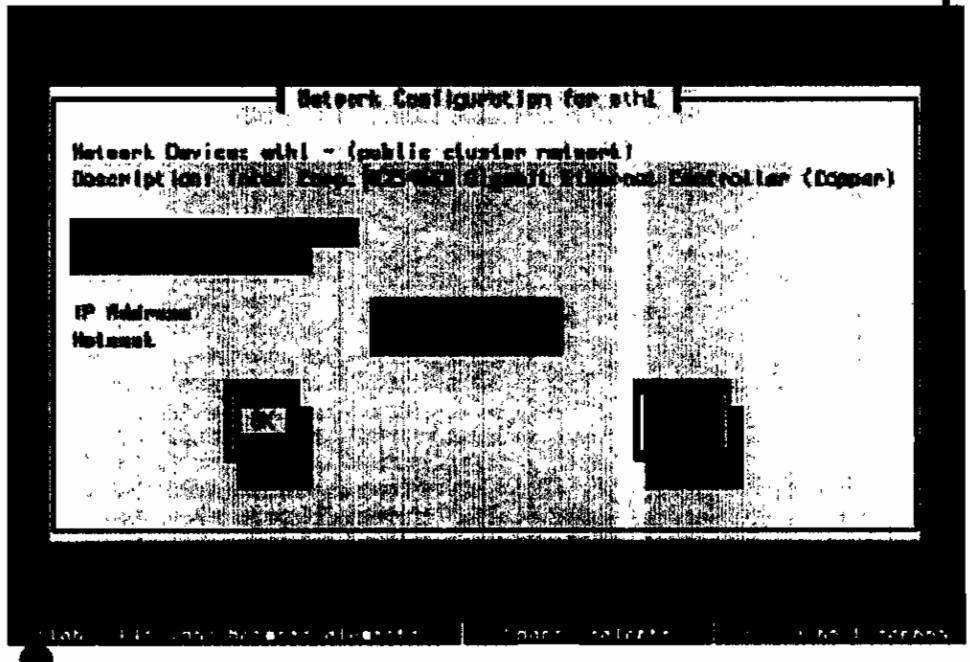
Particionamiento del disco duro

...continua en la página siguiente

Figura 2-22 (continuación). Proceso de instalación del *frontend* utilizando *Rocks*



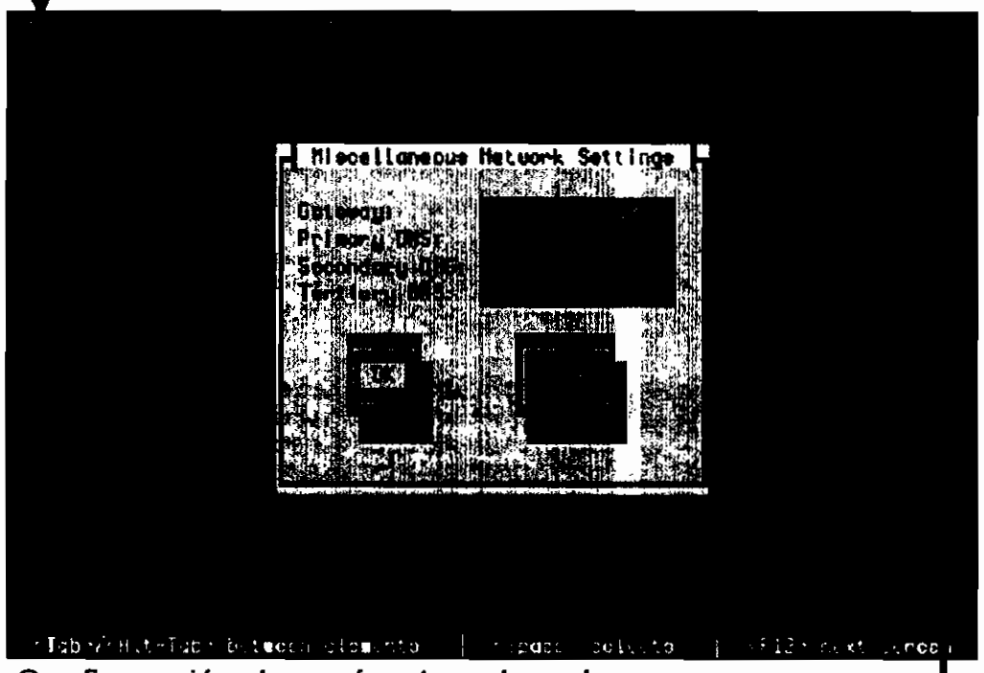
Configuración de
la interfaz eth0



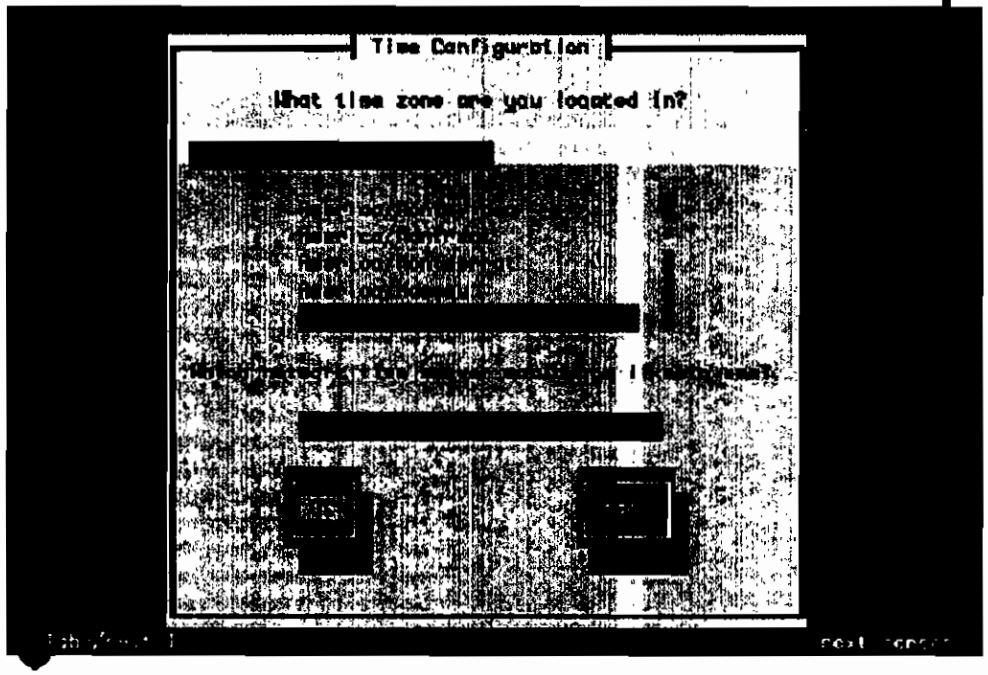
Configuración de
la interfaz eth1

...continua en la
página siguiente

Figura 2-22 (continuación). Proceso de instalación del *frontend* utilizando *Rocks*



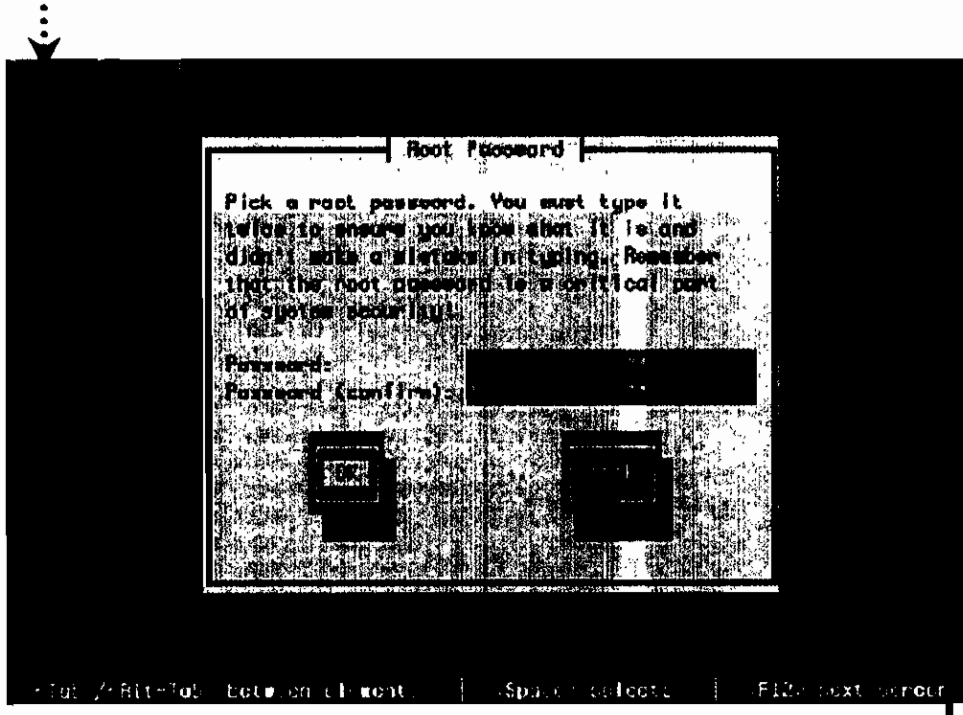
Configuración de parámetros de red adicionales: DNS y *gateway*



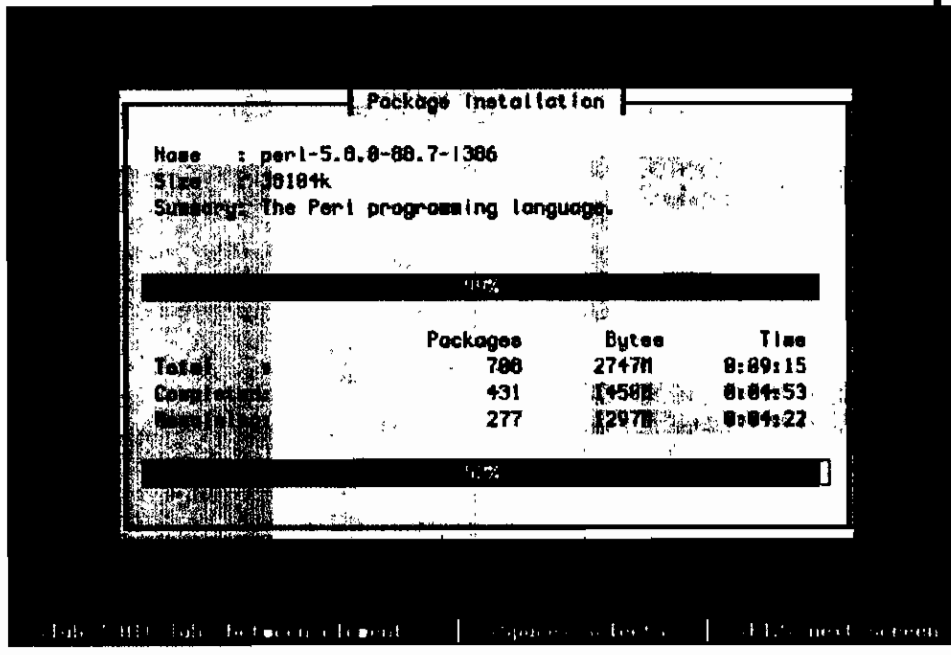
Configuración de NTP

...continua en la página siguiente

Figura 2-22 (continuación). Proceso de instalación del *frontend* utilizando *Rocks*



Password de la cuenta *root*



Instalación de paquetes

Figura 2-22 (continuación). Proceso de instalación del *frontend* utilizando *Rocks*

2.4. OTRAS HERRAMIENTAS PARA INSTALACIÓN AUTOMÁTICA DE *CLUSTERS*

2.4.1. SCore

El Consorcio de Computación del Mundo Real [30] (RWCP – *Real World Computing Partnership*) es un grupo de investigación de Tokio formado en 1992. RWCP resuelve un amplio rango de problemas relacionados con *clusters* que incluyen comunicaciones de bajo nivel, alto rendimiento y administración de sistemas. Este grupo desarrolló un paquete denominado SCore, el cual provee integración semiautomática de nodos usando la herramienta de instalación interactiva de Red Hat y un lanzador de tareas similar a **rexec** de UC Berkeley de Estados Unidos. La última versión (SCore 5.4.0) fue liberada en junio de 2003.

2.4.2. *SCYLD BEOWULF*

Scyld Beowulf [31] fue desarrollado por la Corporación de Computación SCYLD. *Scyld Beowulf* es un sistema operativo, el cual presenta una Imagen Única del Sistema (SSI) a los usuarios, modificando tanto el *kernel* de Linux, como la librería GNU de C y algunas utilidades a nivel de usuario. En los *clusters Scyld*, la configuración de los nodos cliente se realiza mediante un programa desarrollado por *Scyld*, el cual corre sobre el nodo servidor. *Scyld* provee un buen programa de instalación, sin embargo no tiene soporte para nodos heterogéneos. La última versión (*Scyld Beowulf Series 29 cz-5*) fue liberada en mayo de 2005.

2.4.3. AMBIENTE DE *CLUSTERS* ESCALABLES

El proyecto SCE (*Scalable Cluster Environment*) es un esfuerzo sobre *clusters* desarrollado por la Universidad Kasetsart de Tailandia. SCE es un paquete de software que incluye herramientas que permiten instalar software, administrar, monitorear y planificar tareas en el *cluster*. Es responsabilidad del usuario instalar el nodo servidor con Red Hat Linux, para luego agregar la funcionalidad de SCE usando una interfaz gráfica GUI. La instalación y el mantenimiento de los nodos

cliente se realizan usando las características de una Imagen Única de Sistema a través del *booteo* de red en nodos sin disco o usando las características de Red Hat en nodos con disco. La información del sistema se obtiene y se visualiza con una herramienta Web denominada **SCMSWeb**[35]. La última versión (OpenSCE 1.6.2) fue liberada en mayo de 2005. *Rocks* y SCE trabajan en conjunto, es posible agregar las características de SCE a un *cluster Rocks* utilizando el *roll* SCE.

2.4.4. Cfengine

Cfengine [32] ha sido desarrollado por *Oslo University College* de Noruega; es una herramienta de administración basada en políticas y permite configurar equipos UNIX o NT. Cfengine no instala el sistema operativo base. Después de la instalación del sistema operativo (manual o a través de alguna herramienta), se utiliza Cfengine para realizar la configuración inicial de los nodos y luego mantener la configuración consistente mediante consultas a un archivo central de políticas (accesible a través de NFS). El archivo central de políticas se escribe en un lenguaje de configuración específico de Cfengine, el cual permite al administrador definir la configuración para todos los nodos dentro del dominio de administración. Cada nodo que habilite Cfengine consulta este archivo para mantener su configuración coherente. La última versión (Cfengine 2.1.14) fue liberada en diciembre de 2004.

2.4.5. LCFG

LFCG [33] es un instalador desarrollado por la Universidad de Edinburgh del Reino Unido, que se basa en descripciones. LCFG hace uso de un lenguaje de configuración propietario para sus archivos fuente y provee un compilador con un perfil específico para combinar sus archivos fuente dentro de un único perfil XML. LCFG no utiliza **Kickstart** para instalar el ambiente operativo, en su lugar usa su

[35] **SCMSWeb** es un software de monitoreo basado en Web para sistemas *grid* y *clusters* desarrollado por el proyecto SCE.

propio ambiente de *booteo* para configurar el computador, para detectar el hardware, realizar el particionamiento del disco e instalar los RPMs. Tiene soporte para Red Hat 9.0 y Fedora Core 3.0 y la última versión fue liberada en junio del 2005.

2.4.6. XCat

XCat [34] (*eXtreme Cluster Administration Toolkit*) ha sido desarrollado por IBM, utiliza descripciones para crear archivos *kickstart* y posee soporte para **YaST**[36] de SuSe. XCat se puede considerar de código abierto, pero su licencia sólo permite su uso sobre hardware IBM. Las descripciones deben generarse por el administrador del sistema y cada nodo debe tener su propio archivo de instalación. XCat provee una estructura para crear los archivos de descripción, pero se requiere de *scripting* para definir los diferentes tipos de nodos y los recursos.

2.4.7. WAREWULF

Warewulf [35] ha sido desarrollado por el Laboratorio Nacional Lawrence de Berkeley, California; es un conjunto de herramientas para implementar *clusters*. El sistema de archivos de los nodos cliente es administrable desde un sólo punto; además, provee automatización de la distribución del sistema de archivos de los nodos cliente durante la fase de inicialización de los nodos. Incluye las herramientas necesarias para construir archivos de configuración, de monitoreo y de control de los nodos. Es totalmente personalizable y se adapta a cualquier tipo de *cluster*. Sin embargo, *Warewulf* no se considera una solución HPC, se lo considera más un sistema para replicar y administrar sistemas Linux pequeños desde un nodo servidor. Sobre *Warewulf* se pueden distribuir fácilmente paquetes HPC como LAM-MPI/MPICH, SGE, PVM, entre otros. La última versión (Warewulf-2.4.1) fue liberada en abril de 2005.

[36] **YaST** es la herramienta de configuración del sistema operativo Linux SuSe.

2.4.8. openMosix

openMosix [36] ha sido desarrollado por Moshe Bar, es un software que extiende el *kernel* de Linux para proveer una Imagen Única de Sistema. Estas extensiones del *kernel* permiten convertir un conjunto de computadores en un supercomputador Linux. openMosix se instala sobre computadores con sistemas Linux preinstalados. Permite migración de tareas, si un nodo tiene demasiada carga puede migrarla a otro con menos carga. Si se agrega un nuevo nodo, no se requiere configuración alguna, openMosix lo detecta e inmediatamente adapta al *cluster* a la nueva configuración. La última versión (openMosix 2.6) se liberó en abril de 2005.

2.4.9. clusterKNOPPIX

clusterKNOPPIX [37] es una herramienta, para desarrollar *clusters* basados en CD. Los *clusters* basados en CD se construyen cuando se los necesita, simplemente se utiliza un CD para iniciar el sistema. Todo el software que el *cluster* requiere se encuentra en el CD y no se realiza ninguna modificación al contenido del disco duro del equipo en el cual se ejecuta. Cuando ya no se requiere el *cluster*, simplemente se reinicia el computador; es decir, el *cluster* persiste sólo hasta que exista un reinicio del sistema. Se basa en Debian y utiliza el *kernel* de openMosix. Para iniciar a los clientes se utiliza PXE, DHCP y TFTP. Los clientes no requieren iniciarse desde CD, disco duro o disquete, clusterKNOPPIX los autodescubre e integra al *cluster.* La versión más reciente (clusterKNOPPIX 3.6) fue liberada en agosto del 2004.

Existen otras herramientas que no son muy conocidas, entre ellas se pueden citar:

- **Power Cockpit** [38] – Es una herramienta desarrollada por Mountain View Data de Japón; permite convertir un conjunto de equipos en un *cluster* Linux totalmente configurado en minutos.

- **CPlant** [39] – Permite construir un *cluster* utilizando computadores y componentes de red genéricos.

- **Mosix** [40] – Es un software que modifica el *kernel* de Linux agregando funcionalidad para *clusters*. Consigue que todos los nodos aparenten ser un sólo equipo multiprocesador. Contrario a openMosix, Mosix requiere una licencia.

- **BCCD** [41] – Es otra herramienta basada en CD. Fue desarrollado como una herramienta educacional para facilitar la enseñanza de computación paralela.

2.5. REFERENCIAS

[1] MUGLER, John; NAUGHTON, Thomas; SCOTT, Stephen: *OSCAR Clusters.*

[2] http://thin-oscar.ccs.usherbroke.ca/
Último acceso: 15/06/2005

[3] http://opencluster.org/HA-OSCAR/
Último acceso: 15/06/2005

[4] LEANGSUKSUN, Chokchai; SHEN, Lixin; SCOTT, Stephen: *The Modeling and Dependability of High Availability OSCAR Cluster System.*

[5] http://ganglia.sourceforge.net
Último acceso: 15/06/2005

[6] http://hdf.ncsa.uiuc.edu/HDF5
Último acceso: 15/06/2005

[7] http://www.OpenPBS.org
Último acceso: 15/06/2005

[8] http://supercluster.org/maui
Último acceso: 15/06/2005

[9] *NPACI Rocks Cluster Distribution: Users Guide. User's Guide for NPACI Rocks version 3.2.0 Edition*. 2004.
http://rocks.npaci.edu
Último acceso: 15/06/2005

[10] PAPADOPOULOS, Philip; KATZ, Mason; BRUNO, Greg: *NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux Clusters.* 2002.
http://rocks.npaci.edu
Último acceso: 15/06/2005

[11] PAPADOPOULOS, Philip; KATZ, Mason; BRUNO, Greg: *Leveraging Standard Core Technologies to Programmatically Build Linux Cluster Appliances.* 2002.
http://rocks.npaci.edu
Último acceso: 15/06/2005

[12] http://gridengine.sunsource.net/
Último acceso: 15/06/2005

[13] PAPADOPOULOS, Philip; KATZ, Mason; BRUNO, Greg: *Rolls: Modifying a Standard System Installer to Support User-Customizable Cluster Frontend Appliances.* 2004.

http://www.rocksclusters.org
Último acceso: 15/06/2005

[14] http://www.parl.clemson.edu/pvfs
Último acceso: 15/06/2005

[15] www.nsf-middleware.org
Último acceso: 15/06/2005

[16] www.intel.com/software/products/distributors/rocks_cluster.htm
Último acceso: 15/06/2005

[17] http://www.opensce.org
Último acceso: 15/06/2005

[18] www.java.sun.com
Último acceso: 15/06/2005

[19] www.cs.wisc.edu/condor
Último acceso: 15/06/2005

[20] *NPACI Rocks Tutorial - NPACI All Hands Meeting*. 18 de Marzo de 2003.

http://www.rocksclusters.org
Último acceso: 15/06/2005

[21] http://www.centos.org

Último acceso: 15/06/2005

[22] http://www.PBSpro.com
Último acceso: 15/06/2005

[23] SACERDOTI, Federico. *Rocks Clusters.* SUN HPC Consortium. Noviembre 2004.

http://www.rocksclusters.org
Último acceso: 15/06/2005

[24] KATZ, Mason; BRUNO, Greg; PAPADOPOULOS, Philip: *NPACI Rocks Tutorials – Building Clusters.* 2002.

http://www.npaci.edu
Último acceso: 15/06/2005

[25] GAMESS - *General Atomic and Molecular Electronic Structure System.*

http://www.msg.ameslab.gov/GAMESS/GAMESS.html
Último acceso: 15/06/2005

[26] NAMD - *Scalable Molecular Dynamics.*

http://www.ks.uiuc.edu/Research/namd/
Último acceso: 15/06/2005

[27] NWChem - *High Performance Computational Chemistry Software.*

http://www.emsl.pnl.gov:2080/docs/nwchem/nwchem.html
Último acceso: 15/06/2005

[28] AMBER - *Assisted Model Building with Energy Refinement.*
http://www.amber.ucsf.edu/amber/amber.html
Último acceso: 15/06/2005

[29] http://bioinformatics.org/biobrew
Último acceso: 15/06/2005

[30] http://pdswww.rwcp.or.jp/
Último acceso: 11/01/2005

[31] http://www.scyld.com/
http://www.penguincomputing.com
Último acceso: 15/06/2005

[32] http://www.cfengine.org
Último acceso: 15/06/2005

[33] http://www.lcfg.org/
Último acceso: 15/06/2005

[34] http://www.alphaworks.ibm.com/tech/xCAT
Último acceso: 15/06/2005

[35] http://warewulf.lbl.gov/pmwiki/
Último acceso: 15/06/2005

[36] http://www.openmosix.org/
Último acceso: 15/06/2005

[37] http://bofh.be/clusterknoppix/
Último acceso: 15/06/2005

[38] http://www.mountainviewdata.com/us/products/pwc/index.html
Último acceso: 15/06/2005

[39] http://www.cs.sandia.gov/cplant/
Último acceso: 15/06/2005

[40] http://www.mosix.org/
Último acceso: 15/06/2005

[41] http://bccd.cs.uni.edu/
Último acceso: 15/06/2005

CAPÍTULO 3. HERRAMIENTAS DE DESARROLLO PARA APLICACIONES

Este capítulo presenta las ideas básicas involucradas en el desarrollo de aplicaciones paralelas, presentando aspectos relacionados a lenguajes de programación C y C++, que proveen soporte para desarrollo de aplicaciones paralelas a través de software gratuito GNU, y la estandarización del envío de mensajes a través de MPI (*Message Passing Interface*) y PVM (*Parallel Virtual Machine*).

Se presentan diferentes alternativas de implementaciones de librerías de paso de mensajes bajo el estándar MPI Versión 1 y 2: MPICH (MPI *CHameleon*) y LAM (*Local Area Multicomputer*), su arquitectura de software, algunas consideraciones para el diseño de aplicaciones, y un ejemplo de código.

Se mencionan también las librerías para manipulación de vectores y matrices, como BLAS (*Basic Linear Algebra Subprograms*), ScaLAPACK (*Scalable Linear Algebra PACKage*), que junto con MPI y PVM facilitan el desarrollo de aplicaciones.

3.1. HERRAMIENTAS PARA PROGRAMACIÓN PARALELA

El aparecimiento de la computación paralela permitió que emerjan métodos de programación que hicieron posible la implementación de algoritmos utilizando recursos compartidos: CPUs (*Central Processing Unit*), memoria, datos y servicios; dichos métodos a su vez permiten la portabilidad del código fuente a diferentes arquitecturas.

La solución a ciertos problemas computacionales y de cálculos intensivos, es provista por la implementación de tareas paralelas.

El paralelismo se puede implementar mediante una aproximación del modelo cliente–servidor, llamado maestro–esclavo *(master–worker)*. Ésta es fácil de implementar, implica dividir el problema computacional en tareas independientes; es decir, el maestro coordina la solución del problema computacional, asignando tareas independientes a otros procesos, los cuales se denominan procesos esclavos.

Los algoritmos y mecanismos de ejecución en este esquema involucran tareas de paralelismo que contemplan varios pasos:

1. Dividir el problema computacional en tareas independientes.
2. Inicializar los procesos esclavos.
3. Especificar las tareas de comunicación que se realizarán entre el maestro y los esclavos.
4. Emitir los resultados parciales desde los esclavos, y recopilarlos en el maestro (ensamblaje de las soluciones parciales y presentación de la solución final).
5. Asegurar que se recolectaron todos los resultados, y que los procesos esclavos fueron liberados.

Si el trabajo total pudiera ser fácilmente dividido en tareas de tamaño arbitrario, el trabajo de planificación se simplificaría. Si existen ***n*** procesos esclavos, el trabajo debería ser repartido en ***n*** partes, cada una de las cuales debería ser concluida en la misma cantidad de tiempo. A esto se le conoce como planificación estática *(static scheduling)*.

El maestro realiza la asignación inicial de tareas a los esclavos, y luego espera por la finalización del procesamiento de las tareas en los esclavos.

Desde el punto de vista de lenguajes de programación, existen dos alternativas importantes para escribir programas paralelos. Éstas son:

1. El uso de directivas basadas en un lenguaje de programación paralelo.
2. El paso de mensajes explícito mediante llamadas a librerías de lenguajes de programación estándar como: C, C++ y Fortran.

Las herramientas más prominentes de la primera alternativa son HPF[1] (Fortran de Alto Desempeño – *High Performance Fortran*) y OpenMP[2] [2]; en las cuales el código serial se transforma a paralelo, mediante la adición de directivas. Las directivas aparecen como comentarios en el código serial y describen el comportamiento de la aplicación.

En la segunda alternativa, es tarea del programador especificar de manera explícita la división de los datos y las tareas que debe realizar cada proceso. El modelo de paso de mensajes presenta las siguientes características:

- La computación paralela consiste de un número de procesos (locales y/o remotos), cada uno trabajando sobre datos locales. Cada proceso tiene variables locales, y no existe un mecanismo para que algún proceso acceda directamente a la memoria de otro.

- Se logra compartir los datos entre los procesos mediante el envío y recepción de mensajes.

- Este modelo involucra procesos que no necesariamente se ejecutan en diferentes procesadores.

- Este modelo puede ser implementado en distintas plataformas, como multiprocesadores de memoria compartida, redes de estaciones de trabajo y computadores personales de un sólo procesador.

[1] HPF es una extensión al lenguaje Fortran estándar, diseñada para desarrollar aplicaciones paralelas.

[2] OpenMP es un conjunto de compiladores y librerías que permiten escribir programas paralelos con los lenguajes C, C++ y Fortran. Es una librería bastante similar a MPI, constituida como una especificación estándar.

PVM y MPI son especificaciones de librerías de paso de mensajes. Estas administran la transferencia de datos entre instancias de un programa paralelo, las cuales se ejecutan usualmente en múltiples procesadores.

3.1.1. PVM - *PARALLEL VIRTUAL MACHINE*

PVM [5] fue desarrollado a inicios de 1990, por el Laboratorio Nacional Oak Ridge de la Universidad de Tennessee, en conjunto con la Universidad de Emory de Estados Unidos.

La distribución de PVM se obtiene en formatos de fuentes binarias[3] o archivos empaquetados (RPMs) para ambientes UNIX o Linux. Se encuentra disponible para sistemas Windows NT/XP desde la Versión 3.4 como un paquete MSI (MicroSoft *Installer*).

PVM es un conjunto integrado de herramientas de software y librerías que conforman un *framework*[4] de propósito general.

El sistema PVM está compuesto de dos partes: un proceso demonio y de librerías basadas en rutinas. El proceso demonio se denomina **pvmd3.** Cada vez que un usuario ejecuta una aplicación PVM, primero se crea una máquina virtual[5].

Cada usuario puede ejecutar varios programas PVM simultáneamente sobre su propia máquina virtual; además, se pueden configurar varias máquinas virtuales (una por cada nodo del *cluster*) para formar una sola máquina virtual de mejores características.

---

[3] Una descripción de la instalación de paquetes mediante fuentes binarias se presenta en el ANEXO C.

[4] *Framework* es un marco de trabajo por medio del cual se brindan servicios para escribir programas o aplicaciones de manera más sencilla.

[5] La máquina virtual de PVM tiene funcionalidad similar a la ofrecida por la máquina virtual de Java.

La interfaz de la librería PVM contiene las primitivas necesarias para la cooperación entre las tareas de una aplicación. Aquí se definen todas las rutinas para paso de mensajes, sincronización de tareas, creación de procesos, y configuración de la máquina virtual. El sistema PVM actualmente soporta los lenguajes C, C++, y Fortran.

En los lenguajes C y C++, las implementaciones de PVM utilizan una interfaz con funciones basadas en las convenciones del lenguaje C, que permiten acceder a sus diferentes librerías. Los argumentos de estas funciones son una combinación de parámetros y punteros. En el lenguaje Fortran, la funcionalidad de PVM se implementa como subrutinas en lugar de funciones.

3.1.2. OpenMP

OpenMP ofrece un API para escribir programas paralelos para sistemas multiprocesadores, cada uno de ellos con acceso a los mismos recursos de memoria, denominados sistemas de memoria compartida.

OpenMP es un conjunto de librerías para C y C++, regidas por las especificaciones ISO/IEC[6] (*International Standard Organization / International Engineering Consortium*), basado en el uso de directivas[7] para ambientes paralelos.

OpenMP tiene soporte para diferentes sistemas operativos como UNIX, Linux y Windows.

Un programa escrito con OpenMP inicia su ejecución, en un solo hilo activo, llamado maestro. El hilo maestro ejecuta una región de código serial antes de que la primera construcción paralela se ejecute. Bajo el API de OpenMP la construcción paralela se obtiene mediante directivas paralelas. Cuando se

[6] http://www.openmp.org/specs/

[7] Para los lenguajes C y C++ las directivas de OpenMP se encuentran definidas mediante la palabra reservada **#pragma** del prepocesador.

encuentra una región de código paralelo, el hilo maestro crea (*fork*) hilos adicionales, convirtiéndose en el líder del grupo. El hilo maestro y los nuevos hilos ejecutan de forma concurrente la sección paralela (realizan trabajo compartido). Unirse al grupo (*join*) es el procedimiento donde al finalizar la ejecución de la región de código paralelo los hilos adicionales se suspenden o se liberan, y el hilo maestro retoma el control de la ejecución. Este método se conoce como *fork & join*.

La diferencia entre el modelo de paso de mensajes y el modelo de memoria compartida, se debe al número de procesos o hilos activos. En un modelo de paso de mensajes, durante la ejecución del programa, todos los procesos se encuentran activos. Por otra parte, en un modelo de memoria compartida, sólo existe un hilo activo al iniciar y al finalizar el programa; sin embargo, durante la ejecución del programa, la cantidad de hilos activos puede variar de forma dinámica.

En la Figura 3-1 se indica como un hilo maestro encuentra una sección de código paralelo y crea hilos adicionales para ejecutar dicha sección. Una vez realizadas las tareas de ejecución, el hilo maestro retoma el control del programa.

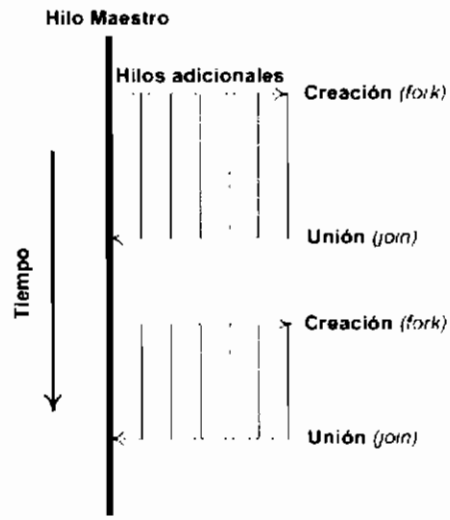


Figura 3-1. Modelo de ejecución *fork & join*

3.1.3. MPI – *MESSAGE PASSING INTERFACE*

La estandarización de MPI empezó en Abril de 1992. La primera versión del estándar MPI aparece en Mayo de 1994. La Versión 1.1 agregó algunas aclaraciones y refinamientos y fue publicada a mediados de 1995. Las Versiones 1.0 y 1.1 fueron diseñadas para los lenguajes C y Fortran 77.

En Marzo de 1995, se extendió la versión original, y culminó con la creación del estándar MPI Versión 2. La Versión 2 incluye la implementación para C++ y Fortran 90.

MPI no es un lenguaje de programación, es un conjunto de funciones y macros que conforman una librería estándar de C y C++, y subrutinas en Fortran.

MPI ofrece un API, junto con especificaciones de sintaxis y semántica que explican como sus funcionalidades deben añadirse en cada implementación que se realice (tal como almacenamiento de mensajes o requerimientos para entrega de mensajes). MPI incluye operaciones punto a punto y colectivas, todas destinadas a un grupo específico de procesos.

MPI realiza la conversión de datos heterogéneos como parte transparente de sus servicios, por medio de la definición de tipos de datos específicos para todas las operaciones de comunicación. Se pueden tener tipos de datos definidos por el usuario o primitivos.

3.2. FUNDAMENTOS DE MPI

El método de programación comúnmente utilizado para sistemas de memoria distribuida es el paso de mensajes. En un método básico de paso de mensajes, los procesos coordinan sus actividades mediante mensajes de envío y recepción.

MPI asume que los procesos se ejecutan localmente (creados bajo el espacio de memoria del usuario), el número de procesos requeridos se asignan antes de la ejecución del programa, y no se crean procesos adicionales mientras la aplicación se ejecuta.

A cada proceso se le asigna una variable que se denomina *rank*, la cual identifica a cada proceso, en el rango de **0** a **p-1**, donde **p** es el número total de procesos.

El método de programación basado en paso de mensajes utiliza el paradigma SPMD[8] (*Single Program / Multiple Data*). En un programa SPMD, el control de flujo se realiza mediante la variable *rank*, es decir que mediante la variable *rank* se determina el proceso que ejecuta determinada porción de código.

MPI define un *comunicator* como una colección de procesos, los cuales pueden enviar mensajes el uno al otro; en lenguajes como C, el *comunicator* se representa mediante un macro especial de la especificación MPI. Por ejemplo, el *comunicator* básico **MPI_COMM_WORLD**, definido por un macro del lenguaje C, agrupa a todos los procesos activos durante la ejecución de una aplicación.

3.2.1. MENSAJES EN MPI

Un mensaje consiste del cuerpo del mensaje, el cual contiene los datos a ser enviados, y de la envoltura, que indica el proceso fuente y el destino.

El cuerpo del mensaje en MPI se conforma por tres piezas de información:

- El *buffer*, es la localidad de memoria donde se encuentran los datos de salida o donde se almacenan los datos de entrada.

[8] SPMD es un paradigma de la computación paralela y ha sido utilizado como patrón de diseño para la implementación de aplicaciones seriales y paralelas. SPMD se utiliza en una gran cantidad de simulaciones científicas; en este paradigma, se ejecuta un programa idéntico en cada proceso o procesador.

- Tipo de dato, indica el tipo de los datos que se envían en el mensaje. En casos simples, éste es un tipo básico o primitivo, por ejemplo, un número real tipo *float*[9], y que en aplicaciones más avanzadas puede ser un tipo de dato construido a través de datos primitivos. Los tipos de datos derivados son análogos a las estructuras de C o a las clases de C++.

- El contador o *count* es un número de secuencia que junto al tipo de datos permiten al usuario agrupar ítems de datos de un mismo tipo en un solo mensaje. MPI estandariza los tipos de datos primitivos, evitando que el programador se preocupe de las diferencias que existen entre ellos, cuando se encuentran en distintas plataformas.

La envoltura de un mensaje en MPI es similar a la envoltura de papel de una carta. La envoltura típicamente contiene la dirección destino, la dirección de la fuente, y cualquier otra información que se necesite para transmitir y entregar la carta.

La envoltura de un mensaje en MPI, consta de cuatro partes:

- La fuente; es decir, el proceso transmisor.
- El destino; es decir, el proceso receptor.
- El *comunicator*, que especifica el grupo de procesos a los cuales pertenecen la fuente y el destino.
- Una etiqueta *(tag)*, que se utiliza para clasificar el mensaje.

El campo etiqueta es un entero definido por el usuario que puede ser utilizado para distinguir los mensajes que recibe un proceso. Por ejemplo, se tienen dos procesos A y B. El proceso A envía dos mensajes al proceso B, ambos mensajes contienen un dato. Uno de los datos es utilizado para realizar un cálculo, mientras el otro es utilizado para imprimirlo en pantalla. El proceso A utiliza diferentes etiquetas para los mensajes. El proceso B utiliza los valores de etiquetas definidos en el proceso A e identifica que operación deberá realizar con el dato de cada

[9] *float* es un tipo de dato primitivo utilizado en varios lenguajes de programación. Este es un dato de punto flotante, que representa un número decimal.

mensaje. Una etiqueta consiste de un número entero de 32 bits; se pueden utilizar valores desde 0 a 32767.

Los programas escritos con este modelo de paso de mensajes, consisten de múltiples instancias de programación serial que se comunican mediante llamadas a librerías.

En la Figura 3-2 se muestra un mensaje típico de MPI.

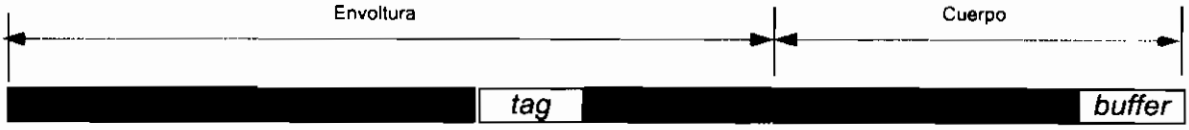


Figura 3-2. Formato de un mensaje de MPI

Las llamadas (que pueden ser funciones o subrutinas) a librerías se dividen en cuatro clases:

1. Llamadas utilizadas para inicializar, administrar, y finalizar comunicaciones.
2. Llamadas utilizadas para la comunicación entre un par de procesos.
3. Llamadas que desempeñan operaciones de comunicación entre una gran cantidad de procesos.
4. Llamadas utilizadas para crear tipos de datos definidos por el usuario.

La primera clase de llamadas permiten inicializar la librería de paso de mensajes, identificar el número de procesos *(size)* y el rango de los procesos *(rank)*. La segunda clase de llamadas, incluye operaciones de comunicación punto a punto, para diferentes tipos de actividades de envío y recepción. La tercera clase de llamadas son conocidas como operaciones grupales, que proveen operaciones de comunicaciones entre grupos de procesos. La última clase de llamadas provee flexibilidad en la construcción de estructuras de datos complejos.

3.2.2. MODOS DE COMUNICACIÓN EN MPI

MPI provee una gran flexibilidad para especificar los mensajes a enviarse. Existe una variedad de modos de comunicación que define el procedimiento utilizado para transmitir el mensaje; además, se especifica el criterio para determinar cuando un evento de comunicaciones completó su operación. Por ejemplo, una llamada de envío sincrónica finaliza cuando el mensaje en el destino fue recibido y se confirmó su recepción. Una llamada de envío tipo *buffer*, finaliza cuando los datos de salida se copian en un *buffer* local (internamente creado por MPI); y no es primordial que el mensaje haya llegado a su destino.

Existen cuatro tipos de modos de comunicación disponibles para el envío de mensajes, éstos son:

- Sincrónico. Una versión de llamada de envío se realiza en el proceso transmisor, la cual no culmina una transmisión hasta que una llamada de recepción haya sido invocada en el proceso receptor, y además la recepción de los datos enviados por el proceso trasmisor haya iniciado.

- De tipo *buffer*. La llamada de envío de un proceso transmisor se realiza de forma local. En este modo la culminación de un envío no depende de la existencia de una llamada de recepción en el proceso receptor. En otras palabras el mensaje se copia en un *buffer* creado con ayuda de MPI. Se debe conocer el tamaño de los datos para poder utilizar este tipo de modo.

- Listo (*ready*). Este modo se utiliza si el proceso transmisor puede saber con certeza que el proceso receptor realizó una llamada de recepción, antes de que el proceso transmisor haya realizado un llamada de envió.

- Estándar. La implementación de MPI decide si los mensajes deben ser almacenados en un *buffer* o si deben enviarse en modo sincrónico. Es funcional solo para mensajes pequeños (el tamaño de los mensajes depende de la implementación de MPI).

Para llamadas de recepción, existe solamente un modo de comunicación. Una recepción finaliza cuando los datos entrantes han llegado y están disponibles para su uso.

3.2.3. LLAMADAS BLOQUEANTES Y NO BLOQUEANTES

Además del modo de comunicación utilizado, un envío o recepción puede ser bloqueante o no bloqueante. Un envío o recepción bloqueante no presenta el resultado (valor de retorno) de la llamada antes de que la operación finalice.

Una llamada no bloqueante retorna[10] inmediatamente. La ventaja es que el programa está libre para realizar otras tareas mientras la comunicación se está ejecutando en el *background*[11].

3.2.4. COMUNICACIÓN CON ESTRUCTURA TIPO ÁRBOL

Se puede representar a la distribución y división del trabajo computacional entre los procesos mediante un árbol de procesos, en el cual se designa el identificador 0 al proceso raíz (proceso principal del árbol).

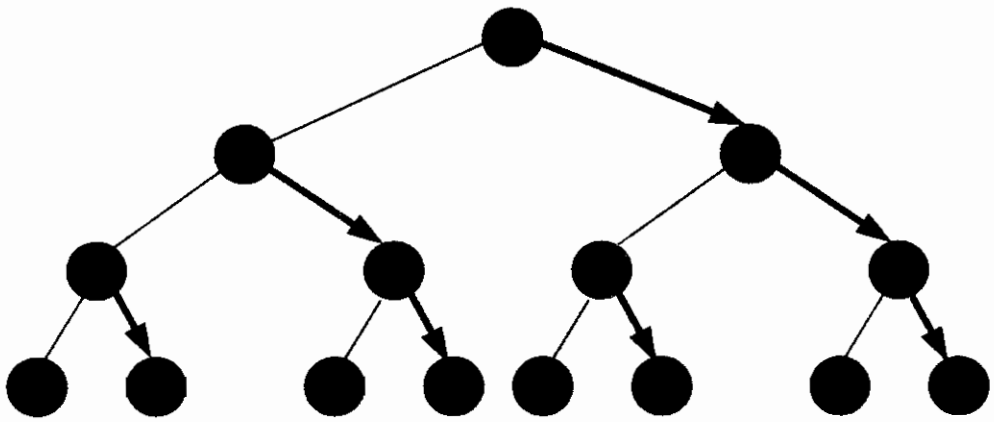
En la Figura 3-3 se muestra la estrategia de distribución de los datos entre 7 procesos. Las circunferencias representan los procesos, y las flechas indican el camino de distribución.

Durante la primera etapa de la distribución, el proceso 0 envía los datos a 1. En la siguiente etapa, 0 envía los datos al proceso 2, mientras 1 envía los datos a 3. Durante la última etapa, 0 envía los datos a 4, mientras 1 envía los datos a 5, 2 envía los datos a 6, y 3 envía los datos a 7. De esta forma se reduce la

[10] La llamada no bloqueante retorna un código de error.

[11] En general se habla de *background* para indicar que algunos procesos que se crean no están definidos para servir a un usuario en particular. Un proceso *foreground* implica que un proceso fue creado para brindar servicios a un usuario en particular.

distribución de los datos, de siete etapas a tres etapas. Generalizando este esquema, si se tienen **p** procesos, se podrán obtener $\lceil \log_2(p) \rceil$[12] etapas de envío de datos, entre los procesos que conforman el árbol [1]. Por ejemplo, $\lceil \log_2(8) \rceil$ = 3. Es decir si se utiliza una estructura de árbol, conformada por 8 procesos, el tiempo requerido por la aplicación para distribuir los datos será reducido por un factor cercano a 3. Con p = 1000 la reducción estaría en el orden de 100.



Fuente: [1], Capítulo 5, página 66

Figura 3-3. Estructura tipo árbol

3.2.5. COMUNICACIONES PUNTO A PUNTO

La forma de comunicación predefinida por la librería de MPI se realiza mediante llamadas punto a punto.

Las llamadas punto a punto, o extremo a extremo, requieren la participación activa de los procesos en ambos lados. Un proceso (la fuente) que envíe, y otro proceso (destino) que reciba.

En general, la fuente y el destino operan de la siguiente manera: el proceso fuente completa la transmisión del mensaje instantes antes de que el destino haya terminado de recibirlo por completo, y el proceso destino recibe el mensaje instantes después de que la transmisión del mensaje haya finalizado. Estos

[12] El valor *ceiling* de un número real x es el número entero inmediato mayor o igual que x. El *ceiling* se denota por $\lceil x \rceil$.

mensajes que han sido enviados, pero aún no han sido recibidos por completo se los conoce como mensajes pendientes.

Una característica importante de MPI es que los mensajes pendientes no son almacenados en una cola FIFO. MPI asigna atributos para cada mensaje pendiente, y el proceso destino puede utilizar dichos atributos para determinar cual mensaje será recibido. Es decir, el proceso receptor conoce los detalles de cada uno de los mensajes pendientes mediante los atributos.

Las llamadas básicas de comunicaciones punto a punto en MPI se realizan mediante las funciones: **MPI_SEND** y **MPI_RECV**. Ambas funciones son bloqueantes, es decir que el proceso que realiza la llamada se bloquea hasta que la operación de comunicación se complete.

El mensaje de una llamada **MPI_SEND** o **MPI_RECV** contiene: el cuerpo y una envoltura del mensaje. Adicionalmente, la llamada devuelve un código que indica su éxito o fracaso.

MPI_SEND por defecto utiliza un modo de comunicación estándar, es decir, que la implementación de MPI se encarga de seleccionar entre un modo tipo *buffer* o un modo sincrónico para la transmisión del mensaje.

En el modo de comunicación sincrónico el envío y la recepción podrían causar lo que se conoce como interbloqueo (*deadlock*). Un interbloqueo ocurre cuando 2 o más procesos están bloqueados y cada uno espera que el otro se desbloquee para realizar alguna operación. Los procesos no pueden realizar procesamiento debido a que dependen de los otros para poder realizar nuevas tareas.

Para prevenir el *deadlock* y el retardo, MPI provee otra forma de invocar operaciones de envío y recepción. Es posible separar el inicio de la transmisión o recepción realizando dos llamadas separadas a MPI. La primera llamada inicia la operación, y la segunda llamada completa dicha operación. Entre las dos

llamadas, el programa es libre de realizar cualquier otra actividad. A este tipo de operaciones se las conoce como asíncronas.

MPI define estructuras de datos internos relacionadas a las comunicaciones y otras operaciones. Para referirse a estas estructuras de datos se utilizan los manipuladores (*handles*). Los manipuladores son valores de retorno de varias funciones y también pueden ser utilizados como argumentos de llamadas de MPI. En C y C++ los manipuladores son punteros a tipos de datos definidos por el usuario, los cuales son creados mediante *typedef* [13].

Un proceso llama a la función **MPI_ISEND** para iniciar una transmisión no bloqueante sobre una operación de envío. Esta llamada es similar a la realizada por la llamada de **MPI_SEND,** e incluye un argumento de salida adicional, denominado manipulador de petición. Este manipulador identifica la operación de envío realizada por la llamada **MPI_ISEND**, y permite chequear el estado de la operación de envío.

Un proceso llama a la función **MPI_IRECV** para iniciar una operación de recepción no bloqueante. Esta función dispone de un argumento adicional, denominado manipulador de petición, el cual identifica la operación de recepción realizada por la llamada **MPI_IRECV**; además, permite verificar el estado de la operación de recepción.

El estado de las operaciones de envío o recepción realizadas por las llamadas **MPI_ISEND** o **MPI_IRECV,** respectivamente, necesita ser comprobado mediante una llamada de la familia de funciones de finalización, como **MPI_TEST** y **MPI_WAIT**.

Tanto **MPI_TEST** como **MPI_WAIT** reciben como argumento el manipulador de petición que identifica la operación correspondiente. **MPI_WAIT** es una llamada bloqueante y retorna cuando la operación de envío o recepción se completa.

[13] La declaración *typedef* sirve para asignar un sinónimo a un tipo dado.

MPI_TEST permite verificar si la operación de envío o recepción ha finalizado, esta función primero chequea el estado de la operación de envío o recepción y luego retorna.

En los sistemas donde la latencia es grande, el posponer la recepción es con frecuencia una estrategia efectiva para enmascarar el *overhead*[14] en las comunicaciones.

3.2.6. COMUNICACIONES COLECTIVAS

Adicionalmente a las comunicaciones punto a punto entre pares de procesos, MPI incluye funciones para realizar comunicación grupal. Estas funciones permiten a un grupo de procesos comunicarse de varias maneras, por ejemplo, uno a varios o varios a uno.

Las principales ventajas de utilizar estas rutinas sobre el equivalente de comunicaciones punto a punto son:

- La posibilidad de error se reduce significativamente. Una línea de código (mediante una llamada grupal) reemplaza varias llamadas punto a punto.

- El código fuente es sencillo, lo cual facilita la depuración y el mantenimiento.

MPI posee comunicaciones grupales que incluyen operaciones tipo difusión (*broadcast*), recolección (*gather*), distribución (*scatter*) y reducción. Las llamadas que MPI provee para este efecto son:

- Barreras de sincronización.
- Difusión desde un proceso a todos los procesos.

[14] El término sobrecarga (overhead) se utiliza para denotar el exceso dentro de una transmisión; por ejemplo, se considera sobrecarga a todo aquello que dentro de un paquete IP no constituye datos, como las cabeceras IP, pero que son esenciales para la comunicación extremo - extremo.

- Operación global de reducción.
- Recolección desde todos los procesos a sólo uno.
- Distribución[15] desde un proceso a los otros procesos.
- Operación avanzada donde todos los procesos reciben el mismo resultado desde una recolección, distribución o reducción.

3.2.6.1. Operaciones con barreras de sincronización

En estas operaciones no existe ninguna clase de intercambio de información. Es una operación puramente de sincronización, que bloquea a los procesos de un comunicador hasta que todos ellos han pasado por la barrera. Suele emplearse para dar por finalizada una etapa del programa, asegurándose de que todos han terminado antes de dar comienzo a la siguiente.

MPI define la función **MPI_BARRIER** para operaciones con barreras de sincronización.

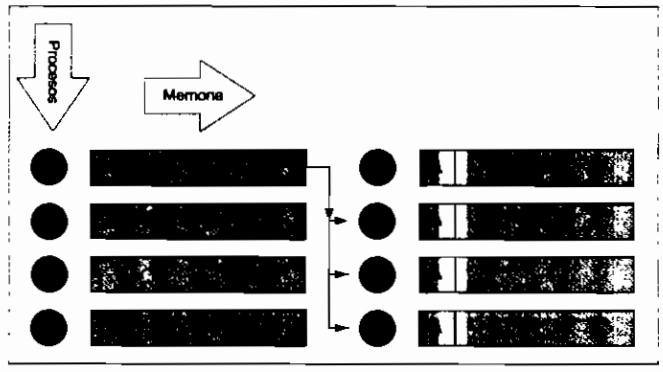
3.2.6.2. Operaciones de difusión

Permite a un proceso enviar una copia de sus datos a otros procesos dentro de un grupo definido por un comunicador.

En la Figura 3-4 se muestra la transferencia de información entre diferentes procesos usando llamadas de difusión. Los círculos junto con las celdas representan un proceso con su espacio de memoria. A la izquierda se muestra el proceso 0 que contiene la letra A almacenada en la primera dirección de su memoria. Instantes después, el manipulador que contiene el valor de A, es transmitido mediante un mensaje de difusión a los procesos involucrados en la comunicación.

[15] La diferencia entre las operaciones de Distribución y Difusión se presenta en los siguientes párrafos.

Luego de la operación, cada proceso miembro del grupo definido por el *comunicator* (proceso 0, 1, 2 y 3) posee una copia del valor de A.



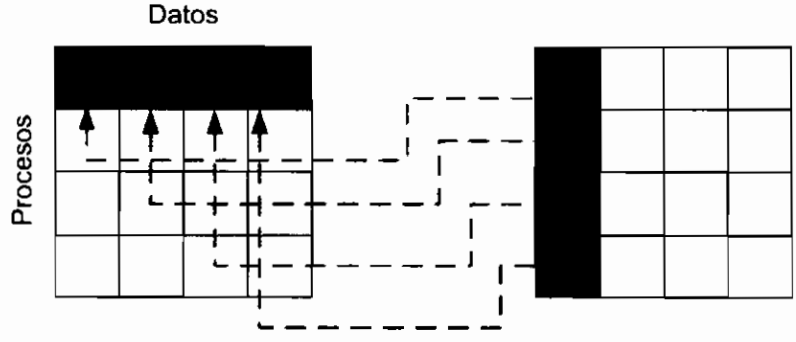
Fuente: [3], Capítulo 6, página 68

Figura 3-4. Operación tipo Difusión

MPI define la función **MPI_BCAST** para utilizar la operación de difusión.

3.2.6.3. Operaciones tipo recolección y distribución

En las operaciones tipo recolección, los datos (un arreglo de algún tipo) son recolectados en un sólo proceso. Es decir que los datos que se encuentran en localidades de memoria similares en procesos diferentes se copian en localidades de memoria contiguas en el proceso raíz. En la Figura 3-5 se muestra la operación de recolección.



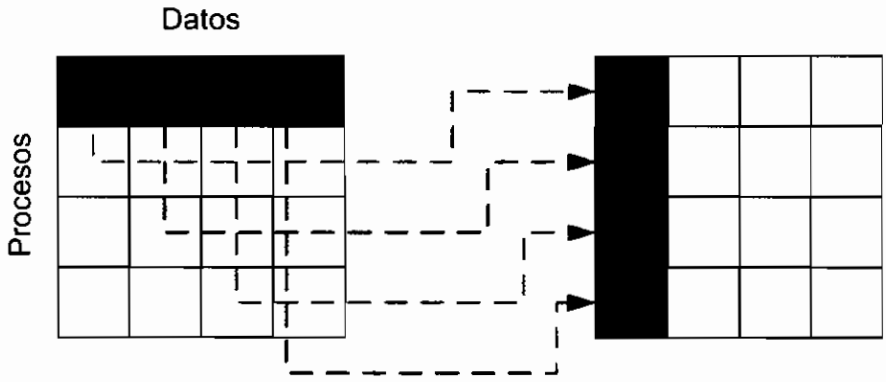
Fuente: [3], Capítulo 2, página 15

Figura 3-5. Operación tipo Recolección

MPI define la función **MPI_GATHER** para la operación de recolección y **MPI_SCATTER** para la operación de distribución.

Después de la operación de distribución, los datos se distribuyen en diferentes procesos, por lo que se dice que las operaciones de recolección y distribución van de la mano.

En la Figura 3-6 se muestra la operación de distribución, la cual distribuye los datos en múltiples procesos; se puede ver que existen datos que no pueden ser separados y deben ser enviados en conjunto a un solo proceso (cuadro con dos tonalidades).

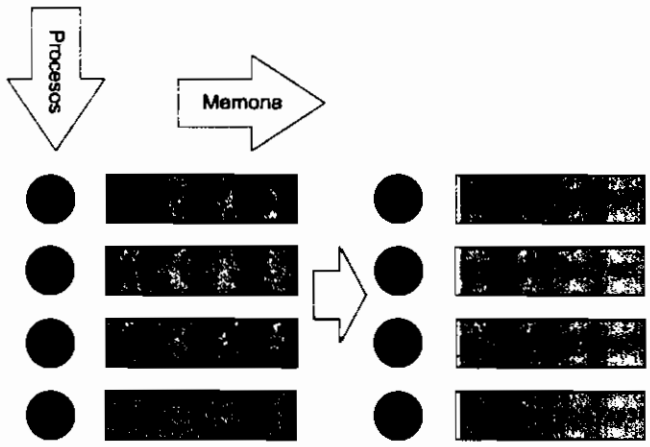


Fuente: [3], Capítulo 2, página 15

Figura 3-6. Operación tipo Distribución

Después de que los datos se copian en el proceso raíz, se puede realizar una operación MPI de difusión a todos los procesos enviando el resultado de la operación anterior; de esta forma, todos los procesos tendrán la misma información y pueden realizar un cálculo global; se puede realizar una llamada a una función que involucre las operaciones de recolección y de difusión. MPI define la función **MPI_ALLGATHER** para este efecto.

En la Figura 3-7 se muestra la transferencia de datos entre procesos usando una llamada a **MPI_ALLGATHER**, y se puede apreciar que cada proceso posee su propio dato antes de realizar las operaciones de recolección y difusión. En la operación de difusión cada proceso envía la información que éste posee. El proceso raíz no está involucrado en la operación de difusión.



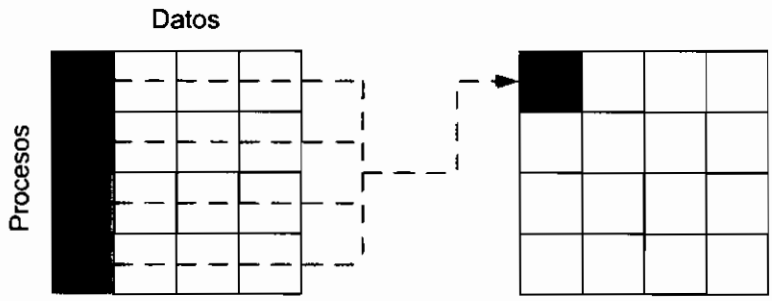
Fuente: [3], Capítulo 6, página 75

Figura 3-7. Operaciones tipo Recolección - Difusión

3.2.6.4. Operaciones de reducción

En las operaciones de reducción, el proceso raíz colecta datos desde otros procesos en un grupo, y los combina en un solo ítem de datos. Por ejemplo, se podría utilizar una operación reducción, para calcular la suma de los elementos de un arreglo que se distribuyó en algunos procesos. La Figura 3-8 muestra como los datos son colectados en un sólo proceso.

Es posible realizar otras operaciones aritméticas; por ejemplo, realizar operaciones lógicas y de bits.



Fuente: [3], Capítulo 2, página 16

Figura 3-8. Operación tipo Reducción

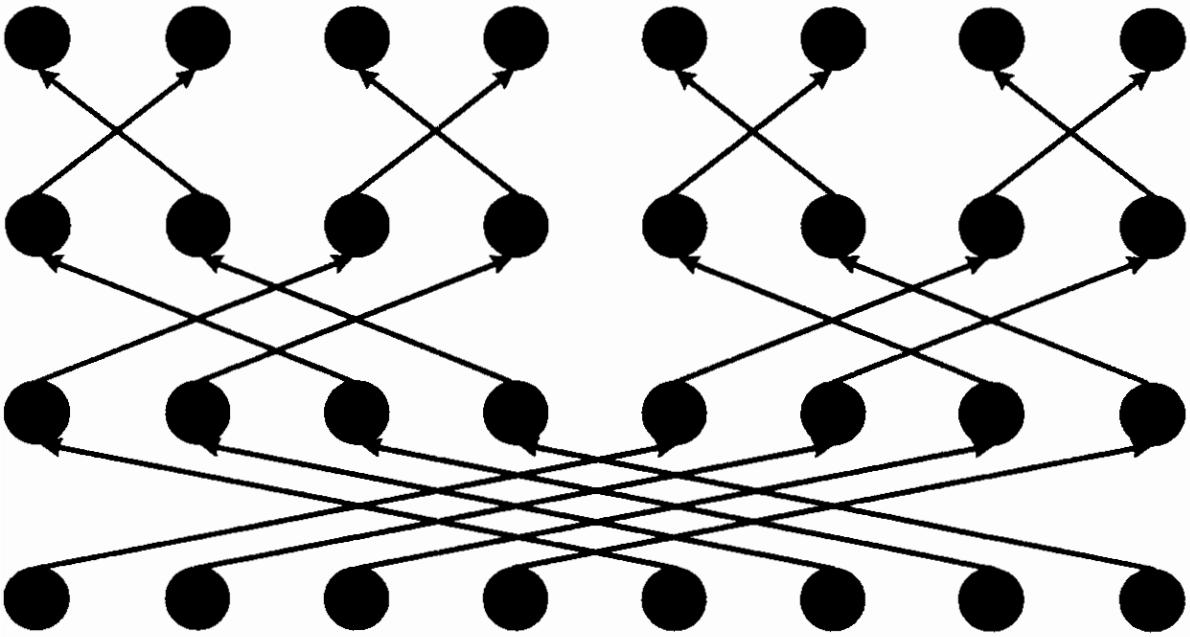
MPI define la función **MPI_REDUCE** para realizar una operación de reducción.

Al finalizar una tarea en otros procesos, la llamada a una operación de reducción devuelve el valor 0, este valor indica que la tarea realizada ha sido exitosa. Por

ejemplo, si se desea imprimir el resultado de un cálculo; los procesos (a excepción del proceso raíz) imprimen el valor cero en lugar del resultado parcial o total. Y si se desea utilizar el resultado en cálculos adicionales, cada operación en otros procesos deberá retornar el valor correspondiente a dicho cálculo.

Los procesos pueden obtener el valor de un cálculo utilizando una operación de reducción para recolectar los datos; luego mediante una operación de difusión enviar el resultado a los procesos que pertenecen al *comunicator*.

En la Figura 3-9, se tienen 8 procesos en los cuales se desea realizar un cálculo (por ejemplo, una suma) y luego almacenar los resultados en los procesos.



Fuente: [1], Capítulo 5, página 77

Figura 3-9. Estructura de árbol tipo mariposa

Los procesos 0 y 4, 1 y 5, 2 y 6, y los procesos 3 y 7, intercambian sus resultados locales. Cada proceso añade su resultado al resultado recibido, conformando una estructura de árbol denominado mariposa (*butterfly*). Si se añaden líneas verticales entre los procesos involucrados con un mismo rango en filas adyacentes, se puede visualizar que cada proceso se convierte en proceso raíz.

MPI define la función **MPI_ALLREDUCE** para realizar una operación tipo mariposa.

3.2.7. MENSAJES Y DATOS EN MPI

MPI provee tres mecanismos para agrupar datos individuales en un solo mensaje. El primero involucra el uso del parámetro *count* y varias operaciones de comunicaciones. En este mecanismo se utilizan llamadas de envío, recepción, difusión y reducción, donde cada una de ellas contiene un parámetro *count* y un parámetro *datatype*. Estos parámetros permiten al usuario agrupar datos en un solo mensaje utilizando el mismo tipo de datos primitivo.

El segundo mecanismo involucra tipos de datos derivados, o construidos por el usuario. Un tipo de datos derivado en MPI es una estructura, que se construye durante la ejecución del programa y puede pasarse como argumento a las funciones de comunicaciones de MPI.

Para construir un tipo de datos derivado, se debe especificar:

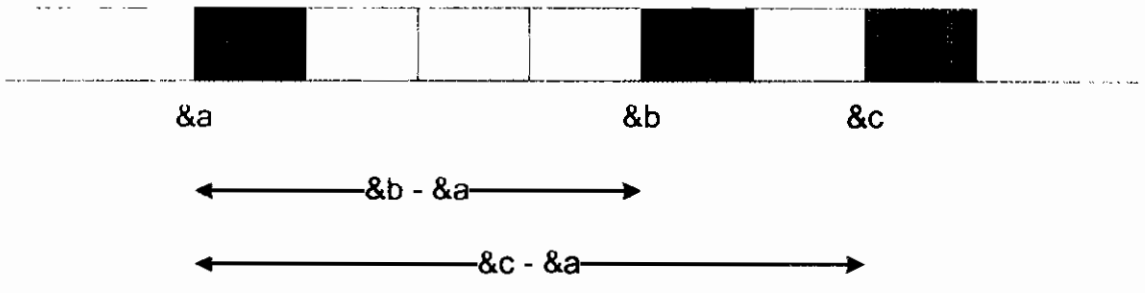
- El número de elementos en el tipo de datos.
- Los tipos de los elementos que conforman el nuevo tipo.
- Los desplazamientos relativos, o *displacements*, de los elementos en la memoria.

Por ejemplo, se desea transmitir un mensaje que contiene 3 tipos de datos diferentes: tipo A, tipo B, y tipo C. Cada uno tiene direcciones de memoria &a, &b y &c, que contienen valores a, b y c, respectivamente.

Para poder transmitir estos datos se necesita especificar:

- Que existen 3 elementos a transmitirse.
- Que el primer elemento es de tipo A, el segundo tipo B y el tercero tipo C.
- Que la dirección del primer elemento es &a, del segundo es &b y del tercero es &c.

Dado este problema, se puede calcular el desplazamiento relativo de memoria de b y c desde a, a partir de la dirección de memoria &a. Es decir, que el desplazamiento de memoria de b es (&b - &a) bytes a continuación de a, y (&c - &a) bytes para c a continuación de a. En la Figura 3-10, se muestran los desplazamientos relativos que se calculan a partir de &a.



Fuente: [1], Capítulo 6, página 92

Figura 3-10. Diagrama de desplazamientos relativos

Con mayor formalidad, un tipo de datos derivado en MPI es una secuencia de pares (*tuplas*):

$$\{(t_0, d_0), (t_1, d_1), \ldots, (t_{n-1}, d_{n-1}), (t_n, d_n)\}$$

Donde *ti* es un tipo de datos primitivo de MPI y cada *di* es un desplazamiento en bytes. Entre los tipos de datos primitivos en MPI se tienen: **MPI_INT**, **MPI_CHAR**, **MPI_FLOAT**.

Para mensajes más complejos, se pueden construir tipos de datos derivados, o utilizar el tercer mecanismo, que mediante la utilización de las funciones: **MPI_PACK** y **MPI_UNPACK** permite empaquetar y desempaquetar los datos. La primera función permite especificar datos que se almacenan de manera no contigua en localidades de memoria, y está constituida por un parámetro especial que indica los tipos de datos a ser desempacados mediante la función **MPI_UNPACK**.

3.2.8. COMUNICADORES

El uso de comunicadores (*comunicators*) hace que MPI sea diferente de otros sistemas de paso de mensajes. El mecanismo que MPI provee para trabajar con un subconjunto de procesos perteneciente al universo de comunicaciones se conoce como comunicador.

MPI permite definir comunicadores específicos acordes a nuevas necesidades y que se acoplen a las aplicaciones.

En MPI existen dos tipos de comunicadores: los intra-comunicadores y los inter-comunicadores. Los intra-comunicadores son esencialmente una colección de procesos que pueden enviar mensajes el uno al otro y acoplarse a operaciones de comunicación colectivas. Los inter–comunicadores, están fuera del alcance de este documento, y éstos se utilizan para enviar mensajes entre procesos pertenecientes a intra–comunicadores no comunes.

Un intra-comunicador se conforma por: el grupo y el contexto. Un grupo es una colección de procesos ordenados. Si un grupo consiste de *p* procesos, a cada proceso en el grupo se le asigna un *rank*. El contexto, es un sistema de etiquetas definido para identificar de manera única a un comunicador. Es decir que dos comunicadores distintos presentarían diferentes contextos; sin embargo, éstos pueden estar asignados a un mismo grupo.

3.2.9. IMPLEMENTACIONES DE MPI

Se han creado varias implementaciones de MPI basadas en la publicación del estándar, muchas de ellas son de libre distribución y algunas tienen limitaciones de portabilidad de código. Algunas implementaciones de MPI se las puede encontrar en la página Web:

http://www-unix.mcs.anl.gov/mpi/implementations.html

A continuación se describen algunas implementaciones:

- LAM/MPI (*Local Area Multicomputer*), fue diseñada en el Centro de Supercomputadores de la Universidad de Ohio. Puede ejecutarse sobre redes heterogéneas de equipos SUN, DEC, IBM, de estaciones de trabajo y computadores personales. Se puede descargar una implementación gratuita de la página Web:

  http://www.lam-mpi.org/

- MPICH, desarrollado a la par del estándar MPI. Se tienen varias implementaciones, y sus primeras versiones difieren de las más actuales, ya que fueron diseñadas para estaciones de trabajo y de computadores personales, donde el desempeño de software estaba limitado por la funcionalidad de *sockets* de Unix. Se puede descargar una implementación gratuita de la página Web:

  http://www-unix.mcs.anl.gov/mpi/mpich/download.html

- Unify, provisto por la Universidad Estatal de Mississippi, en ésta se recopilan capas de software de MPI sobre una versión de PVM. Unify permite incluir llamadas de MPI y PVM dentro de un mismo programa.

3.2.9.1. LAM/MPI

LAM/MPI es una implementación de alto desempeño, y de código abierto del estándar MPI, desarrollado y administrado por los Laboratorios Open Systems de la Universidad de Indiana.

LAM/MPI no es solo una librería MPI, contiene un ambiente de ejecución, que está definido para el espacio de usuario y administrado por un demonio que provee los servicios requeridos por los programas de MPI.

LAM/MPI ha sido desarrollada para diferentes plataformas como NT, y viene incluida con algunas distribuciones Linux (por ejemplo, Red Hat 9.0).

El componente principal de LAM/MPI es el marco de trabajo (*framework*) denominado SSI (*System Service Interface*). Las funcionalidades de SSI pueden ser extendidas mediante la instalación y configuración de pequeños módulos que pueden seleccionarse en tiempo de ejecución.

El SSI es el corazón de LAM/MPI, define la manera en la que se ejecutan varios comandos y procesos MPI.

Existen cuatro componentes utilizados por LAM/MPI:

1. *boot*: Este componente permite la inicialización del ambiente LAM en tiempo de ejecución; es utilizado mediante el comando **lamboot**.

2. *coll*: Este componente manipula las comunicaciones colectivas de MPI; utilizados únicamente por los procesos MPI.

3. *cr*: (Checkpoint/Restart) Permite incluir puntos de comprobación y reinicialización; utilizados por los comandos de MPI y los procesos MPI.

4. *rpi*: Manipula las comunicaciones MPI punto a punto; utilizadas únicamente por los procesos MPI.

3.2.9.2. MPICH

La implementación de MPI conocida como MPICH es una de las versiones más populares de las librerías MPI. La primera versión de MPICH fue presentada pocos días después de que la especificación MPI Versión 1 fue publicada (en el año 1992). La intención era mostrar la funcionalidad que proveía mediante el paso de mensajes. Sin embargo, la versión que permitió tener toda la funcionalidad de la especificación MPI Versión 1 se desarrollo en 1998.

La nueva versión se conoce como MPICH2; ésta incluye toda la funcionalidad de los estándares MPI Versión 1 y 2.

MPICH ha sido desarrollado para diferentes plataformas como Linux, UNIX, Windows NT y otras.

La versión actual de MPICH2 está disponible en la página Web: http://www.mcs.al.gov/mpi/mpich. Esta versión contiene:

- El código fuente de MPICH2.
- Los *scripts* de configuración para la construcción de MPICH2 sobre varios sistemas (incluyendo *clusters* bajo sistemas Linux).
- Varios programas MPI de ejemplo.
- El sistema de administración de procesos paralelos conocido como MPD (MPICH *Process Daemon*).

3.2.9.2.1. Arquitectura de software MPICH

El diseño de la arquitectura de software de MPICH se basó en dos principios fundamentales: el primero, el deseo de maximizar la cantidad de código que puede ser compartido sin comprometer el desempeño; es decir, que una gran cantidad de código en cualquier implementación sea independiente de la plataforma; el segundo, se basa en el deseo de proveer una estructura donde MPICH pueda transportarse rápidamente a una nueva plataforma, y luego gradualmente se adapte a dicha arquitectura, mediante el reemplazo de partes de código compartido por código específico de la plataforma.

El mecanismo principal para conseguir los objetivos de portabilidad y desempeño se logra a través de una especificación. Esta especificación se conoce con el nombre de ADI (*Abstract Device Interface*), donde residen todos los macros y las funciones implementados a través de los lenguajes C o C++.

Una de las implementaciones de ADI se denomina interfaz de canal (*channel interface*). La interfaz de canal provee un acceso rápido para transportar código de MPICH a un nuevo ambiente, a través de varios mecanismos de comunicación que permiten el soporte para *sockets* y memoria compartida. Usualmente la

interfaz de canal se denomina **ch3**, debido a que es la tercera versión de la implementación de la interfaz.

3.2.9.2.2. Instalación y Administración de MPICH

MPICH puede instalarse sobre un sistema *cluster* basado en Linux. La distribución de MPICH con frecuencia se encuentra como archivos fuente. A continuación se muestran una serie de comandos que permiten descomprimir e instalar MPICH.

Instalación a partir de una distribución basada en archivos fuente:

```
[src]# tar -zxvf mpich2-1.0.tar.gz
[src]# cd mpich2-1.0
```

Iniciando la configuración y definiendo el *path* de instalación.

```
[mpich2-1.0]# ./configure --prefix=/usr/local/mpich2-1.0
[mpich2-1.0]# make
[mpich2-1.0]# make install
```

Configuración de las variables de entorno a través del comando **set**:

```
[mpich2-1.0]# set PATH /usr/local/mpich2-1.0/bin:$PATH
```

Inicialización del demonio **mpd** en el *background*:

```
[mpich2-1.0]# mpd -d &
```

La compilación de programas se realiza mediante los comandos **mpicc** o **mpicxx**, para los lenguajes C y C++, respectivamente.

Bajo el lenguaje C:

Para compilar:

```
# mpicc -c nombre_archivo.c
```

Para enlazar y crear el ejecutable:

```
# mpicc -o nombre_archivo nombre_archivo.o
```

Bajo el lenguaje C++:

Para compilar:

```
# mpicxx -c nombre_archivo.cpp
```

Para enlazar y crear el ejecutable:

```
# mpicxx -o nombre_archivo nombre_archivo.o
```

Los trabajos de MPICH se ejecutan bajo el sistema administrador MPD, mediante el comando **mpirun**. La sintaxis del comando **mpirun** es:

```
# mpirun -n número_de_procesos programa
```

También es posible decidir o escoger en que computadores se pueden ejecutar los procesos. Se especifican los computadores y la cantidad de procesos por computador mediante la sintaxis:

```
# mpirun -n procesos -host nodo0 nodo1 : -n procesos -host nodo2 nodo3 programa
```

En la Sección 3.5 se describe un programa escrito bajo MPI, que fue compilado y ejecutado bajo la implementación de MPICH.

3.3. LIBRERÍAS MATEMÁTICAS

Se han desarrollado algunas librerías para resolver una serie de problemas matemáticos y científicos, como: PETSc y ScaLAPACK. ScaLAPACK es muy útil para resolver problemas de algebra lineal, especialmente aquellos que involucran matrices. PETSc por otra parte permite resolver problemas científicos, en los cuales se requiere derivación e integración compleja [1].

MPI soporta ambas librerías, y el uso de comunicadores permite aislar el universo de las comunicaciones de una parte a otra del programa. Estas librerías matemáticas se las puede encontrar en el sitio Web:

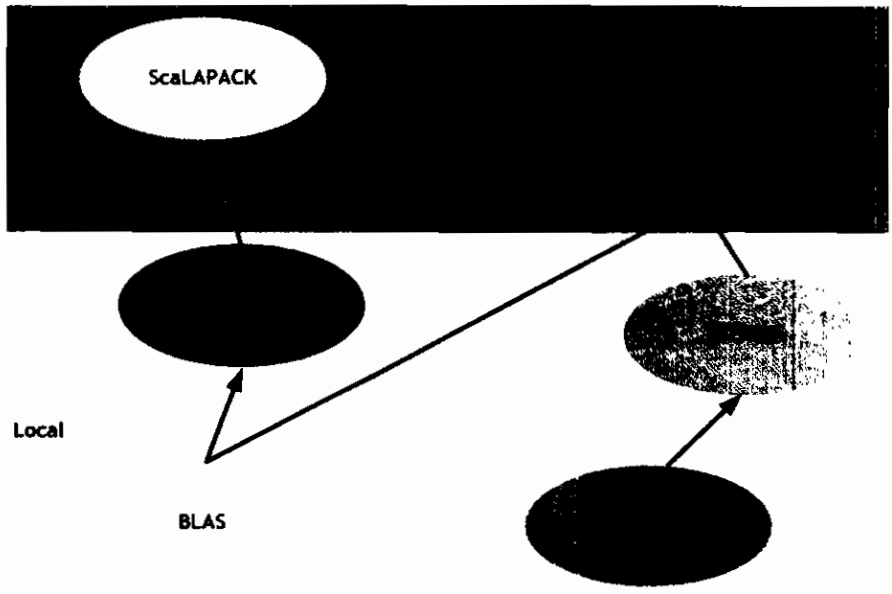
http://www.netlib.org/lapack/lawns

3.3.1. LIBRERÍAS MATEMÁTICAS SERIALES Y PARALELAS

Los desarrolladores de software científico crearon varias librerías matemáticas de entre las cuales se pueden citar:

- BLAS: *Basic Linear Algebra Subprograms.*
- BLACS: *Basic Linear Algebra Communication Subprograms.*
- PBLAS: *Parallel* BLAS.
- PBLACS: *Parallel* BLACS.
- LAPACK: *Linear Algebra PACKage.*
- ScaLAPACK: *Scalable* LAPACK.

En la Figura 3-11, las librerías que se describen bajo la palabra "Local" son seriales. Se puede apreciar que LAPACK se basa en BLAS. Cada grupo o empresa que ha desarrollado dichas librerías ha optimizado su trabajo sobre sistemas de un solo procesador: computadores vectoriales, estaciones de trabajo y computadores personales. Esta optimización se desarrolló a través de un bloque de algoritmos diseñados para almacenar y reutilizar datos críticos en niveles bajos de la jerarquía de memoria, tales como: registros, caché primaria, caché secundaria y por último memoria local.



Fuente: [3], Capítulo 10, página 161

Figura 3-11. Jerarquía de librerías matemáticas

Las librerías que están bajo la palabra clave "Global" son librerías paralelas. De manera análoga a las librerías seriales, los contenidos de ScaLAPACK están basados en PBLAS. Estas son las librerías que se utilizan para realizar cálculos algebraicos lineales en paralelo. La Figura 3-11 muestra que ScaLAPACK está construida sobre BLACS, debido a que ésta última utiliza llamadas que permiten transferir datos locales de memoria de un procesador a otro. En realidad, las llamadas de BLACS son rutinas encapsuladas que hacen llamadas a mensajes de bajo nivel de MPI.

3.3.1.1. BLAS y PBLAS

Estas librerías contienen versiones seriales y paralelas de procedimientos de algebra lineal básica. Definen tres niveles en los cuales pueden encontrarse sus llamadas:

- Nivel 1: operaciones vector–vector, tales como: copia, suma, producto punto.
- Nivel 2: operaciones matriz–vector, tales como multiplicación, producto cruz.
- Nivel 3: operaciones matriz–matriz, de entre ellas se tienen: multiplicación, matriz transpuesta, y matriz inversa.

Todos estos niveles fueron diseñados para trabajar con una variedad amplia de matrices, como: genéricas, simétricas, complejas, y matrices triangulares.

3.3.1.2. BLACS

Esta librería permite transferir datos entre procesos. Contiene llamadas para realizar comunicaciones tipo punto a punto, tipo difusión, y la posibilidad de realizar cálculos globales (de manera local y en máquinas diferentes) tales como: maximización y minimización; siempre y cuando los datos residan en diferentes procesadores.

Una de las funcionalidades de las llamadas BLACS en las comunicaciones es el tratamiento de arreglos: los datos son transferidos dentro de un arreglo.

La librería posee llamadas importantes para crear y examinar la topología de procesadores, esta librería ha sido desarrollada e implementada junto a PBLAS y ScaLAPACK.

3.3.1.3. ScaLAPACK

ScaLAPACK es un estándar de facto de librerías paralelas numéricas. Esta librería fue diseñada e instalada en varias plataformas paralelas multiprocesador, como Cray T3E, SGI, Origin 2000, IBM SP2, Intel Paragon, redes de estaciones de trabajo, y sistemas *clusters* tipo *Beowulf*.

La librería LAPACK contiene llamadas para cálculos de algebra lineal serial optimizadas para una variedad de procesadores. Por otra parte, ScaLAPACK incluye dos librerías: PBLAS y PBLACS.

PBLAS fue diseñada para desempeñar operaciones paralelas de algebra lineal de bajo nivel. Las llamadas y rutinas de PBLACS son las responsables de la comunicación entre los procesos, y reemplaza código de MPI en relación a las comunicaciones colectivas.

Estas librerías que en conjunto conforman a ScaLAPACK, permiten independizar al usuario para que pueda decidir como manejar matrices y vectores dentro de procesadores físicos.

3.3.1.4. PETSc

PETSc *(Portable Extensible Toolkit for Scientific Computation)*, es una librería para utilizarse en la solución de ecuaciones diferenciales parciales y diferentes problemas relacionados, como resolución de sistemas de ecuaciones.

Esta librería está conformada en parte por BLAS y LAPACK, y además provee una interfaz para lenguajes Fortran, C y C++. PETSc presenta funcionalidad para llamadas bloqueantes, y contiene un gran número de opciones, las cuales pueden ser configuradas y elegidas en tiempo de ejecución. Por ejemplo, provee un conjunto de programas para solución de sistemas lineales, que pueden ser utilizados desde una misma aplicación. Se puede elegir el tipo de solución de sistemas lineales, mediante el programa específico en tiempo de ejecución, por medio de un parámetro que se ingresa en línea de comandos. Ésta es una característica muy útil, ya que permite utilizar una herramienta adicional en lugar de recompilar el código (donde se añade funcionalidades nuevas para la solución de los sistemas lineales).

3.4. DESARROLLO DE APLICACIONES CON MPI

Una vez que se han descrito brevemente cada uno de los conceptos y fundamentos de MPI, se espera escribir aplicaciones bajo la implementación MPICH.

Para poder escribir una aplicación primero se debe conocer que es lo que se va hacer, los recursos computacionales que se requieren y la plataforma o ambiente de programación a utilizar.

3.4.1. ESTRUCTURA DE UN PROGRAMA MPI

La estructura básica de un programa escrito con MPI, basado en los lenguajes de programación C y C++, puede estar conformado por las siguientes partes:

1. Archivos de cabecera.
2. Convenciones de nombres MPI.
3. Llamadas y valores de retorno MPI.
4. Manipuladores.
5. Tipos de datos.

6. Inicialización y liberación de librerías MPI.
7. Comunicadores.

3.4.2. CABECERAS DE MPI

Las cabeceras son archivos que contienen los prototipos de las funciones de MPI, así como también la definición de macros, constantes especiales, y tipos de datos. Todo archivo de código donde se requiera incluir dichas cabeceras deberán contener la directiva "**#include**" con la respectiva cabecera:

Para C:

#include <mpi.h>.

Para C++:

#include <mpicxx.h>.

3.4.3. CONVENCIONES DE NOMBRES DE MPI

Todos los nombres de funciones y tipos de MPI poseen un prefijo para C: **MPI_**. Por otra parte, en C++ se tiene un espacio de nombres que contiene todas las clases de MPI, el cual es designado por **MPI** seguido del operador de alcance (su sintaxis es **MPI::**, por ejemplo: **MPI::INT**).

Todas las constantes en MPI están designadas en mayúsculas precedidas por el prefijo **MPI_**.

3.4.4. LLAMADAS Y VALORES DE RETORNO DE MPI

Toda función en MPI para los lenguajes C y C++, tiene un valor de retorno tipo entero. Si la llamada ha sido exitosa, el valor de retorno es **MPI_SUCCESS**. Si el valor es diferente a este macro, dependiendo de la implementación de MPI que se disponga, se dispondrá de otros tipos de macros específicos para cada error.

Toda llamada en MPI inicia con el prefijo **MPI_** para el caso del lenguaje C. En el lenguaje C++ se debe crear un objeto para acceder a sus funciones miembro.

3.4.5. MANIPULADORES

MPI define y mantiene su propia estructura de datos relacionada a las comunicaciones, inicialización, y otras funcionalidades. MPI hace referencia a estas estructuras de datos a través de manipuladores. Los manipuladores se utilizan como valores de retorno para algunas llamadas de MPI, y pueden ser utilizados como argumentos dentro de otras llamadas de MPI.

3.4.6. TIPOS DE DATOS

MPI define tipos de datos primitivos que tienen correspondencia con los tipos de datos en C.

Para el lenguaje C, MPI provee una amplia variedad de tipos de datos especiales (definidos como estructuras). De los más importantes se tienen a:

- MPI_Comm: Define el comunicador.
- MPI_Status: Define una estructura que contiene piezas de información del estado de las llamadas de MPI.
- MPI_Datatype: Es una estructura para definir un tipo de dato especial.

Para el lenguaje C++, MPI define clases para tipos de datos especiales como:

- MPI::Comm
- MPI::Status

Los tipos de datos para C se especifican en la Tabla 3-1, y los tipos de datos para C++ se especifican en la Tabla 3-2.

Tipo de dato MPI	Tipo de dato equivalente
MPI_CHAR	char
MPI_SHORT	short
MPI_INT	int
MPI_LONG	long
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED_LONG	unsigned long
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	ninguno
MPI_PACKED	ninguno

Tabla 3-1. Tipos de datos de MPI para C

Tipo de dato MPI	Tipo de dato equivalente
MPI::CHAR	char
MPI::SHORT	short
MPI::INT	int
MPI::LONG	long
MPI::UNSIGNED	unsigned int
MPI::UNSIGNED_CHAR	unsigned char
MPI::UNSIGNED_SHORT	unsigned short int
MPI::UNSIGNED_LONG	unsigned long
MPI::FLOAT	float
MPI::DOUBLE	double
MPI::LONG_DOUBLE	long double
MPI::BYTE	ninguno
MPI::PACKED	ninguno

Tabla 3-2. Tipos de datos de MPI para C++

3.4.7. INICIALIZACIÓN Y LIBERACIÓN DE LIBRERÍA

Es necesario inicializar la librería de paso de mensajes de MPI, para ello se utiliza la llamada **INIT**:

En C:

```
MPI_INT MPI_Init(&argc, &argv);
```

En C++:

```
void MPI::Init(argc,argv);
```

Esta función debe ser llamada una sola vez durante el programa. Ésta toma los valores de las variables desde una entrada de datos estándar, que por defecto en diferentes sistemas operativos es un *shell* (**argc** y **argv**).

Una vez que el programa concluye sus tareas, es imprescindible liberar las librerías de MPI, para ello se utiliza la función **FINALIZE**:

En C:

```
MPI_INT MPI_Finalize();
```

En C++:

```
void MPI::Finalize();
```

3.4.8. CÓDIGO DE EJEMPLO

Para poder ilustrar las funcionalidades de MPI se desarrolló un programa paralelo para el cálculo de PI, mediante MPICH2.

El programa encuentra el valor de PI por medio de integración numérica. Para ello se utiliza la integración de la función $\frac{4}{1+x^2}$, en el intervalo de 0 a 1.

El resultado se obtiene mediante:

$$\int_0^1 \frac{1}{1+x^2}dx = \arctan(1) - \arctan(0) = \arctan(1) = \frac{\pi}{4}$$

(3-1)

Por lo que el cálculo de PI se reduce a:

$$\pi = \int_0^1 \frac{4}{1+x^2}dx$$

(3-2)

La aproximación se realiza dividiendo el intervalo $[0;1]$, en un número de subintervalos y luego se calcula el área total de los rectángulos obtenidos, asignando a cada proceso el cómputo de las áreas de un subintervalo. En la Figura 3-12 se muestra la división del área bajo la curva y la obtención de subintervalos.

El algoritmo aplicado para la resolución de PI utiliza un modelo maestro-esclavo. Donde el proceso raíz o maestro recopila la información para presentar el resultado, además de realizar operaciones de cálculo. Por otra parte, mientras más rectángulos existan más trabajo deben realizar los esclavos y el error en el valor de PI es menor.

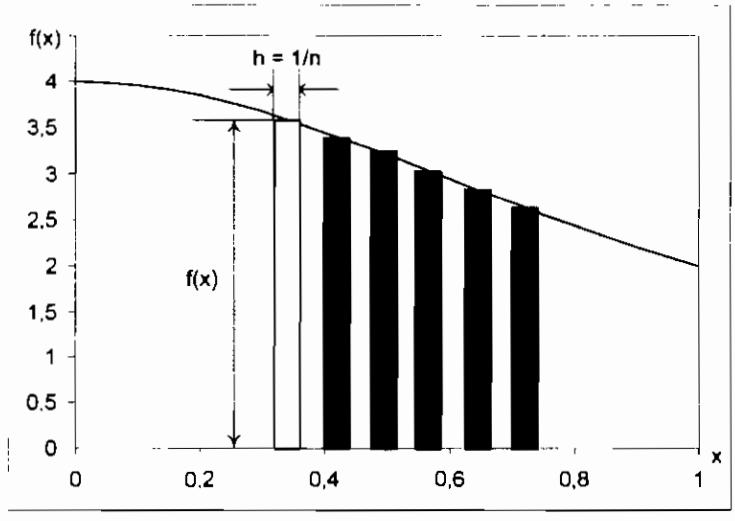


Figura 3-12. Integración mediante suma de áreas de rectángulos

Código en C++:

```
#include <mpi.h>          // libreria de MPI

#include <iostream>       // librerias de entradas y salidas
#include "math.h"         // librerias matematicas

using namespace std;      // espacio de nombres estandar

int main(int argc, char *argv[])
{
        int n;                  // numero de intervalos (trapezoides)
        int rank;               // identificador de proceso
        int size;               // numero de procesos
        int i;                  // identificador de iteraciones
        double mypi;            // buffer de envio
        double pi;              // buffer de recepcion
        double h;               // ancho del rectangulo
        double sum;             // altura de rectangulo
        double x;               // valor del eje de abscisas
        double PI25DT = 3.14159265358979323846264643;
                                // valor experimental de PI

        // Inicializacion de MPI
        MPI::Comm  comm =  MPI::Comm(MPI_COMM_WORLD);
        MPI::Init(argc, argv);            // Inicializacion de MPI
        size = comm.Get_size();           // Obtencion del # de procesos
        rank = comm.Get_rank();           // Obtencion del identificador
                                          // del proceso

        while (1)
        {
            if (rank == 0)
            {
                  cout << "Ingrese el numero de intervalos: (0 sale)"
                       << endl;
                  cin >> n;
            }

            // Envio de datos a todos los procesos
            // Bcast(*void mensaje, int count, MPI_datatype tipodedato, int root)
            // Se envia el numero de intervalos, en un solo mensaje
            // valor 1), tipo entero MPI, desde el proceso raiz.
            comm.Bcast(&n, 1, MPI::INT, 0);

            if (n==0)           // Fallo en las comunicaciones
                  break;
            else
            {
                  h = 1.0 / (double) n;  // calculo del # de rectangulos
                  sum = 0.0;
                  for (i = rank + 1; i <= n; i += size)
                  {
                        // calculo del ancho de rectangulos
                        x = h * ((double)i - 0.5);
                        // sumatorio del area parcial de los rectangulos
                        sum += (4.0 / (1.0 + x*x));
                  }
                  // calculo del area total, resultado final
                  mypi = h * sum;
```

```
                // Recepcion de datos de todos los procesos
                // Reduce(void* operator,void* result, int count, MPI_Op operator,
                //           int process)
                // Se envia el valor parcial de pi (operando), se envia
                // el parametro donde se va a almacenar el resultado,
                // la transmision en un solo mensaje (valor 1),
                // operacion de MPI  tipo suma, al proceso raiz.
                comm.Reduce(&mypi, &pi, 1, MPI::DOUBLE,MPI::SUM, 0);

                // En el proceso raiz, imprime el resultado de la
                // operación y se determina el error relativo que se
                // obtiene de la aproximacion realizada
                if (rank == 0)
                      cout << "\nPI es aproximadamente: " << pi
                           << ", El error es: " << fabs(pi - PI25DT)
                           << endl << endl;
          }
        }
        MPI::Finalize();  // liberacion de libreria MPI
        return 0;
}
```

La compilación y la ejecución del programa se realizaron sobre un sistema Linux, con un sólo procesador. Los resultados obtenidos se muestran en la Figura 3-13.

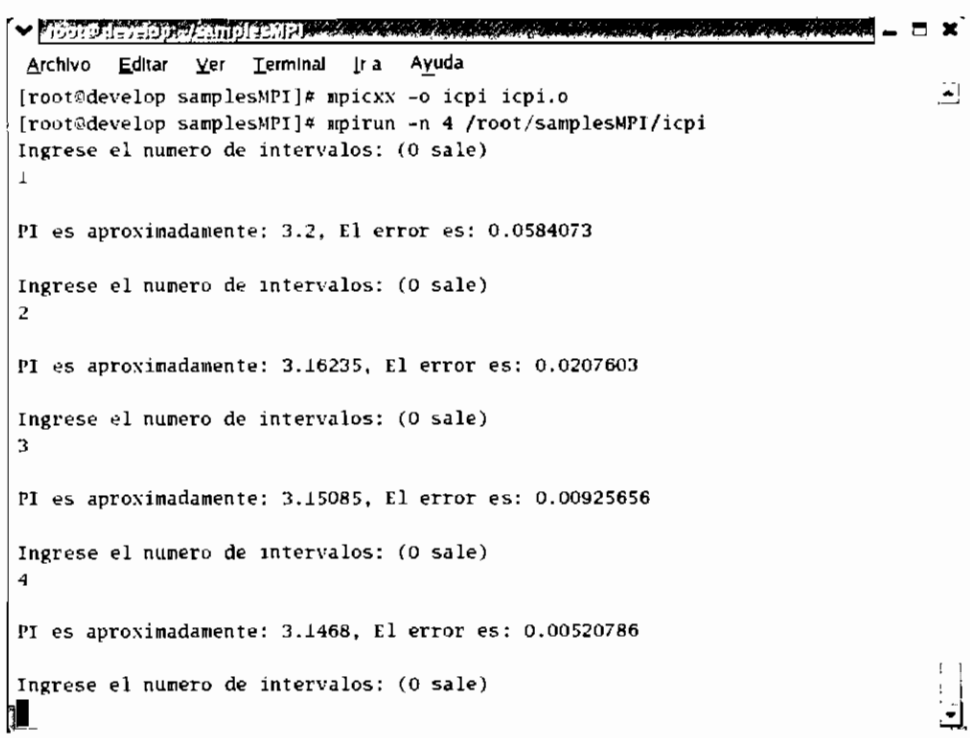


Figura 3-13. Compilación y ejecución de programa paralelo

3.5. REFERENCIAS

[1] PACHECO, Peter: *Parallel Programming with MPI.* Morgan Kaufmann Publishers, Inc., Primera Edición, Estados Unidos, 1997.

[2] GROPP, William; LUSK, Ewing; DOSS, Nathan; SKJELLUM, Anthony: *A High Performance, Portable Implementation of MPI Message Passing Interface Standard.* White Paper: Argonne National Laboratory y Department of Computer Science & NSF Engineering Research Center for CFS Mississippi State University, Estados Unidos, 2001.

[3] PACS, Training Group & NCSA: *Introduction to MPI.* NCSA Tutorial, Estados Unidos, 2001.

[4] http://www.llnl.gov/computing/tutorials/mpi/index.html
Ultimo acceso: 25/09/2004

[5] GROPP, William; LUSK, Ewing: *Why Are PVM and MPI So Different?* White Paper: Mathematics and Computer Science Division, Argonne National Laboratory, Estados Unidos, 2000.

[6] GROPP, William; LUSK, Ewing: *Goals Guiding Design: PVM and MPI.* White Paper: Mathematics and Computer Science Division, Argonne National Laboratory, Estados Unidos, 2000.

CAPÍTULO 4. ADMINISTRACIÓN, SISTEMAS DE ARCHIVOS Y PLANEACIÓN DE TAREAS

Este capítulo presenta las ideas básicas involucradas en la administración de recursos en los sistemas paralelos. Se mencionan algunas alternativas de herramientas de software que permiten la administración, monitoreo y balanceo de carga computacional en los *clusters* de computadores personales. Estas tareas específicas pueden ser administradas con herramientas bien conocidas, tales como: C3 (*Cluster Command & Control*) y Ganglia, que presentan funcionalidades para la administración y monitoreo; Condor y PBS (*Portable Batch System*), que permiten la planificación, asignación de recursos y tareas. Se discutirá brevemente algunas alternativas de sistemas de archivos paralelos y algunos aspectos relacionados a la instalación y configuración de la implementación más difundida de sistemas de archivos paralelos, denominada PVFS (*Parallel Virtual File System*).

4.1. HERRAMIENTAS PARA LA ADMINISTRACIÓN DE *CLUSTERS*

La operación de *clusters* requiere de un manejo adecuado de los recursos asociados. Los recursos del *cluster* deben ser administrados adecuadamente para que el administrador invierta la menor cantidad de tiempo en detectar, investigar y recuperar fallos de hardware y software, y de este modo definir posibles medidas de contingencia y tratar que el sistema esté libre de errores. A su vez, estos pasos permiten la adaptabilidad a los requerimientos y cambios constantes que se presentan en la manipulación de tecnologías *cluster*, en cuanto se refiere al hardware, software y al uso de ciertos patrones de diseño.

El administrador de un *cluster* debe tomar en cuenta algunos aspectos, una vez que se ha completado la instalación de los recursos básicos de hardware y software. Estos aspectos incluyen la configuración e instalación de un sistema de archivos universal, la configuración y administración de recursos mediante herramientas implementadas en software; el monitoreo de sus actividades y el registro de cada uno de los eventos generados por la ejecución de cálculos computacionales.

Varios de los sistemas más importantes para la instalación automática de *clusters*, incluyen herramientas de monitoreo, administración y registro de eventos mediante paquetes de distribución para sistemas Windows y Linux. Entre estos sistemas están OSCAR y *Rocks* NPACI; ambos sistemas permiten el uso de herramientas de software que tienen propósitos específicos tales como:

- Definición y administración de nodos.
- Administración de colas por lotes (*Batch Queue Management*).
- Administración de recursos: grupos NIS (*Network Information Service*), cuotas de disco y CPU.
- Administración de servicios de resolución de nombres: DNS (*Domain Name System*) para *clusters*.
- Registro de usuarios para *clusters* de dimensiones superiores a los 100 nodos.
- Monitoreo de carga – *cluster*.

A continuación se describen ciertos conceptos básicos de administración de *clusters* y un conjunto de herramientas de software para ambientes paralelos que cumplen con las características para el monitoreo, registro de eventos y de cuentas de usuario.

4.1.1. CONCEPTOS DE ADMINISTRACIÓN DE *CLUSTERS*

La administración de *clusters*, implica tomar medidas preventivas y planificar tareas. La administración implica los siguientes aspectos:

- Registro de eventos.
- Monitoreo o medida del estado de los recursos del *cluster.*
- Recuperación ante fallos de hardware, software, incluyendo el sistema de archivos.
- Administración del registro de usuarios y grupos de usuarios, de los servicios del *cluster* (*accounting*).
- Planificación de tareas y balanceo de carga.

4.1.1.1. Registro de Eventos

La administración de *clusters* implica fundamentalmente el manejo de *logs*, o el registro de eventos que genera tanto el *kernel* del sistema operativo, como los diferentes servicios que han sido habilitados para el establecimiento de comunicación entre los nodos.

El registro (*logging*) es el proceso por medio del cual todos los aspectos de la operación de cada nodo y del *cluster* en general, pueden ser almacenados para un uso futuro en un archivo o archivos de información. En un *cluster*, tal como en un computador personal con una distribución Linux instalada, el sistema operativo y sus servicios se configuran, y toda esta información y su estado se almacenan en archivos de registro (*logs*).

En niveles más altos (referidos a capas de software o *middleware*), el software de administración, las librerías, y cualquier otro programa de usuario generan automáticamente un registro de los eventos de sus diferentes actividades.

En un sistema Linux, el servicio **syslog** permite registrar los eventos que se producen por la ejecución de los programas en el sistema. La ubicación física de los mensajes generados dentro del sistema de archivos varía de acuerdo al servicio o paquete instalado; sin embargo, el directorio común es el `/var/log`. Algunos servicios del *cluster* almacenan en dicho directorio su registro de eventos.

Se pueden utilizar comandos del sistema operativo Linux para poder visualizar los archivos de *logs*, o utilizar herramientas de monitoreo tales como:

- *LogCheck* [1].
- *Swatch* [2].
- *LogSentry* [3].
- *LogDog* [4].

En Linux, el comando más utilizado para visualizar *logs* es: **tail**. **tail** permite visualizar un número determinado de líneas de un archivo desde el final, además, mediante la opción **–f** se visualizan sólo las últimas líneas que han cambiado:

```
# tail -f archivo_de_log
```

Los archivos de *logs* pueden crecer de manera indefinida y su tamaño puede llegar a ser sumamente grande. Para minimizar y optimizar el espacio en disco, y disminuir el crecimiento de los archivos de *logs*, se utiliza el servicio **cron**, que permite programar tareas para comprimir archivos y subdirectorios, y a su vez renombrarlos.

4.1.1.2. Monitoreo y Estado del *Cluster*

En algún determinado momento, los diferentes componentes de hardware y software pueden llegar a operar en un estado degradado. A este nivel las aplicaciones funcionan, sin embargo no cumplen con los niveles de desempeño esperados y puede ocasionar problemas de desempeño futuros si es que el sistema o los componentes fallan por completo.

El monitoreo permite conocer si todos los componentes de hardware y software están disponibles y operando de acuerdo a lo esperado. Es decir, debe asegurarse que todos los componentes de hardware estén disponibles durante el arranque del sistema operativo (CPUs, memoria, discos, dispositivos de red y otros), y de igual forma, que todos los servicios de software, tales como:

planificadores de tareas, administradores de recursos, y demonios de monitoreo se ejecuten correctamente en el *cluster*.

Varias herramientas de monitoreo para *clusters* se pueden encontrar para diferentes distribuciones del sistema operativo Linux, entre las cuales se tienen: *Big Brother* [4], *Cluemon* [7], Ganglia [12], *Nagios* [8], PARMON [9], *Performance Co-Pilot* [10] y *Supermon* [11].

4.1.1.3. Recuperación ante fallos

La administración del *cluster* implica resolver problemas provocados por fallos de hardware y/o software. Los fallos causados por hardware pueden ocasionar que el *cluster* quede inutilizable. Algunos de los fallos pueden ser detectados mediante las herramientas de monitoreo y la verificación física del estado de los componentes de la red del *cluster*: *routers*, *switches*, nodos y el sistema de cableado estructurado.

Por otra parte, el impacto puede ser menor, si el fallo es ocasionado por componentes del sistema que no degradan completamente sus actividades. Por ejemplo, si un nodo falla, todos los demás deberán estar disponibles para desarrollar las actividades computacionales normales; sin embargo, los tiempos de respuesta de la aplicación que se ejecuta sobre los nodos, pueden verse considerablemente afectados.

La recuperación ante fallos a nivel de hardware implica:

- Aislar los componentes que fallaron para asegurar que no causen un considerable impacto en las actividades del *cluster*.
- Manejar los componentes de respaldo (*backup*), para poder hacer reemplazos y minimizar los efectos del fallo.

Los fallos de componentes de software muchas veces no tienen solución o forma de recuperación. Si se considera que el sistema operativo está basado en Linux, la mayor parte de aplicaciones requieren de parches o nuevas versiones para mejorar o recuperarse de errores; sin embargo, este proceso es muy complejo y

conlleva mucho tiempo. Por tal motivo, si un componente de software falla lo único que resta por hacer es informar al vendedor, diseñador o desarrollador de la aplicación y esperar por las mejoras.

Para asegurar que el software (*kernel*, distribución, planificador de tareas o administrador de recursos, y librerías) opere en condiciones normales y evitar en cierta medida los fallos, se recomienda [5]:

- Buscar información sobre nuevas versiones y parches.
- Desarrollar pruebas y verificaciones antes de actualizar nuevas versiones de software.
- Tratar de regresar (*rollback*) a versiones anteriores, por ejemplo, si nuevas versiones de distribución son instaladas y estas no resuelven los fallos.
- Mantener un registro de los fallos sin solución, debido a que muchos de ellos pueden desaparecer una vez reiniciado el sistema.

Si un componente de hardware o software falla, se perderán únicamente los resultados de las aplicaciones que se estaban ejecutando, y que pueden ser recuperados una vez que se reinicien los servicios. Por otra parte, si el sistema de archivos contiene resultados almacenados de una cantidad de trabajo realizada por meses, y esta información vital se pierde, el impacto podría ser crucial. Por esta razón, el componente del *cluster* que se considera más crítico es el sistema de archivos y almacenamiento, que los usuarios utilizan para guardar sus datos y aplicaciones.

Una de las formas de prevención más utilizada se realiza mediante respaldos (*backup*) de la información en una fecha y hora determinada (que utilizan servicios de tareas programadas como **`cron`**).

Existen varias formas de realizar respaldo de la información, éstas son:

- Utilizar sistemas de redundancia tales como RAID 0, 3, o 5.
- Utilizar sistemas de archivos que realicen tareas específicas planificadas previamente; similar a las opciones del comando **chkdsk** y *toolkits* como

Norton Utilities para sistemas Windows y los comandos **fsck** o **e2fsck** para sistemas Linux.

4.1.1.4. *Accounting*

Los ambientes Linux ofrecen algunas alternativas para mantener copias de un conjunto de archivos en varios equipos. La forma más común y fácil de administrar las copias de un conjunto de archivos involucra la utilización de una red basada en servicios para la administración de cuentas o registros de usuario. Cuando se emplea esta alternativa, cada computador realiza consultas a un servicio central, el cual maneja la autorización, la autenticación y la información de los usuarios dentro del sistema.

Para la configuración manual de *clusters,* los servicios más utilizados son NIS (*Network Information Service*) o LDAP (*Lightweight Directory Access Protocol*); sin embargo, también se habilitan de forma automática con los *toolkits* de OSCAR y NPACI *Rocks*. Al usar NIS, la contraseña, el grupo, y otros archivos de seguridad (archivo de contraseñas: `/etc/shadows`, archivo de computadores disponibles en la red `/etc/hosts`, `/etc/aliases`, archivo de grupo NIS `/etc/netgroup`, archivo de protocolos de comunicación `/etc/protocol`, archivos de servicios RPC `/etc/rpc`, archivo de servicios disponibles en el dominio NIS `/etc/services`, archivo de identificador de red del dominio NIS `/etc/netid` y archivo de los servidores NIS disponibles `/var/yp/ypservers`) se almacenan de forma centralizada en un servidor denominado **ypserver**. Cada nodo se configura como un cliente NIS. El nodo de administración se convierte en el servidor NIS, el cual almacena los archivos en su base de datos y transfiere la información de estos archivos a los nodos cliente. LDAP, por otra parte, es descrito como un directorio distribuido, donde residen un conjunto o arreglos de objetos con atributos propios: nombres, contraseñas, grupos, carpetas compartidas, permisos, accesos, y otros.

4.1.1.5. Planificación de Tareas y Balanceo de Carga

La planificación de tareas y el balanceo de carga en un *cluster*, son las actividades más visibles cuando se refiere a ejecutar aplicaciones de usuario.

A continuación se enumeran actividades de administración y balanceo de carga que son críticas para un entorno *cluster*:

- Administrar la disponibilidad de los nodos.
- Configurar atributos de los nodos que sean importantes para balanceo de carga.
- Administrar usuarios y grupos mediante cuotas de disco.
- Configurar y diseñar políticas de planificación de tareas.
- Administrar reservaciones y recursos dedicados.
- Monitorear y generar un historial de utilización de recursos para usuarios y grupos.

4.1.2. SOFTWARE DE ADMINISTRACIÓN DE *CLUSTERS*

A continuación se describen las herramientas para administración de *clusters:* C3 (*Cluster, Command & Control*) y Ganglia.

4.1.2.1. Herramienta C3 – *Cluster Command & Control*

C3 es un conjunto de utilidades basadas en línea de comandos. Éstas son utilizadas para ejecutar tareas comunes de administración. Estos comandos se diseñaron para proveer un ambiente similar a los comandos comunes que se utilizan bajo la administración de una máquina con UNIX o Linux. Estos comandos son *scripts* escritos en Phyton[1].

[1] Lenguaje orientado a objetos para desarrollo de *scripts* bajo ambientes Linux.

C3 fue desarrollado por el Laboratorio Nacional Oak Ridge, y su distribución es libre[2]. Esta herramienta se instala automáticamente con la distribución de OSCAR.

A continuación se describe el proceso de instalación, configuración y administración de C3.

4.1.2.1.1. Instalación de C3

Para instalar C3 se requiere tener un conjunto de servicios previamente habilitados:

- **rsync**. Es una utilidad Linux que provee transferencia de archivos de forma rápida. Sólo envía los bytes que han sido modificados no el archivo completo, por lo que es utilizado para sincronizar datos entre equipos.

- Perl. Es un lenguaje para desarrollar *scripts*. Provee una interfaz de desarrollo integrado en el *shell* de sistemas Unix [19].

- SSH (*Secure Shell*). Es una utilidad Linux que implementa un mecanismo de autenticación basado en llave pública, que encripta todo el tráfico entre un proceso local y uno remoto mediante una llave privada.

- Phyton. Es un lenguaje orientado a objetos para desarrollo de *scripts*. Existen implementaciones para sistemas Windows y Linux [20].

Es imprescindible disponer del servicio de resolución de nombres de dominio DNS o debe existir el archivo `/etc/hosts`[3].

---

[2] Posee una licencia GPL (*General Public License*) amparada bajo el *copyleft*, donde su distribución contiene su código fuente, el cual puede ser alterado, distribuido y obtener remuneración por las mejoras realizadas.

[3] El archivo `/etc/hosts` debe contener los nombres de los computadores que conforman el *cluster* y de sus direcciones IP asociadas.

Existen dos formas de instalar C3. La primera permite centralizar la administración en un nodo principal, y la segunda hace que todos los nodos hereden la configuración del nodo principal. En la segunda alternativa se forma una jerarquía entre el nodo principal y los otros nodos, lo cual permite establecer con mayor rapidez el comando a ejecutarse.

C3 se puede descargar del sitio Web: http://www.cms.ornl.gov/torc/C3/. C3 se puede obtener como un paquete RPM o como una fuente binaria[4]. Los comandos para la instalación de C3 son:

Para instalar C3 a partir de un RPM:

```
# rpm -ihv c3-4.0.1.i386.rpm
```

Para instalar C3 a partir de una fuente binaria:

```
# tar -zxvf c3-4.0.1.tar.gz
# cd c3-4.0.1
# ./install-c3
```

En ambos casos, tanto la instalación así como todos los *scripts*, variables de entorno y páginas de ayuda se almacenan en el directorio `/opt/c3-4`. Luego de la instalación del paquete, se crea un archivo de configuración, el cual está ubicado en el directorio `/etc/c3.conf`.

En la Figura 4-2, se muestra el archivo de configuración de C3. El texto en color azul representa comentarios, y el color negro la configuración de C3.

[4] En Linux, se utiliza el término fuente binaria (*binary source*) para referirse a un paquete que tiene que ser compilado de acuerdo a la distribución Linux que se está utilizando. Una descripción de la instalación mediante fuentes binarias se presenta en el ANEXO C.

```
# nombre del cluster al cual se registran 3 nodos en el dominio
# dominio

cluster nombre_cluster{
nombre1.dominio.int
nombre2.dominio.int
nombre3.dominio.int
}

# El archivo /etc/hosts contiene las direcciones IP de los nodos
```

Figura 4-1. Archivo de configuración de C3

Para terminar la instalación de las herramientas C3, se debe configurar **ckill**. El comando **ckill**, permite liberar procesos en los nodos del *cluster* utilizando el nombre del proceso en lugar de su identificador. Este comando debe estar presente en todos los nodos del *cluster*. Por este motivo se debe ejecutar el *script* **ckillnode**:

```
# cexec mkdir /opt/c3-4
# cpush /opt/c3-4/ckillnode
```

El primer comando crea el directorio `/opt/c3-4/` en cada nodo, y el segundo copia el *script* en el directorio `/opt/c3-4/` de cada nodo.

4.1.2.1.2. Utilización de C3

C3 presenta una serie de comandos, en la Tabla 4-1 se mencionan los más importantes, su descripción y un ejemplo de utilización.

Comando	Descripción	Ejemplo
cexec	Ejecuta un comando en todos los nodos.	`# cexec "ps | grep ps.txt"`
cget	Copia archivos de una cierta ubicación en los nodos. Ignora enlaces y directorios. Si existe un nombre de archivo con el mismo nombre lo renombra con un sufijo formado por el nombre del nodo del *cluster*.	`# cget /etc/rc.d/rc.local`
ckill	Permite finalizar un proceso en ejecución en los nodos del *cluster*. Para utilizarlo se utiliza el nombre del proceso, y no su ID de proceso debido que en cada nodo el ID es diferente.	`# ckill -u talkd log.txt`
cpush	Permite mover archivos de una cierta ubicación en los nodos.	`# cget /home/local /home/rc.bk`
crm	Permite eliminar archivos y directorios en los nodos. Su funcionamiento es similar al comando rm, con las opciones de interactivo y recursivo.	`# crm -iR /home/dafa/ver1/`
cshutdown	Permite apagar, reiniciar o suspender un nodo. Las opciones son las mismas del comando ***shutdown*** en un sistema Linux. Adiciona el uso de la opción t para especificar el tiempo que tomará para ejecutar la acción.	`# cshutdown r t 0`

clist, cget, cnum	Son utilizados para hacer consultas de archivos de configuración.	`# clist`
		`# cname nombrecluster:0-1`
		`# cnum nodo4.dominio.int`

Tabla 4-1. Comandos de C3

4.2. ADMINISTRACIÓN CON GANGLIA

Existen varias herramientas que se pueden utilizar para monitorear el estado del *cluster*. Las más conocidas son: Ganglia, *Clumon* y *Performance Co–Pilot* (CPC).

Ganglia es una herramienta de monitoreo en tiempo real para *clusters* y *grids*. Ganglia utiliza la misma base de datos desarrollada para MRTG[5] (*Multi Router Traffic Grapher* [21]) basado en mecanismos de actualización de registros *round-robin*[6].

El primer prototipo de Ganglia aparece con la Versión 1.0 en el año 2000, con las siguientes características:

- Demonios escritos en Perl.
- Introducción de subprogramas conocidos como **axons**, que permiten el almacenamiento de los datos resultantes de la comunicación *multicast* en memoria y los comparten a través de conexiones lógicas (*sockets).*
- Inclusión de una interfaz Web que se comunica con los **axons** para recuperar la información.

La Versión 2.x aparece en el año 2001. Esta versión es más estable y su código fue escrito en lenguaje C. Los **axons** y los demonios de *Perl* originales se incluyeron dentro de un proceso conocido como **gmond**. Esta implementación fue diseñada bajo el concepto de P2P (*Peer to Peer*) utilizando XML (*eXtensible Markup Language*) y XDR[7] (XML *Data Reduce*).

Desde el año 2002, el desarrollo de Ganglia está a cargo del grupo *SourceForge* [12], quienes hicieron posible que su distribución sea multiplataforma: Linux *(*i386, IA64, SPARC, ALPHA*)*, FreeBSD, Windows, AIX, IRIX, MacOS X, y otros.

---

[5] MRTG es una herramienta para monitorear el tráfico de enlaces de red.

[6] En el metodo de actualización de registros *round-robin*, los datos más antiguos se desechan y los nuevos se añaden a la base de datos.

[7] http://www.faqs.org/rfcs/rfc1014.html

La última versión de Ganglia (Versión 2.5) publicada en Enero de 2002, se desarrolló bajo el modelo cliente-servidor y está compuesta de cuatro partes fundamentales, que a continuación se describen:

- El demonio de monitoreo: **gmond**, el cual debe ser instalado en cada nodo del *cluster*.
- El *backend* para la recolección de los datos, el demonio: **gmetad.** Instalado únicamente en el nodo de administración.
- La interfaz Web, o mejor conocido como *frontend*. Esta debe ser instalada en el nodo de administración únicamente.
- Los datos se transmiten utilizando XML y XDR mediante una conexión TCP y *multicast*. Esto se logra a través de una clase creada con Phyton, para ordenar y clasificar los datos, y desempeñar funciones de comunicación, como la transmisión y recepción de los datos.

Además de los componentes fundamentales, existen dos herramientas bajo línea de comandos. La primera: **gstat**, la cual provee un medio de comunicación para realizar consultas al demonio **gmond**, y permite crear reportes del estado del *cluster*. La segunda: **gmetric**, que permite monitorear de manera sencilla las métricas de los equipos, además de las predefinidas por Ganglia, como: número de procesadores, memoria utilizada, velocidad del CPU entre otros.

El comando **gmetric** trabaja en conjunto con el demonio **crond**, permitiendo realizar un itinerario de tareas; es decir, realizar tareas a cierta hora en un día determinado.

Ganglia provee un ambiente de ejecución único mediante la utilización del comando **gexec,** emplea el uso de 3 hilos de ejecución, uno para las entradas estándar (*stdin*), uno para las señales del sistema, y otro para las entradas y salidas de error (*stderr*). **gexec** permite ejecutar comandos en el *cluster* de manera transparente y redireccionar las salidas por medio de las entradas y salidas estándar (*stdin*, *stdout* y *stderr*).

4.2.1. INSTALACIÓN, UTILIZACIÓN Y ADMINISTRACIÓN CON GANGLIA

4.2.1.1. Instalación de Ganglia

Ganglia se instala mediante fuentes binarias o utilizando paquetes RPM. La instalación del software se lo realiza sobre la estación de administración; es decir, aquella que mantiene la base de datos y recolecta la información de los otros nodos.

En las estaciones en las cuales se instala **gmetad** y la interfaz Web, se necesita instalar el paquete **RRDTool**[8] [13], el lenguaje Perl y habilitar el servidor Web (Apache) con soporte para PHP. Los nodos clientes no necesitan tener habilitados estos servicios.

RRDTool es una base de datos de tipo *round robin*, es decir, cuando se requiere añadir información a la base de datos, los datos más antiguos se desechan y los nuevos se añaden a la base de datos. Esto permite almacenar datos de manera compacta y, además, permite que la información en la base de datos no se expanda en un determinado tiempo.

Para instalar **RRDTool** a partir de una fuente binaria, es necesario descomprimir y compilar la fuente mediante la utilización de los comandos **tar**, **configure**, **make** y **make install**.

Para instalar **RRDTool** a partir de un paquete RPM:

```
# rpm -ihv rrdtool-1.48.i386.rpm
```

Para instalar **RRDTool** a partir de una fuente binaria:

```
# tar -zxvf rrdtool-1.48.tar.gz
# ./configure
# ./make
# ./make install
```

[8] RRDTool: *Round Robin Database Tool*

Para obtener los resultados esperados, y no tener que introducir variables de entorno adicionales es recomendable instalarlos en el directorio: `/usr/local/src`.

Se debe asegurar que los archivos de configuración de Apache y el módulo de PHP estén disponibles. Para la versión Linux Red Hat 9.0, el archivo de configuración primario de Apache es `httpd.conf`, que se encuentra ubicado en el directorio `/etc/httpd/conf/`, y los demás archivos de configuración se encuentran en el directorio `/etc/httpd/conf.d/`. El módulo de PHP es `/usr/local/lib/libphp4.so`.

Una vez configurados **RRDTool**, Apache, PHP, y otros servicios adicionales se debe descargar las fuentes binarias o los archivos RPM de Ganglia desde http://ganglia.sourceforge.net/.

El corazón del monitoreo de Ganglia está conformado por **gmond** y **gmetad**.

Para instalar Ganglia a partir de un paquete RPM:

```
# rpm -ihv ganglia-monitor-core-2.5.6.i386.rpm
```

Para instalar Ganglia a partir de una fuente binaria:

```
# tar -zxvf ganglia-monitor-core-2.5.6.tar.gz
# cd ganglia-monitor-core-2.5.6
[ganglia-monitor-core-2.5.6]# ./configure
[ganglia-monitor-core-2.5.6]# ./make
[ganglia-monitor-core-2.5.6]# ./make install
```

Si no se instaló la herramienta **RRDTool** en el directorio indicado, durante el proceso **configure**, se debe indicar a Ganglia el directorio de las librerías de **RRDTool**. Luego, si se requiere de **gmetad** (opcional), se debe añadir una clave durante el proceso **configure**.

Luego del proceso de configuración[9] (paso previo a la instalación) e instalación, es necesario modificar el archivo de configuración de **gmetad** (`gmetad.conf`) y luego copiarlo en el directorio `/etc/`. El objetivo de modificar el archivo de configuración de **gmetad**, es para especificar la lista de los nodos que serán monitoreados. La configuración por defecto del archivo de **gmond** (`gmond.conf`) es suficiente para objetivos de monitoreo, por tal motivo no suele ser necesario modificarlo.

Para finalizar la configuración de **gmond**, se deben ejecutar los siguientes comandos:

```
[ganglia-monitor-core-2.5.6]# cp ./gmond/gmond.init > /etc/rc.d/init.d/gmond
[ganglia-monitor-core-2.5.6]# chkconfig --add gmond
[ganglia-monitor-core-2.5.6]# service  /etc/rc.d/init.d/gmond start
```

Para asegurar que **gmond** se inicie el momento del arranque se debe ejecutar el siguiente comando (opcional):

```
# chkconfig gmond on
```

Antes de iniciar el servicio **gmetad**, se debe crear el directorio donde se almacena la base de datos de **RRDTool**:

```
[ganglia-monitor-core-2.5.6]# mkdir -p /var/lib/ganglia/rrds
[ganglia-monitor-core-2.5.6]# chown[10] -R nobody > /var/lib/ganglia/rrds
```

[9] Si se disponen de fuentes binarias. antes de instalar el paquete, es necesario configurarlo mediante el *script* `configure`. `configure` determina el sistema y las características del sistema sobre el cual se ejecuta y crea archivos denominados `Makefile`, los cuales contienen las intrucciones necesarias para construir e instalar las fuentes en el sistema.

[10] `chown` permite cambiar el propietario de un archivo.

Una vez realizado este proceso se copian los archivos de configuración y se procede a iniciar el servicio **gmetad**:

```
[ganglia-monitor-core-2.5.6]# cp ./gmetad/gmetad.init \ >
/etc/rc.d/init.d/gmetad
[ganglia-monitor-core-2.5.6]# chkconfig --add gmetad
[ganglia-monitor-core-2.5.6]# service /etc/rc.d/init.d/gmetad start
```

Para verificar que ambos programas están ejecutándose correctamente, se puede utilizar **telnet** para comprobar que los puertos 8649 de **gmond** y 8651 de **gmetad** estén abiertos y listos para aceptar conexiones.

```
# telnet localhost 8649
```

Finalmente, se debe configurar la estación de monitoreo con el software de la interfaz Web. Es necesario verificar cuáles son los directorios que están dentro de la variable de entorno de la directiva **DocumentRoot**[11] del archivo de configuración de *Apache*:

```
# grep DocumentRoot /etc/httpd/conf/httpd.conf
```

La directiva **DocumentRoot** define el directorio donde se debe copiar el archivo fuente de la interfaz Web. Por defecto este directorio es el `/var/www/html/`.

```
# cp ganglia-webfrontend-2.5.5.tar.gz /var/www/html/
# cd /var/www/html/
# tar -zxvf ganglia-webfrontend-2.5.5.tar.gz
```

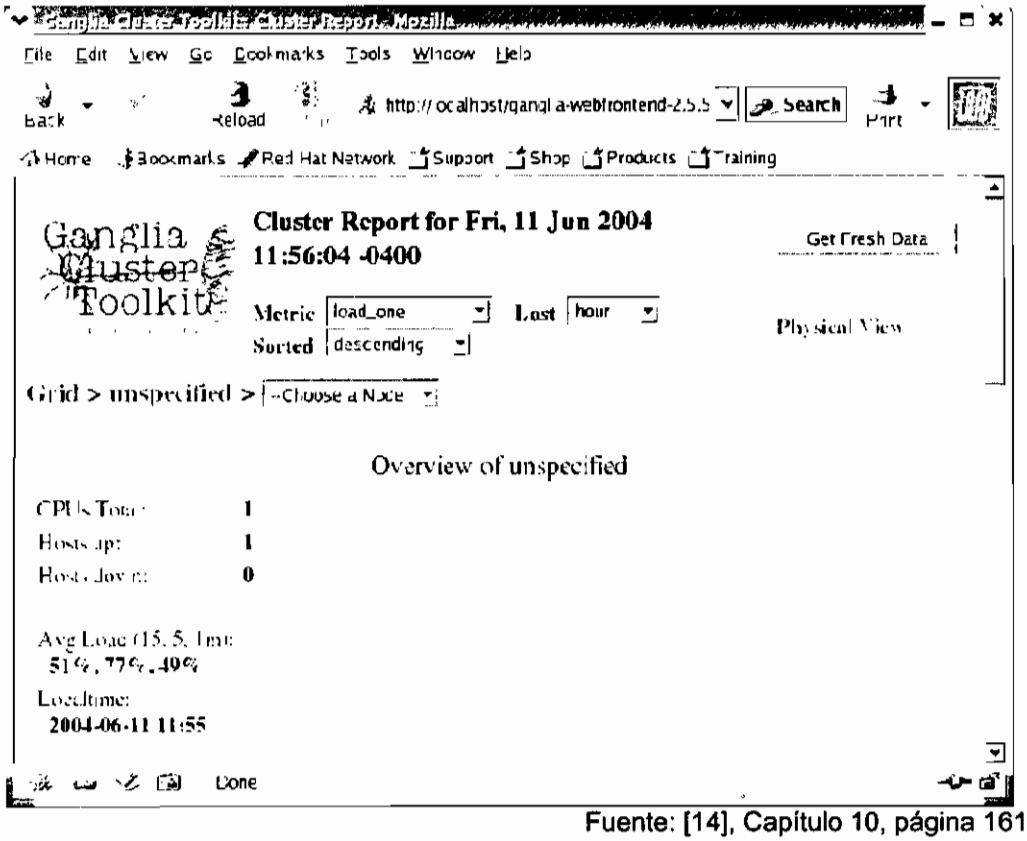
[11] La directiva **DocumentRoot** indica el directorio donde se ubican las páginas Web del servidor Apache.

En este caso no se debe compilar, ejecutar, configurar o instalar algo adicional. Sin embargo, se puede configurar los temas de presentación o aspecto gráfico del ambiente Web que se han instalado mediante el archivo `php.conf`.

Una vez que el proceso de instalación ha concluido, se empieza el monitoreo utilizando una interfaz Web desde el URL:

http://localhost/gangliawebfrontend-2.5.5/

En la Figura 4-2 se presenta la página Web de inicio de la interfaz Web de Ganglia.



Fuente: [14], Capítulo 10, página 161

Figura 4-2. Interfaz Web de Ganglia

Un ejemplo y los resultados de monitoreo mediante *Ganglia* se presentan en el Capítulo 5.

4.3. HERRAMIENTAS PARA PLANIFICACIÓN DE TAREAS

Un *cluster* es una herramienta poderosa, sin embargo, la administración de su carga computacional es un reto sumamente importante. Podría ser que se ejecuten cientos o miles de tareas de muchos usuarios. Algunas tareas podrían ejecutarse sobre ciertos nodos debido a que no todos los nodos en un *cluster* son idénticos; así, algunos nodos poseen mayor capacidad de memoria que otros, por tal motivo existen nodos que no funcionan de acuerdo a lo esperado o sus resultados pueden ser relativamente lentos.

Algunos usuarios requieren prioridad de acceso a parte o a todos los recursos del *cluster*; sin embargo, algunas tareas pueden ejecutarse a cierta hora del día o solamente después de que otros tareas hayan finalizado.

Se puede generar un cuello de botella (*bottleneck*) si se requieren realizar diversas actividades mientras se incrementa la demanda de acceso a los recursos del *cluster*. Se utilizan herramientas de software que permiten realizar balanceo de carga; mediante estas herramientas se definen políticas locales de administración de recursos y tareas, que restringen y brindan el acceso a recursos, y minimizan el cuello de botella.

4.3.1. PRINCIPIOS DE BALANCEO DE CARGA

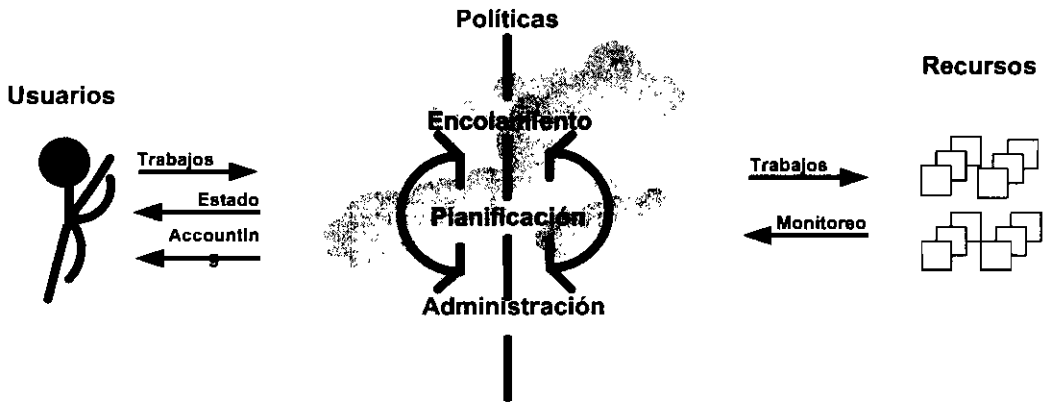
La administración del balanceo de carga permite que ciertas tareas se ejecuten de acuerdo a determinadas políticas para la utilización de los recursos del *cluster*. Existen algunos sistemas de administración de balanceo de carga como: Condor, PBS (*Portable Batch System*) y Maui.

La administración del balanceo de carga de los sistemas debe desempeñar las siguientes actividades:

- Manejo de colas por lotes.
- Planificación.
- Monitoreo.

- Administración de recursos.
- *Accounting* (Registro de usuarios).

La relación típica entre usuarios, recursos y otras actividades de balanceo de carga se presentan en la Figura 4-3, donde se aprecia que el software de balanceo de carga reside entre los usuarios del *cluster* y los recursos del mismo. Primero, los usuarios envían sus tareas a una cola. Luego, las tareas esperan en la cola hasta que sean programados para iniciarse en el *cluster*. La programación específica (cronológica) está definida por reglas basadas en políticas. Los mecanismos de administración de recursos manipulan los detalles del lanzamiento apropiado de las tareas y de limpieza, después de que el trabajo finalice o aborte. Mientras estos acontecimientos suceden, el sistema de administración y balanceo de carga monitorea el estado del sistema, sus recursos y el registro de cuentas de usuario *(accounting)* de quienes utilizan dichos recursos.



Fuente: [5], Capítulo 13, página 302

Figura 4-3. Actividades fundamentales para balanceo de carga

4.3.1.1. Manejo de Colas

Las tareas que el usuario desea que se realicen, se entregan al sistema de administración y balanceo de carga en un contenedor llamado trabajo por lotes (*batch job)*. El trabajo por lotes consiste de dos partes fundamentales: un conjunto de directivas de recursos (la cantidad de memoria o el número de CPUs necesarios), y una descripción de las tareas a ser ejecutadas. Esta descripción

contiene toda la información necesaria para que el sistema de administración y balanceo de carga inicie una tarea de usuario cuando el tiempo sea asignado.

La descripción de la tarea podría contener información relevante al nombre del archivo a ser ejecutado, una lista de archivos de datos requeridos para la tarea, y las variables de entorno o argumentos de líneas de comando a pasarse durante la ejecución.

Una vez realizada la petición de recursos al sistema de administración, los trabajos por lotes son colocados en una cola[12] hasta que los recursos solicitados estén disponibles.

Un aspecto importante de las colas es que se puede limitar su uso, por medio de restricciones de acceso a las mismas. Esto permite que el administrador del *cluster* tome el control total sobre la utilización de una política.

4.3.1.2. Planificación

La planificación o programación de tareas define el proceso de selección del trabajo para la ejecución. El seleccionar el mejor trabajo depende de la utilización de un conjunto de políticas administradas localmente, y la disponibilidad de balanceo de carga. Las políticas encapsulan todo aquello que se está utilizando en los recursos del *cluster*, y define el direccionamiento, prioridad de acceso, y control de tráfico que se debe mantener dentro del sistema.

La programación permite que se cumplan las políticas de acceso; es decir, la prioridad de acceso a recursos designado a un trabajo específico. Mientras se implementan las políticas, el planificador trata de optimizar recursos mediante la explotación de recursos sin utilización.

[12] Una cola se puede encontrar en servicios bancarios, autoservicios y otros lugares, y son útiles para realizar ciertas transacciones. Este principio se aplica a computadores y trabajos por lotes.

Para describir de forma más sencilla todo el proceso se propone el siguiente procedimiento:

- Seleccionar la mejor tarea a ejecutarse, de acuerdo a la política y disponibilidad de recursos.
- Iniciar la tarea.
- Detener la tarea y/o realizar un proceso de limpieza después de que la tarea ha concluido.
- Repetir el proceso.

4.3.1.3. Monitoreo

El monitoreo de recursos es parte de cualquier sistema de administración y balanceo de carga. Este provee la información necesaria a los administradores, usuarios y el sistema de planificación sobre el estado de tareas y recursos. El monitoreo de recursos juega un papel importante en tres instancias críticas:

- Cuando los nodos están ociosos, para verificar que están en funcionamiento antes de empezar otra tarea.

- Cuando los nodos están ocupados ejecutando una tarea. Los usuarios y administradores pueden chequear la utilización de ciertos recursos como memoria, CPU, red, I/O, y otros.

- Cuando una tarea finaliza. Se utiliza el monitoreo de recursos para asegurar que no existen procesos asociados a la tarea que concluye o que están realizando alguna actividad; finalmente el monitoreo permite verificar que un nodo esté en funcionamiento antes de empezar con una nueva tarea.

4.3.1.4. Administración de Recursos

La administración de recursos es responsable de iniciar, detener o realizar limpieza de tareas, después que las tareas fueron ejecutadas en los nodos del *cluster*.

Algunos sistemas de administración de balanceo de carga proveen mecanismos para asegurar el inicio y limpieza de tareas de manera satisfactoria y así mantener el estado interno de los datos del nodo; sin embargo, dichas tareas son inicializadas solamente en aquellos nodos que están disponibles y funcionando correctamente.

La administración de recursos permite la adición y/o eliminación de recursos computacionales, esto se debe a que los *clusters* no son estáticos, por ejemplo: se pueden añadir o eliminar nodos.

4.3.1.5. *Accounting*

El registro de datos de balanceo de carga se utiliza para varios propósitos, tales como:

- Generar reportes semanales de la utilización del sistema.
- Preparar reportes de utilización mensual de recursos por usuario.
- Sintonizar las políticas de planeamiento.
- Anticiparse a requerimientos computacionales futuros.
- Determinar áreas que deben ser mejoradas dentro del sistema.

4.3.2. CONDOR

Condor es un producto de *Condor Research Project* de la Universidad de Wisconsin, Madison y desarrollado por el Departamento de Ciencias de la Computación hace ya más de 10 años. Condor es un software de código abierto. Los paquetes, fuentes binarias, y documentación se pueden descargar desde el sitio Web:

http://www.cs.wisc.edu/condor

Cientos de organizaciones en la industria, el gobierno de EEUU y las universidades utilizan Condor para establecer ambientes de cómputo que superan el rango de cientos de estaciones de trabajo.

Condor es un sistema de administración especializado para monitorear y satisfacer necesidades computacionales en trabajos de cómputo intensivos. Este sistema provee un mecanismo de manejo de colas, políticas de planificación, esquema de prioridades, monitoreo de recursos, y administración de los mismos.

Los usuarios realizan peticiones a Condor, que luego son colocadas en una cola, en donde mediante un proceso de selección se establece en dónde y cuándo se ejecutarán.

Se ha realizado un breve resumen de las funcionalidades más importantes del sistema *Condor;* las cuales se describen a continuación:

- Directiva *ClassAds*: Esta directiva provee un marco de trabajo flexible y expresivo para determinar si solicitudes de acceso a recursos (trabajos) coinciden con los recursos ofrecidos por el sistema (computadores).

- Entrega distribuida: *Condor* no establece un computador central que reciba y entregue solicitudes de recursos (tareas). Las tareas se entregan desde varios computadores, donde cada uno posee su propia cola de trabajos.

- Prioridades de usuario: Los administradores pueden asignar prioridades a los usuarios mediante un mecanismo que habilita una política de compartición justa (*fair share*), orden estricto o una combinación de políticas [5].

- Prioridades de tareas: El orden de ejecución de las tareas de los usuarios se controla mediante asignación de prioridades.

- Dependencia de tareas: Algunas tareas no son independientes por tal motivo si existe un conjunto de tareas relacionadas se requiere un orden de inicialización de las tareas. Por ejemplo una tarea X se inicia solo si una tarea Y ha completado el trabajo que estaba realizando.

- Soporte de tareas simultáneas: Permite manipular tareas de tipo serial y paralelas con la incorporación de PVM y MPI.

- Puntos de verificación (*checkpoints*) y migración de tareas: Condor puede realizar puntos de verificación de manera transparente; es decir, proporciona la ilusión que la tarea se ejecuta normalmente. Un punto de verificación es como tomar una imagen instantánea (*snapshot*) del estado de la tarea. La tarea puede continuar su ejecución desde el punto de verificación. Un punto de verificación habilita la migración transparente de tareas desde un nodo a otro. Condor coloca un punto de verificación de una tarea, cuando éste programa los recursos a ser asignados a diferentes tareas o cuando un recurso es devuelto a su propietario. Mediante los puntos de verificación y la migración de tareas se provee una forma de tolerancia a fallos, que garantiza la utilización del tiempo de computación acumulado para una tarea; es decir, reduce las pérdidas ante eventos de fallos del sistema tales como apagado inadecuado o deficiencias del hardware.

- Suspensión y reanudación de tareas: Condor puede preguntar al sistema operativo si puede suspender o reanudar una tarea cuando se requiera. Para poder desempeñar estas tareas Condor utiliza reglas basadas en políticas.

- Autenticación y autorización: Condor permite tener autenticación de red utilizando una gran variedad de mecanismos; por ejemplo, Kerberos e ITU-T X.509, lo que permite la inclusión de certificados digitales basados en llave pública.

- Plataformas heterogéneas: Condor tiene soporte para sistemas Linux, UNIX y Windows.

- *Grid computing*: Condor incorpora funcionalidades basadas en computación *grid*. Condor incluye el software necesario para recibir tareas

de otros *clusters*, supercomputadores y sistemas distribuidos utilizando el *toolkit* **Globus** [22]. Condor puede entregar tareas mediante recursos administrados por otros sistemas de planificación tales como PBS utilizando **Globus**.

4.3.3. PLANIFICADOR MAUI

Maui fue desarrollado por el Centro de Computación de Alto Desempeño Maui, el Laboratorio Nacional del Noreste del Pacífico, el Centro de Supercomputación de San Diego y el Laboratorio Nacional Argonne.

Maui está soportado sólo en sistemas basados en UNIX [28]. A partir del año 2005 es desarrollado por la empresa Clusters Systems, Inc.

Maui es un planificador externo, es decir, no incluye un administrador de recursos pero extiende las funcionalidades de otros planificadores y administradores de recursos. Para obtener un sistema de información y dirigir las actividades de planificación, Maui utiliza APIs nativos de **OpenPBS** y de **PBSPro**. Por otra parte, es responsabilidad del administrador de recursos: la administración de nodos y el seguimiento de tareas. Maui controla la ejecución de las tareas.

Entre las funcionalidades que extiende Maui se tienen las siguientes:

- Definición de políticas, prioridades y configuraciones para tareas.
- Soporte para reservación de recursos.
- Soporte para calidad de servicio QoS.
- Permite diagnóstico detallado del sistema.
- Seguimiento extensivo de recursos y generación de estadísticas de utilización.

4.4. PLANIFICACIÓN CON PBS

PBS (*Portable Batch System*) es un sistema flexible de balanceo de carga y planificación de tareas, inicialmente fue desarrollado para administrar recursos computacionales de la NASA. PBS ha sido el líder en la administración de recursos y considerado el estándar de facto para los sistemas de planificación bajo sistemas Linux [5].

En el año de 1986 la NASA, junto al Centro de Investigación Ames [5] desarrolló el primer sistema de manejo de colas para el sistema operativo UNIX, denominado NQS (*Network Queueing System*). NQS en pocos años se convirtió en un estándar de facto para el manejo de colas por lotes. Una vez que los sistemas paralelos aparecieron, el sistema NQS se volvió inadecuado para manipular requerimientos complejos de administración de recursos.

La NASA lideró un intento por recopilar los requerimientos para un sistema de administración de recursos de siguiente generación. Estos requerimientos y funcionalidades de un sistema para administración de recursos fueron adoptados por la IEEE (*Institute of Electrical and Electronics Engineers*) en el estándar POSIX 1003.2d. Luego, en el año de 1999 la NASA diseñó un sistema que cumple con los requerimientos del estándar de la IEEE. Este sistema se denominó PBS el cual reemplazó rápidamente a NQS en los supercomputadores tradicionales y sistemas tipo servidor.

La empresa Veridian diseño el sistema PBS para la NASA, y luego, en Marzo de 2003, desarrolló una solución integral de administración de balanceo de carga, conocida como **PBS Pro**, cuya licencia fue adquirida por la empresa Altair Engineering, Inc.

Actualmente existen dos versiones de PBS: una versión antigua de código abierto **Altair OpenPBS** y la versión comercial **Altair PBS Pro**. Actualmente **OpenPBS** y **PBS Pro** se incluyen en algunos *toolkits* para la instalación automática de

clusters. **PBS Pro** provee algunas funcionalidades y beneficios para los administradores del *cluster*.

Las características más importantes del sistema PBS son:

- Múltiples interfaces de usuario: Proveen una interfaz gráfica de usuario para realizar peticiones por lotes y ejecución de tareas.

- Listas de seguridad y control de acceso: El administrador puede permitir o denegar el acceso al sistema PBS basándose en el nombre de usuario, grupo, nodo o dominio de red.

- Registro de tareas: *logs* detallados de las actividades del sistema mediante el análisis de utilización por usuario, por grupo y por nodo.

- Soporte de tareas paralelas: Permite la utilización de librerías de programación paralela como MPI, PVM, y HPF. Se puede planificar la ejecución de aplicaciones sobre un computador de un sólo procesador o mediante la utilización de múltiples computadores.

- Monitoreo del sistema: Mediante una interfaz gráfica permite realizar un monitoreo completo del ambiente distribuido.

- Soporte para *grids*: Provee tecnología para *grids* computacionales y la integración del *toolkit* **Globus**.

- API de desarrollo: Permite desarrollar aplicaciones que integran PBS como una de sus herramientas para requerimientos de balanceo de carga.

- Nivel de carga automático: Provee diversas formas de distribuir la carga en los computadores que conforman el *cluster*, basados en la configuración de hardware, disponibilidad de recursos, actividad del teclado y el manejo de políticas locales.

- *Distributed clustering*: Permite la disponibilidad de recursos de *clusters* y otros sistemas distribuidos en una red WAN.

- Ambiente común de usuario: Ofrece al usuario una visión común de las tareas entregadas y solicitadas, el estado del sistema y seguimiento de tareas.

- Prioridad de tareas: Permite a los usuarios especificar prioridades para la asignación de recursos y ejecución de sus tareas

- Disponibilidad para diferentes plataformas: Permite el soporte de Windows 2000 y XP, junto con la mayoría de versiones de UNIX y Linux, desde estaciones de trabajo y servidores hasta supercomputadores.

4.4.1. INSTALACIÓN, UTILIZACIÓN Y ADMINISTRACIÓN CON PBS

PBS está conformado por dos componentes: comandos de usuario y demonios del sistema.

PBS provee comandos por consola que se basan en la especificación POSIX 1003.2d. Estos comandos son utilizados para realizar peticiones, monitorear, modificar o eliminar tareas. Estos comandos se instalan en cualquier sistema que sea soportado por PBS. Existen tres clases de comandos: los primeros denominados comandos de usuario, que son comandos que cualquier usuario autorizado puede utilizar; los comandos del operador y del administrador, los cuales requieren privilegios de acceso específicos que se encuentran descritos en las secciones de seguridad de la guía del administrador de PBS [5].

El servidor de tareas, es el elemento principal de PBS que mantiene la cola y el registro de roles de administración de recursos. Se denomina **Server** y su proceso demonio es **pbs_server**. Todos los comandos y otros demonios de PBS se comunican con el **Server** mediante el protocolo IP. El **Server** tiene como función

principal la de proveer servicios básicos tales como recibir o crear lotes de tareas, modificar tareas, proteger el sistema frente a posibles fallos, y ejecutar tareas.

El demonio de ejecución de tareas coloca todas las tareas en una cola para su posterior ejecución. Este demonio se denomina **MOM** (*Machine Oriented Mini-Server*) debido a que es la madre de todos los procesos de ejecución. Su proceso demonio es el **pbs_mom**. **MOM** coloca un proceso en ejecución cuando recibe una copia de la tarea desde el **Server**.

Los demonios de **MOM**, de manera colectiva, son responsables de los roles de monitoreo y administración de recursos (y parte del registro) y de la administración y balanceo de carga.

El demonio de planificación o programación, denominado **pbs_sched**, implementa y asigna las políticas del sistema, que controlan cada una de los tareas en ejecución. El programador se comunica con varios **MOMs** para realizar peticiones de estado de los recursos del sistema, y se comunica con el **Server**, para conocer las tareas que van a ser ejecutadas. La interfaz para comunicarse con el **Server**, se lo realiza por medio del mismo API y con los comandos de usuario disponibles.

4.4.1.1. Instalación de PBS

Cuando PBS se instala en un *cluster*, un demonio **MOM** se ejecuta sobre cada nodo presente en la red. El **Server** y el planificador de tareas únicamente residen en uno de los sistemas o en el nodo de administración.

PBS se puede obtener como un paquete RPM o como una fuente binaria. Los comandos para la instalación de PBS son:

Para instalar PBS a partir de un paquete RPM:

```
# rpm -ivh OpenPBS-2.3.16.i386.rpm
```

Para instalar PBS a partir de una fuente binaria:

```
# tar -zxvf OpenPBS-2.3.16.tar.gz
# cd OpenPBS-2.3.16
[OpenPBS-2.3.16]# ./configure \
> --set default-server=server0 -enable-docs -with-scp
[OpenPBS-2.3.16]# make
[OpenPBS-2.3.16]# cd /usr/local/src/OpenPBS
[OpenPBS]# make install
[OpenPBS]# clean
```

Con **OpenPBS**, los archivos de instalación se deben colocar bajo el directorio `/usr/pbs` y los archivos de la cola bajo el directorio `/usr/pbs/spool/`.

4.4.1.2. Utilización de PBS

Una vez instalado PBS, el **Server** y los **MOMs** deben ser configurados y se debe seleccionar una política de planificación de eventos. Afortunadamente las implementaciones de PBS vienen preconfiguradas, sin embargo, a continuación se describen ciertas configuraciones opcionales.

4.4.1.2.1. Demonios PBS

PBS está conformado por tres procesos demonio: **pbs_server**, **pbs_sched**, y **pbs_mom**, los cuales son ejecutados únicamente por el administrador del sistema debido a que se inicializan mediante un valor conocido como UID (*User IDentifier*) que permite identificar al propietario del proceso de manera única.

Típicamente, todos los demonios PBS son iniciados de forma automática una vez que el sistema operativo arranca. El momento del reinicio del sistema operativo, se inicia o se detiene a PBS mediante un archivo de *script* que se encuentra en el directorio `/etc/init.d/pbs`. Este *script* lee el archivo `pbs.conf` para determinar que procesos en el *background* deben ser iniciados o detenidos.

El comando utilizado en un sistema Linux para inicializar o detener PBS tiene la siguiente sintaxis:

```
# service pbs  [ status | stop | start | restart ]
```

Se puede iniciar un servicio que esté disponible desde el arranque del sistema mediante el comando **chkconfig**:

```
# chkconfig pbs on
```

4.4.1.2.2. Direcciones de red y PBS

PBS utiliza los nombres de los nodos para identificar las tareas y su ubicación, por tal motivo, una instalación PBS conoce cuál es el nombre del nodo y sobre cuál nodo se está ejecutando el **Server**.

PBS utiliza los puertos, por defecto, desde el 15001 hasta el 15004; por tal motivo, es esencial que ningún *firewall* bloquee dichos puertos.

4.4.1.2.3. El comando Qmgr

El comando de administración de PBS, conocido como **qmgr**, provee una interfaz de línea de comandos. Una directiva en PBS se puede definir como una sentencia de ejecución y son ingresadas desde la línea de comandos. La sintaxis de cada directiva se chequea y se envía al **Server**. Una directiva **qmgr** puede tener cualquiera de las sintaxis que se muestran en la Figura 4-4.

```
comando  servidor  [nombre  [atributos opciones valores[,….]]
comando  cola  [nombre  [atributos opciones valores[,….]]
comando  nodo  [nombre  [atributos opciones valores[,….]]
```

Figura 4-4. Sintaxis de las directivas qmgr

A los recursos del sistema se los denomina objetos PBS; por ejemplo, un objeto PBS será un computador o nodo, el cual posee características propias como memoria, procesador, y otros.

En la Figura 4-4, `comando` es el nombre de la directiva que permite desempeñar una acción sobre un objeto. Las directivas **qmgr** se listan en la Tabla 4-2.

Directiva	Descripción
active	Coloca los objetos disponibles.
create	Crea un nuevo objeto.
delete	Destruye cualquier objeto.
set	Define o altera los valores y atributos de un objeto.
unset	Elimina los valores o atributos de un objeto.
list	Lista los valores y atributos actuales del objeto.
print	Imprime en pantalla todos los valores y atributos de los objetos de la cola y servidores.

Fuente: [16], Capítulo 16, página 380

Tabla 4-2. Directivas qmgr de PBS

Cualquier usuario puede ejecutar las directivas de listado e impresión de **qmgr**. La creación o eliminación de colas requiere privilegios de administrador de PBS. La configuración o eliminación de atributos del servidor o de las colas requieren privilegios de administrador o de operador.

4.4.1.2.4. Nodos

La interacción entre el planificador y el **Server** determina la ubicación de las tareas a ser ejecutadas; dicha interacción es afectada por el archivo PBS que contiene el listado de los nodos.

El **Server** no puede establecer una comunicación con los demonios del proceso **MOM**, sin una lista de nodos; y por lo tanto, **MOM** no será capaz de generar el reporte de información de las tareas de ejecución o notificar al **Server** cuando las

tareas se han completado. Se debe definir en el archivo PBS de nodos del **Server** a cada uno de los nodos. Por defecto, este archivo se ubica en el directorio: `/usr/spool/PBS/Server_priv`. Éste es un archivo de texto donde se especifica un nodo por línea. El formato del archivo de nodos PBS se muestra en la Figura 4-5:

```
nombre_del_nodo [ : ts ]    [ atributos ]
```

Fuente: [5], Capítulo 16, página 381

Figura 4-5. Formato de archivo de nodos PBS

El nombre del nodo es el nombre del computador en la red (*hostname).* El parámetro **ts** es opcional e indica que el nodo es de tiempo compartido (es decir que múltiples tareas pueden ejecutarse si los recursos necesarios están disponibles). Los nodos pueden tener atributos asociados a ellos. Estos atributos pueden ser de tres tipos: propiedades, nombre del nodo y recursos disponibles.

En la Figura 4-6, cada entrada de configuración es una línea con separaciones y espacios en blanco. Si la línea inicia con un signo de numeral (#), la línea es considerada como un comentario y es ignorada.

```
# Inner son nodos que pertenecen a un cluster.
# Outer son nodos que no pertenecen al cluster.
# moonless es el nombre del cluster
# La última entrada representa un nodo de tiempo
# compartido.

mercury     inner moonless
venus       inner moonless
earth       inner
mars        inner
jupiter     outer
saturn      outer
uranus      outer
neptune     outer
pluto:ts
```

Fuente: [5], Capítulo 16, página 382

Figura 4-6. Ejemplo de archivo de nodos de PBS

4.4.1.2.5. Configuración de MOM

El proceso de instalación crea un archivo de configuración básico de **MOM** que contiene un mínimo de entradas necesarias para ejecutar las tareas de PBS.

MOM es un demonio Linux, que puede ser iniciado, detenido y reiniciado después de realizar cualquier cambio en su archivo de configuración principal.

Cada entrada del archivo de configuración es una línea con separaciones y espacios en blanco. Si la línea inicia con un signo de numeral (#), la línea es considerada como un comentario y es ignorada.

El archivo de configuración de **MOM**, al menos debe contener la sintaxis de la Figura 4-7.

```
$logevent 0x1ff
$clienthost server-hostname
```

Fuente: [5], Capítulo 16, página 384

Figura 4-7. Archivo de configuración de MOM

En la primera línea, `logevent` especifica que los mensajes se enviarán al demonio **syslog**. La segunda línea, `clienthost` identifica al nodo que se comunica con **MOM**. En este ejemplo la entrada `server-hostname` representa el **Server**.

4.4.1.2.6. Configuración del planificador de tareas

El planificador es el responsable de implementar las políticas locales sobre las tareas que están ejecutándose y otros recursos.

OpenPBS y su planificador proveen un rango de políticas que permiten manipular las tareas de diferentes maneras, mediante la asignación de prioridades de

usuarios y grupos. Por defecto, el planificador presenta las políticas que se muestran en la Tabla 4-3.

Opciones	Valores por defecto
round_robin	False
by_queue	True
strict_fifo	False
load_balancing	False
fair_share	False
help_starving_jobs	True
backfill	Trae
backfill_prime	False
sort_queries	True
sort_by	shortest_job_first
smp_cluster_dist	Pack

Fuente: [5], Capítulo 16, página 385

Tabla 4-3. Políticas básicas de PBS

Una vez configurados e iniciados el Server y el planificador, las tareas del usuario pueden ser inicializadas colocando un atributo al **Server** con el valor de verdadero, como se muestra a continuación:

```
[root]# qmgr -c "set server scheduling=true"
```

4.4.1.3. Administración con PBS

Durante la instalación de PBS, en el directorio `/etc` se crea el archivo de configuración `pbs.conf`, donde se controlan a todos aquellos demonios de PBS que se ejecutan en el sistema local. Cada nodo perteneciente al *cluster* posee su propio archivo de configuración `pbs.conf`, con el objetivo de establecer las configuraciones por defecto o cambiar las requeridas durante la ejecución de una aplicación en particular.

4.4.1.3.1. Reportes y registros PBS

El servidor PBS mantiene un *log* de registro, el cual se almacena en el directorio `/usr/spool/PBS/Server_priv/accounting/`. El formato del archivo *log* está definido por el año, mes y día. Un nuevo archivo se genera por día durante el primer evento después de media noche (se ejecuta un suceso programado mediante el demonio **cron**).

El administrador de PBS puede utilizar el comando **pbs_report** para generar un reporte de utilización del sistema, usuarios, y tareas (incluye un análisis estadístico de tareas, y reportes de monitoreo del *cluster*). El programa extrae datos de archivos de registro (*logs*), y realiza las operaciones necesarias para generar el reporte.

4.4.1.3.2. Monitoreo mediante PBS

PBS provee una interfaz gráfica para desplegar la información disponible del ambiente de ejecución mediante el demonio **xpbsmon**. El ambiente gráfico de PBS muestra la lista de nodos que ejecutan uno o más servidores con tareas sobre uno o más nodos de ejecución.

En algunas ocasiones se debe sintonizar el estado del sistema o de una determinada tarea, o leer la información de los archivos de *logs*. Algunas herramientas gráficas permiten ver el progreso de una determinada tarea, las cuales se muestran en la Tabla 4-4.

Comandos para sintonizar tareas	
qstat	Muestra el estado de tareas y servidores.
xpbs	Alerta al usuario cuando una o más tareas se han completado.
tracejob	Muestra y ordena las entradas en los *logs* de PBS para tareas específicas.

Fuente: [5], Capítulo 16, página 387

Tabla 4-4. Herramientas de monitoreo de PBS

Mientras una tarea se ejecuta, el comando **qstat** se utiliza para verificar el estado de la misma. Puede utilizarse **tracejob** para verificar de principio a fin la tarea que realizó.

4.5. SISTEMAS DE ARCHIVOS PARALELOS: *PVFS*

Un sistema de archivos paralelos es un sistema de archivos distribuido que permite a distintas aplicaciones (paralelas y seriales) almacenar datos en servidores de discos distribuidos en red.

Los sistemas de archivos paralelos tiene como primer objetivo proveer acceso a I/O en general, de alto desempeño, debido a que puede existir una gran cantidad de datos, generados desde o para una aplicación paralela. Sin embargo, los sistemas de archivos paralelos no proveen desempeño óptimo para aplicaciones o programas seriales que realizan una sola tarea, y se debe a que estos sistemas fueron diseñados para propósitos diferentes; es decir, que no se pensaron para reemplazar al sistema de archivos NFS (*Network File System*) [14].

4.5.1. SISTEMA DE ARCHIVOS

Los sistemas *cluster* con frecuencia utilizan el sistema operativo Linux, instalado en cada uno de sus nodos. En Linux, todos los directorios y archivos se integran en un sistema de archivos[13] virtual, es decir, que todos los dispositivos físicos (disqueteras, discos duros, particiones de disco duro, dispositivos de CD-ROM, y otros) y sus sistemas de archivos se encuentran combinados en una estructura de árbol gigantesca.

[13] Un sistema operativo ofrece un interfaz para manipular archivos independientemente del dispositivo físico (por ejemplo el disco duro). A este interfaz se lo denomina sistema de archivos, y consiste de llamadas al sistema para crear, leer, borrar y escribir archivos.

El espacio de disco se asigna a particiones (ya sean físicas de **fdisk**[14] o lógicas de LVM - *Logical Volume Manager*[15], con o sin RAID) en una estructura lógica para que acoja los directorios y archivos (estructura de árbol).

Linux soporta gran cantidad de sistemas de archivos, algunos considerados nativos de este sistema (diseñados para él específicamente, o para otros sistemas UNIX y adaptados y adoptados ampliamente bajo Linux [22]), y otros propios de otros sistemas operativos (por ejemplo VFAT[16] de Windows 9X, FAT32 de Windows XP, NTFS de Windows NT/2003 o el HPFS de MAC).

4.5.2. DISTRIBUCIÓN DE DATOS

La intención es crear varios servidores, con réplicas exactas unos de otros, que sirvan todos un mismo contenido; además, se debe encontrar alguna forma de realizar estas réplicas automáticamente, de modo que para el usuario (en este caso, los desarrolladores o encargados de los contenidos) el *cluster* se comporte como un único computador, en el que ellos copian (o trabajan) los archivos en un lugar único, y el software de control del *cluster*, internamente se encargue de hacer llegar una copia a cada uno de los servidores que lo componen.

Para poder realizar dichas tareas se tienen dos estrategias: la replicación física de archivos, en la que cada servidor tendrá una copia de todos los datos en su disco duro; y la distribución de los datos mediante sistemas de archivos distribuidos, en los que un servidor de archivos y el resto de equipos del *cluster* accederán a sus contenidos por la red.

---

[14] **fdisk** es una herramienta que permite la creación, asignación y eliminación de particiones e integrada en sistemas operativos UNIX, Linux y Windows.

[15] LVM (*Logical Volume Manager*) es un subsistema para la gestión avanzada de unidades de almacenamiento en caliente (*hot swap*), que se ha convertido en un estándar de facto en varias implementaciones de UNIX y Linux.

[16] VFAT es un controlador de sistemas de archivos de Linux compatible con DOS, que permite montar e intercambiar datos entre sistemas de ficheros DOS y FAT en sistemas Linux.

4.5.2.1. REPLICACIÓN DE DATOS

La alternativa más primitiva para la distribución del contenido a servir a todos los equipos del sistema *cluster* es la replicación (automática o manual) de los archivos en todos los computadores. Por ejemplo, una forma de replicar los archivos sería tener en un servidor FTP central el contenido a replicar en los clientes, y que éstos, a una hora determinada, ejecuten un *script* (programado en el **cron** del sistema) que se encargue de conectarse al servidor y descargar todo el contenido.

El comando **rsync** optimiza en gran medida la cantidad de datos a transmitir por la red y, en consecuencia, el tiempo necesario para realizar la sincronización.

4.5.2.1.1. rsync

rsync es un programa para copiar archivos entre dos sistemas UNIX o Linux que, utilizando un algoritmo propio, para los archivos que ya existen en ambas máquinas es capaz de enviar de un equipo a otro tan sólo aquellas partes de los archivos que hayan sido modificadas, sincronizando de esta forma los contenidos de los dos equipos. Esta característica hace a **rsync** especialmente apropiado (frente a otros métodos como **rcp** o **ftp**) para mantener al día copias idénticas de directorios entre equipos geográficamente distantes (por ejemplo, réplicas de servidores, y otros). Con **rsync** se puede:

- Copiar o sincronizar archivos, directorios o sistemas de archivos enteros, manteniendo, si fuera necesario, enlaces, permisos y fechas.
- Redireccionar todo el tráfico mediante **ssh** para cifrarlo.

4.5.3. SISTEMAS DE ARCHIVOS DISTRIBUIDOS

Los sistemas de archivos distribuidos son la estrategia opuesta a la replicación; en lugar de llevar todos los datos físicamente a todos los servidores, se dejan en un equipo que hace de servidor central y los demás computadores acceden a ellos por la red según los necesiten.

4.5.3.1. NFS

El sistema de archivos de red de UNIX o Linux, denominado NFS (*Network File System*) permite compartir archivos y directorios en la red.

NFS fue desarrollado por SUN Microsystems en el año de 1984. La primera versión comercial de NFS se incluyó en el sistema operativo UNIX SUN Berkeley 4.2.

NFS está implementado bajo una arquitectura cliente-servidor, donde el servidor almacena los archivos y provee servicios a las peticiones de los clientes.

Por medio de este sistema de archivos distribuido, el *cluster* provee acceso transparente a los archivos de una aplicación paralela, y hace posible que estos archivos estén disponibles en todos los nodos del sistema.

NFS no es eficiente para los requerimientos de almacenamiento de ciertas aplicaciones paralelas de alto desempeño; por tal motivo, los sistemas de archivos diseñados para sistemas *cluster*, utilizan un criterio diferente: priorizan y optimizan el desempeño cuando manejan una gran cantidad de datos desde una aplicación paralela.

4.5.3.2. ALMACENAMIENTO COMPARTIDO DE DATOS

En un *cluster* se desea proveer el acceso a los mismos datos, por tal motivo se deben implementar vías para compartir dichos datos. Las implementaciones más importantes de almacenamiento de datos compartidos basados en hardware son NAS (*Network Attached Storage*) y SAN (*Storage Area Network*). Ambas alternativas proveen soluciones para mejorar el desempeño y la manipulación de datos paralelos.

Los sistemas NAS y SAN son muy costosos debido a que su implementación se la realiza sobre sistemas físicos; por tal motivo se diseñaron sistemas de archivos paralelos basados en software para sistemas *cluster*. A continuación, se

mencionan las características de los sistemas NAS y SAN, y luego se describen algunos sistemas de archivos paralelos basados en software.

4.5.3.2.1. Network Attached Storage

NAS [23] configura un servidor para recibir peticiones de acceso a archivos distribuidos en la red. NAS es el propietario del sistema de archivos, y se considera que es una extensión a un servidor de archivos.

Un dispositivo NAS está dedicado para mover los datos de un lado a otro, es decir, desde el disco duro a la red y viceversa. Esto reduce el *overhead* asociado a las operaciones de archivos que se tienen en aplicaciones de servidor (como servidores de correo, servidores Web, o servidores de base de datos).

Un dispositivo NAS puede considerarse como una tarjeta Ethernet integrada en un disco duro y algún software que permita la manipulación de archivos. Un dispositivo NAS es independiente de la plataforma; es decir, brinda servicios de almacenamiento de forma transparente para diferentes sistemas como Windows, UNIX o Linux.

4.5.3.2.2. Storage Area Network

SAN [24] conceptualmente se considera como una capa (híbrida de hardware y software) que se encuentra entre las aplicaciones de servidores y los dispositivos físicos de almacenamiento, que incluye dispositivos NAS, servidores de bases de datos y servidores de archivos tradicionales, o dispositivos de almacenamiento como discos duros, LVMs, RAIDs, particiones, y otros.

La principal característica de SAN es que se ejecuta como parte separada de la red del *cluster*, usualmente empleando tecnologías de red propietarias o bien conocidas como Gigabit Ethernet, Myrinet y otras. Por tal motivo SAN incluye tecnologías físicas de red de alto desempeño (mediante la utilización de canales de fibra).

4.5.3.2.3. ClusterNFS

ClusterNFS [25] está formado por un conjunto de parches para el servidor NFS. Los parches permiten montar[17] un único sistema de archivos / (*root*) sobre clientes que carecen de disco duro (*diskless*), utilizando un intérprete de nombres de archivo [14]. Los clientes deben ejecutar el demonio NFS. Es un proyecto que para el 2005 se encuentra bajo la administración del grupo SourceForge [12].

4.5.3.2.4. OpenAFS

El sistema de archivos distribuido Andrew fue originalmente desarrollado por la Universidad de Carnegie Mellon; para el 2005 su desarrollo está bajo el control de los Laboratorios IBM de Pittsburg. OpenAFS [18] es la implementación de código abierto del sistema de archivos Andrew [27], el cual ofrece una arquitectura cliente-servidor que permite migración de datos de forma transparente. Se puede considerar a OpenAFS como la mejor alternativa a NFS [14].

4.5.3.2.5. Coda

Coda [15] es un sistema de archivos distribuido desarrollado por la Universidad de Carnegie Mellon, vigente en el 2005 y aún bajo desarrollo [14]. Es un derivado del sistema de archivos Andrew.

El sistema de archivos distribuido Coda se ha desarrollado específicamente para brindar funcionalidades de computación móvil, es decir, con soporte de operaciones de desconexión[18].

Coda es considerado un sistema de archivos de alto desempeño. El alto desempeño se logra mediante la utilización de *cache* [14]. El *caching* es una

[17] En sistemas Unix un dispositivo físico se representa mediante un archivo especial bajo el directorio `/dev`. Para poder tener acceso a la información del dispositivo físico es necesario definir un punto de montaje (considerado un directorio dentro del sistema de archivos) mediante el comando **mount**.

[18] Si el computador se desconecta de la red, el sistema almacena una copia local de los archivos que se estaban utilizando. Estas copias se pueden seguir modificando. Una vez que se reestablece la conexión de red, Coda se encarga de actualizar los nuevos cambios realizados en la versión original del archivo. Este mecanismo es conocido como sincronización.

técnica que permite realizar una copia de archivos o partes de un archivo, que serán recuperados por el servidor Coda. Las copias de archivos son almacenadas en los clientes Coda el tiempo que sea necesario, hasta que se pueda verificar que los datos almacenados en el servidor coinciden con estas copias. De esta forma se reduce la cantidad de tiempo que toma el reiniciar un cliente Coda y se minimiza la cantidad de datos que se necesita transmitir en la red [21].

4.5.3.2.6. InterMezzo

InterMezzo [16] fue diseñado por la Universidad Carnegie Mellon e inspirado en el sistema de archivos distribuido Coda.

Intermezzo es un sistema de archivos distribuido cuya implementación es de código abierto, comúnmente se incorpora como parte del *kernel* de Linux desde la Versión 2.4.5 [21].

Intermezzo realiza copias de archivos sobre los cuales se están trabajando. Intermezzo almacena las copias de archivos en una partición del disco duro local, la cual se utiliza como *cache*. Luego, se actualizan los cambios en el archivo original mediante el archivo que se encuentra en *cache*. Este mecanismo de actualización de archivos es conocido como sincronización de archivos.

4.5.3.2.7. Lustre

El nombre Lustre [17] proviene de la unión de Linux y *cluster*. Lustre es un sistema de archivos distribuido, diseñado para trabajar sobre un *cluster* de grandes dimensiones (superior a los 10000 nodos). Desarrollado bajo licencia de Cluster File Systems, Inc. La última versión se lanzó en febrero del 2005 [24].

Este sistema de archivos se implementa mediante parches del *kernel* y es funcional solo en las versiones de *kernel* 2.4.X.

4.5.3.2.8. Parallel Virtual File System - PVFS

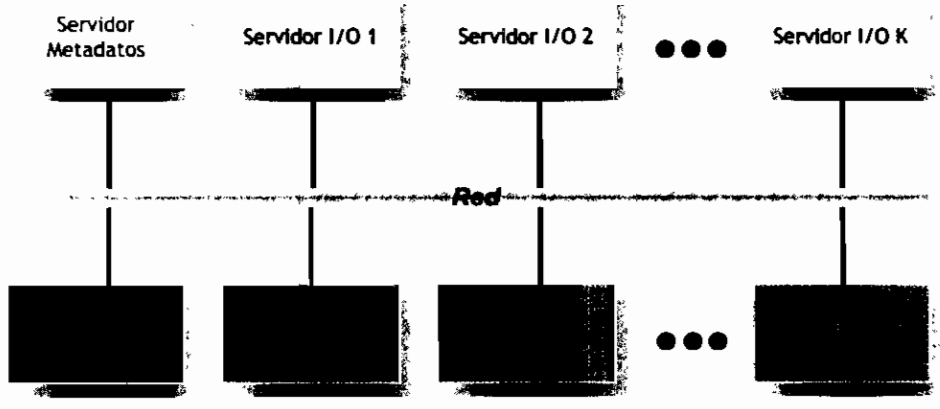
PVFS provee un sistema de archivos de alto desempeño. Este sistema de archivos se incluye con algunos *toolkits* para instalación automática de *clusters* (OSCAR y *Rocks* NPACI). Una explicación más detallada de PVFS se presenta en la Sección 4.5.4.

4.5.4. PVFS

PVFS es una implementación de código abierto del sistema de archivos paralelo que fue desarrollado por el Laboratorio Nacional Argonne y la Universidad de Clemson de los Estados Unidos, únicamente para sistemas Linux [23].

Entre las desventajas o las características que PVFS no presenta se tienen: no provee redundancia de datos, no tiene soporte simbólico o de enlaces forzados (similares a los accesos directos en sistemas Windows), y no provee una utilidad como **fsck** de Linux, que permita el chequeo del sistema de archivos.

La Figura 4-8 muestra la arquitectura para un *cluster* con soporte de PVFS. Los nodos de un *cluster* disponen de un servidor de metadatos para administrar, mantener o recopilar información en el sistema de archivos, tales como propietario del archivo, privilegios de acceso y ubicaciones de los datos.



Fuente: [14], Capítulo 12, página 213

Figura 4-8. Arquitectura interna de un *cluster*

Debido a que PVFS distribuye los archivos entre los nodos del *cluster*, los archivos se ubican en los dispositivos físicos de los servidores de I/O. Los servidores de I/O almacenan los datos utilizando el hardware existente y el sistema de archivos que posee cada nodo (los nodos pueden manejar diferentes sistemas de archivos, y puede ser más de uno[19], por ejemplo **ReiserFS**[20], o **ext3**[21]). Por medio de una técnica de distribución, se puede separar a un archivo en partes, y luego distribuir las partes entre múltiples nodos. De este modo, las aplicaciones tienen varias alternativas para llegar a los datos; es decir, un nodo de cómputo accede a una porción del archivo sobre un servidor, mientras otro nodo utiliza otra porción del archivo que está ubicado en un servidor I/O distinto. Esto reduce el cuello de botella generado al utilizar NFS. Los nodos pueden realizar cálculos computacionales y otras tareas mientras se realizan la distribución y separación de los datos.

Con PVFS, los nodos cliente pueden convertirse en servidores I/O y viceversa. El servidor de metadatos puede ser un servidor I/O[22], pero no puede convertirse en un nodo cliente. Los nodos clientes, los servidores de metadatos y de I/O pueden desempeñar tareas de cómputo y ejecución de otras aplicaciones.

4.5.5. INSTALACIÓN Y CONFIGURACIÓN DE PVFS

Se debe decidir el sistema de particionamiento del *cluster*, es decir que computador será el servidor de metadatos, cuales serán clientes, y que computadores serán los servidores I/O, debido que cada computador requiere diferente software y diferentes configuraciones.

---

[19] En sistemas UNIX el sistema de archivos virtual puede albergar varios sistemas de archivos, incluso de otros sistemas operativos.

[20] **ReiserFS** es un sistema de archivos diseñado exclusivamente para sistemas Linux por Hans Reiser. Utiliza una técnica conocida como *journaling*, la cual permite que toda acción de escritura sea primero añadida a un *journal* (un archivo o partición) y luego de ello sea realizada.

[21] **ext3** sistema de archivos por defecto en algunos sistemas Linux.

[22] Un nodo puede desempeñar los roles de servidor de metadatos, servidor de I/O y cliente a la vez, sin embargo, el sistema sólo puede tener configurado un servidor de metadatos, debido a que este mantiene toda la información del sistema de archivos y directorios (permisos, usuarios, y ubicación de los datos).

PVFS ejecuta un conjunto de demonios, librerías e incluye un módulo de *kernel* de Linux que permite la utilización de herramientas para el acceso a los archivos PVFS mediante el comando **mount**. PVFS consiste de 3 demonios. El primero es el demonio denominado **mgr** el cual mantiene los metadatos para el sistema de archivos. Una copia de este demonio se ejecuta por cada sistema de archivos dado, por ejemplo una copia para **ext3** y otra para **ReiserFS**. El segundo demonio se denomina **iod**, el cual recibe solicitudes y brinda servicios para leer y escribir archivos de datos. Uno de estos dos demonios se ejecuta en cada nodo, y convierte a un nodo en un nodo I/O. El tercer demonio es **pvfsd**, el cual trabaja en conjunto con el módulo del *kernel*, para realizar peticiones PVFS a nivel de *kernel*, es decir, permite definir un punto de montaje PVFS y proveer acceso transparente a las aplicaciones del cliente.

Existen dos alternativas para utilizar PVFS: la primera es mediante la interfaz del cliente PVFS; para crear el cliente se requiere las librerías necesarias para compilar aplicaciones PVFS (**libpvfs.so** y **libpvfs.a**) en los nodos de cómputo. La segunda alternativa se realiza mediante el módulo de *kernel* que, como se dijo anteriormente, permite definir el punto de montaje sin necesidad de escribir aplicaciones que integren PVFS.

A continuación se resume la instalación, configuración y utilización de PVFS mediante la segunda alternativa.

El paquete PVFS generalmente consta de dos fuentes binarias: **pvfs** (por ejemplo `pvfs-1.6.2.tgz`) y el parche para el *kernel* (por ejemplo `pvfs-kernel-1.6.2-linux-2.4.tgz`).

Para instalar PVFS a partir de una fuente binaria:

```
[root]# cp pvfs-1.6.2.tgz pvfs-kernel-1.6.2-linux-2.4.tgz /usr/src/
[root]# cd /usr/src/
[src]# tar -zxvf pvfs-1.6.2.tgz
[src]# mkdir pvfs
```

```
[src]# ln -s pvfs-1.6.2 pvfs
[src]# ls -lF
total 476
lrwxrwxrwx   1   root   root      15   Dec   14   17:42   pvfs -> pvfs-1.6.2
drwxr-xr-x   12  root   root     512   Dec   14   10:11   pvfs-1.6.2
-rw-r--r--   1   root   root  371535   Dec   14   17:41   pvfs-1.6.2.tgz
drwxr-xr-x   6   root   root    1024   Dec   14   10:10   pvfs-kernel-1.6.2-linux-2.4
-rw-r--r--   1   root   root  105511   Dec   14   17:41   pvfs-kernel-1.6.2-linux-2.4.tgz

[src]# cd pvfs
[pvfs-1.6.2]# ./configure
[pvfs-1.6.2]# make
[pvfs-1.6.2]# make install
[pvfs-1.6.2]# clean
```

Por defecto se instalan:

- **mgr**, **iod** en el directorio `/usr/local/sbin`.
- **libpvfs.a** en el directorio `/usr/local/lib`.
- Archivos **include** en el directorio `/usr/local/include`.
- Programas y utilitarios de prueba en el directorio `/usr/local/bin`.
- Páginas de ayuda en el directorio `/usr/local/man`.

Para instalar el módulo de PVFS para el *kernel* a partir de una fuente binaria:

```
# tar -zxvf pvfs-kernel-1.6.2-linux-2.4.tgz
# cd pvfs-kernel-1.6.2-linux-2.4
[pvfs-kernel-1.6.2-linux-2.4]# ./configure \
                                --with-libpvfs-dir=../pvfs/lib
[pvfs-kernel-1.6.2-linux-2.4]# make
[pvfs-kernel-1.6.2-linux-2.4]# make install
[pvfs-kernel-1.6.2-linux-2.4]# clean
```

Por defecto se instalan:

- **pvfsd** en el directorio `/usr/local/sbin`.
- Punto de montaje PVFS (**mount.pvfs**) en el directorio `/sbin/`.

El programa **mount.pvfs** se instala en el directorio `/sbin/` debido a que el comando **mount** de Linux busca por un ejecutable del sistema de archivos específico en dicho directorio.

4.5.5.1.1. Instalación y configuración de servidores y clientes PVFS

El servidor de metadatos se configura mediante tres archivos:

- El ejecutable **mgr**.
- El archivo `.iodtab`.
- El archivo `.pvfsdir`.

El ejecutable **mgr** es el demonio que provee servicios de metadatos en el sistema PVFS. Este se debe ejecutar como usuario `root` y debe ser iniciado antes de que los clientes tengan acceso al sistema.

El archivo `.iodtab` contiene una lista ordenada de direcciones IP y puertos para poder encontrar a los demonios I/O (**iods**). Una vez creada esta lista, no puede ser eliminada debido a que se perdería la integridad de los datos almacenados en PVFS, sin embargo, este archivo puede modificarse en caso de que se integren nuevos servidores I/O al sistema.

El archivo `.pvfsdir` describe los permisos de los directorios en los cuales se almacenan los metadatos.

Los archivos `.pvfsdir` y `.iodtab` se crean mediante el *script* **mkmgrconf**.

Los servidores de I/O tienen ejecutables y un archivo de configuración, distintos a los de los servidores de metadatos y clientes. Estos son:

- El ejecutable **iod.**
- El archivo de configuración `iod.conf`.

El ejecutable **iod** es el demonio que provee servicios de I/O en el sistema PVFS. Normalmente se ejecuta como usuario `root`, sin embargo, otros usuarios también pueden ejecutar dicho demonio.

El archivo `iod.conf` describe la ubicación del directorio de datos PVFS del nodo, y el usuario y grupo que ejecutan el demonio **iod**.

La configuración de los clientes se realiza mediante 6 archivos. Estos son:

- El ejecutable **pvfsd**.
- El módulo `pvfs.o`.
- El archivo de dispositivos `/dev/pvfsd`.
- El ejecutable **mount.pvfs**.
- El archivo `pvfstab`.
- El punto de montaje.

El ejecutable **pvfsd** es el demonio que realiza las transferencias de red frente a peticiones de los programas de cliente o puntos de montaje de clientes mediante el comando **mount**. Normalmente se inicia como usuario `root`. Éste debe ejecutarse antes de que el sistema de archivos PVFS sea montado en el cliente.

El módulo `pvfs.o` registra el sistema de archivos PVFS en el *kernel* de Linux, y permite el acceso a los archivos PVFS mediante llamadas al sistema. El archivo de dispositivos `/dev/pvfsd` es utilizado para simular un dispositivo físico que será utilizado por el módulo del *kernel* `pvfs.o` y el demonio **pvfsd**. El ejecutable **mount.pvfs** es utilizado por el comando **mount** para montar el sistema de archivos PVFS. El archivo `pvfstab` provee entradas similares al archivo `fstab`[23], que define puntos de montaje en el sistema de archivos PVFS y describe las aplicaciones que utilizan las librerías PVFS.

El punto de montaje es un directorio vació dentro de la jerarquía de árbol, generalmente creado bajo el directorio `/mnt/` como `/mnt/pvfs`.

[23] El archivo `/etc/fstab` de Linux define entradas para puntos de montaje de dispositivos físicos, que se montarán una vez que arranca el sistema.

4.5.6. UTILIZACIÓN DE PVFS

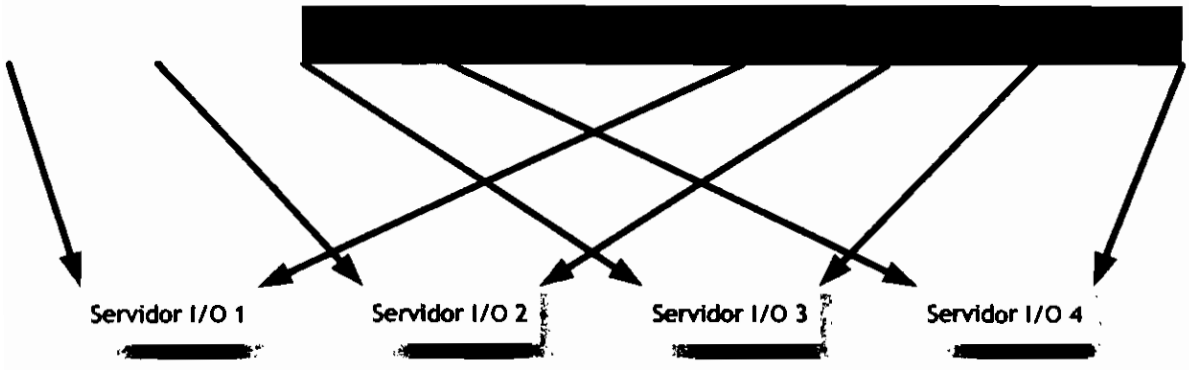
Para hacer efectivo el uso de PVFS, es necesario entender como PVFS distribuye los archivos en el *cluster*. PVFS utiliza un esquema de separación de los datos que consiste de tres parámetros:

- *base*: define el nodo del *cluster* donde se ubican los archivos en un inicio, y se identifican mediante un índice en el servidor I/O con el valor de 0.

- *pcount*: el número de servidores I/O entre los cuales se ha particionado un archivo.

- *ssize*: el tamaño de cada parte del archivo; cada parte puede estar constituida por bloques de datos contiguos; típicamente 64 KB. La idea es seleccionar un bloque de datos que permita optimizar el acceso de datos al archivo en un sistema paralelo.

La Figura 4-9 muestra como se distribuyen los datos para un sistema con 4 servidores I/O: el archivo es dividido en ocho partes y distribuido entre los cuatro servidores I/O; *base* es el índice del primer servidor I/O, *pcount* es el número de servidores utilizados (para el ejemplo cuatro) y *ssize* es el tamaño de cada uno de los ocho bloques.



Fuente: [5], Capítulo 12, página 223

Figura 4-9. División de archivos entre servidores I/O

Se puede examinar la distribución de un archivo utilizando el comando **pvstat**:

```
# pvstat data
data: base = 0, pcount = 5, ssize = 65536

# ls -l data
-rw-r--r--  1 root  root       10485760    Jun   03    12:49 data
```

Fuente: [5], Capítulo 12, página 226

Figura 4-10. Ejemplo de salida del comando pvstat

En la Figura 4-10 se observa la distribución de un archivo llamado *data*. El archivo *data* tiene un tamaño de 10485760 bytes, que fue dividido en 160 partes (104885760 bytes / 64 KB) entre 5 servidores I/O con 32 bloques de datos (160 partes / 5 servidores I/O), en cada servidor I/O.

Si se copia un archivo hacia un sistema de archivos PVFS utilizando **cp**, éste será particionado automáticamente para poder ser utilizado entre los nodos que conforman el sistema paralelo. Para un mejor control, se puede utilizar el comando **u2p**, que mediante la opción **–s**, permite especificar el tamaño del bloque de archivo; con la opción **–b** se puede especificar la base y mediante la opción **–n** decidir cuantos nodos están involucrados en el proceso. El comando **u2p** se utiliza para convertir un archivo serial en un archivo paralelo, y así permitir el acceso a programas o aplicaciones paralelas.

4.6. REFERENCIAS

[1] http://www.psionic.com/abacus/logcheck/
Último acceso: 12/02/2005

[2] http://swatch.sourceforge.net/
Último acceso: 21/12/2004

[3] http://www.linux-sxs.org/files/psionic/
Último acceso: 21/12/2005

[4] http://caspian.dotconf.net/menu/Software/LogDog/
Último acceso: 07/04/2005

[5] GROPP, William, LUSK, Ewing, STERLING, Thomas: *Beowulf Cluster Computing with Linux*. The MIT Press, Segunda Edición, Estados Unidos, 2003.

[6] http://bb4.com/
Último acceso: 21/12/2004

[7] http://clumon.ncsa.uiuc.edu/
Último acceso: 21/12/2004

[8] http://www.nagios.org/
Último acceso: 08/08/2005

[9] http://www.cs.mu.oz.au/~raj//parmon/
Último acceso: 21/12/2004

[10] http://oss.sgi.com/projects/pcp/
Último acceso: 21/12/2004

[11] http://www.acl.lanl.gov/supermon/
Último acceso: 21/12/2004

[12] www.sourceforge.net
Último acceso: 12/07/2005

[13] http://www.rrdtool.org/
Último acceso: 05/06/2005

[14] SLOAN, D., Joseph: *High Performance Linux Clusters, with OSCAR, Rocks, openMosix & MPI.* O' Reilly, Primera Edición, Estados Unidos, 2004.

[15] http://www.coda.cs.cmu.edu/index.html
Último acceso: 03/07/2005

[16] http://www.inter-mezzo.org
Último acceso: 03/07/2005

[17] http://www.lustre.org
Último acceso: 03/07/2005

[18] http://www.openafs.org
Último acceso: 03/07/2005

[19] http://www.perl.org/
Último acceso: 02/01/2005

[20] http://www.python.org
Último acceso: 02/01/2005

[21] http://people.ee.ethz.ch/~oetiker/webtools/mrtg/
Último acceso: 06/07/2005

[22] http://www.globus.org/toolkit/
Último acceso: 28/06/2005

[23] http://www.turbolinux.com/products/tlclb10/doc/html/en/user_guide/
Último acceso: 03/07/2005

[24] LUCKE, W., Robert: *Building Clustered Linux Systems.* Prentice Hall, Primera Edición, Estados Unidos, 2005.

CAPÍTULO 5. IMPLEMENTACIÓN DEL *CLUSTER* Y DESARROLLO DE UNA APLICACIÓN

5.1. EJEMPLOS DE APLICACIONES IMPLEMENTADAS

La clase de aplicaciones que un *cluster* puede solucionar suelen ser aplicaciones de gran exigencia o de súper computación. Las Aplicaciones de Gran Exigencia (GCA - *Grand Challenge Applications*) son problemas fundamentales en ciencia e ingeniería con un amplio impacto económico y científico. Muchas de estas aplicaciones pueden considerarse intratables sin el uso de computadores paralelos. Los requerimientos de recursos, como tiempo de procesamiento, memoria y necesidades de comunicación distinguen a las GCA.

Un ejemplo típico de un problema de gran exigencia es la simulación de algunos fenómenos que no pueden ser medidos a través de experimentos. Entre las GCA se incluyen problemas de cristalografía masiva y estructuras microtomográficas, dinámicas de proteínas y biocatálisis, química cuántica relativista de actínidos, diseño y procesamiento de materiales virtuales, modelamiento global de clima y simulación de eventos discretos.

A continuación se describen algunas aplicaciones que han sido desarrolladas para ser ejecutadas sobre *clusters*.

GAMESS (*General Atomic and Molecular Electronic Structure System*) es un paquete de química cuántica. Es mantenido por los miembros del grupo de investigación Gordon de la Universidad Estatal de Iowa. Para su compilación se requiere de un compilador Fortran 77. La librería de paso de mensajes se denomina DDI (*Distributed Data Interface*) y está implementada en C, por lo que

se requiere también un compilador C. **GAMESS** puede descargarse desde el sitio Web:

http://www.msg.ameslab.gov/GAMESS/GAMESS.html

NAMD (*Scalable Molecular Dynamics*) dispone de código para dinámica molecular paralela, diseñado para la simulación de sistemas biomoleculares. Se basa en un lenguaje paralelo de objetos denominado Charm++. Es posible descargar los archivos binarios o también el código fuente y proceder a compilarlo. Es mantenido por el Grupo de Biofísica Teórica del Centro Beckman de la Universidad de Illinois, Estados Unidos. **NAMD** puede descargarse desde el sitio Web:

http://www.ks.uiuc.edu/Research/namd/

NWChem (*High Performance Computational Chemistry Software*) es un paquete computacional para química diseñado para correr sobre supercomputadores paralelos o *clusters*. NWChem ha sido desarrollado por el grupo de Software de Ciencias Moleculares del Laboratorio Nacional del Noroeste del Pacífico, Estados Unidos (PNNL – *Pacific Northwest National Laboratory*). **NWChem** puede descargarse desde el sitio Web:

http://www.emsl.pnl.gov:2080/docs/nwchem/nwchem.html

AMBER (*Assisted Model Building with Energy Refinement*) es un paquete de programas para simulación molecular. No es gratuito. Una versión de demostración de **AMBER** se puede obtener en el sitio Web:

http://www.amber.ucsf.edu/amber/amber.html

BioBrew es una distribución Linux de código abierto basada en NPACI *Rocks*, permite implementar un *cluster* personalizado para computación bioinformática. Automatiza la instalación del *cluster*, incluye el software HPC necesario en un ambiente *cluster* y contiene algunas aplicaciones bioinformáticas. BioBrew puede ser descargado desde el sitio Web:

http://bioinformatics.org/biobrew

APBS (*Adaptive Poisson-Boltzman Solver*) ha sido desarrollado por el *National Biomedical Computation Resource* de la Universidad San Diego de California, Estados Unidos. Es una aplicación que resuelve ecuaciones usando el método Poisson-Boltzman para estudiar las propiedades electrostáticas de sistemas biomoleculares. **APBS** está disponible en un *roll* que puede instalarse sobre Rocks 4.0.0. Este *roll* simplifica la integración de **APBS** sobre un *cluster Rocks*. **APBS** puede ser descargado desde el sitio Web:

http://agave.wustl.edu/apbs

FLUENT es un programa para modelar flujos de fluidos. Está desarrollado en C y utiliza una arquitectura cliente/servidor. **FLUENT** es desarrollado por Fluent Inc. **FLUENT** puede ser descargado del sitio Web:

http://www.fluent.com

Marc es un programa para análisis de ingeniería en los campos de mecánica estructural, transferencia de calor y electromagnetismo. **Marc** puede descargarse del sitio Web:

http://www.marc.com

5.2. DISEÑO Y CONSTRUCCIÓN DEL *CLUSTER*

La implementación de un *cluster* se divide en tres fases: diseño, instalación y pruebas.

5.2.1. DISEÑO DEL *CLUSTER*

Los supercomputadores pueden proveer un nivel de procesamiento relativamente alto, sin embargo, su alto costo limita su uso. Por otro lado, es posible conseguir procesamiento paralelo mediante la interconexión de dos o más computadores personales; actualmente es posible intercomunicar éstos dispositivos mediante tecnologías de red de alta velocidad a un costo relativamente bajo. Bajo esta

estrategia, el procesamiento requerido por un programa al ser ejecutado sobre un procesador, puede asignarse en forma distribuida para ser ejecutado por varios elementos de cálculo, acelerando la obtención de resultados y minimizando el tiempo de espera para obtener la salida del mismo.

Esto motivó la idea de construir un *cluster* para procesamiento paralelo, basado en computadores personales con componentes conocidos como *commodities* e interconectados por medio de una red de alta velocidad (Gigabit Ethernet). Para probar las ventajas y funcionalidades del *cluster*, el presente proyecto conlleva la implementación de una aplicación que permita resolver las Cadenas de Markov, las cuales pueden requerir de una gran capacidad de procesamiento para su resolución.

El *cluster* implementado se deriva del proyecto Beowulf, utiliza una distribución de Linux como sistema operativo. La instalación del *cluster* se realiza de forma automática mediante las herramientas OSCAR y NPACI *Rocks*.

5.2.1.1. Hardware Utilizado

Para decidir que hardware se utilizaría para la implementación del *cluster* se tomaron en cuenta diversos factores entre los que se mencionan los siguientes:

- Costo.
- Desempeño.
- Soporte técnico.

Es importante notar que el punto más conflictivo se presenta en la elección del procesador a utilizar. Los dos primeros factores mencionados anteriormente, inclinan la elección hacia la arquitectura de los procesadores Athlon XP o Sempron XP de AMD; sin embargo, el ítem soporte técnico favorece notablemente a los procesadores Pentium 4 de Intel. Otro inconveniente que pueden presentar los procesadores de AMD es su capacidad para soportar los

efectos de la temperatura, estos procesadores se calientan demasiado llegando a niveles críticos para su funcionamiento[1].

La elección de la placa madre se realizó tomando en cuenta aspectos adicionales a los ya mencionados como: mejor rendimiento comparado con otras placas madres de la misma familia a la época de la adquisición[2], posibilidad de expandir su funcionalidad mediante ranuras para tarjetas adicionales (PCI, PCI-X, AGP, USB 2.0), posibilidad de incrementar la memoria agregando módulos de memoria adicional (entre 2 y 4 ranuras de memoria), soporte para procesadores[3] de mejores características, soporte para diferentes tipos de discos duros (PATA[4], SATA[5], RAID).

El disco duro se escogió tomando en cuenta algunos factores adicionales, los cuales se muestran en la Tabla 5-1, la cual compara distintas tecnologías de discos duros.

Características	PATA	SATA
Velocidad efectiva de transferencia de datos	133 MByte/s	150 MByte/s
Tamaño del conector	40 pines	7 pines
Detección de errores	Datos	Datos y comandos
Eliminación maestro-esclavo	No	Si
Native Hot-plug	No	Si

Tabla 5-1. Tabla comparativa entre tecnologías de discos duros

[1] Para mayor información acerca del calentamiento de los procesadores AMD se recomienda revisar los sitios Web: http://www.digit-life.com/articles/pentium4athlonxpthermalmanagement/, http://www.idg.es/pcworld/ShowSol.asp?ID=2418, http://www.pcmech.com/show/processors/33/.

[2] Abril de 2005.

[3] Soporte para procesadores como Pentium IV Xtreme Edition con soporte para tecnología Dual Core (la tecnología Dual Core provee mejores características para programación multitarea); Pentium IV, Celeron, Celeron D basados en un socket mPGA.

[4] PATA: *Parallel Advanced Technology Attachment.*

[5] SATA: *Serial Advanced Technology Attachment.*

Para el resto del hardware necesario se buscaron algunas de las mejores alternativas que ofrecía el mercado al momento de la elección; es el caso de las memorias y tarjeta de video.

Para la tarjeta de red se vio conveniente el adquirir una tarjeta con soporte para tecnología Gigabit Ethernet.

En la Tabla 5-2 se resume la información con respecto al hardware utilizado en los computadores, en la Tabla 5-3 se resume la información del hardware de red y en la Tabla 5-4 se resume la información de otros equipos utilizados para implementar el *cluster*.

Componente	Descripción
Placa madre	Intel D865PERL
Procesador	Pentium IV de 3.0 GHz con tecnología HyperThreading
Memoria	512 MB DDR
Tarjeta de red	CNET 10/100/1000 Mbps ProG-2000S
Tarjeta de video	ATI Radeon 9250
Disco duro	2 discos SATA de 120 GB

Tabla 5-2. Características de los computadores utilizados en el *cluster*

Componente	Descripción
Switch	CNET de 8 puertos 10/100/1000 Mbps. Modelo CGS-800.

Tabla 5-3. Características del hardware de red

Componente	Descripción
Switch KVM	DLink 4 puertos. Modelo DKVM-4K
Cables KVM	

Tabla 5-4. Características de otros componentes

Las aplicaciones del switch KVM son básicamente permitir la manipulación de un gran número de computadores con un sólo monitor, mouse y teclado. Cabe hacer notar que el switch no ejerce ningún control directo sobre los computadores que maneja, sólo sirve de interfaz.

5.2.1.2. Software Utilizado

El software utilizado se distribuye de forma libre. Se utilizaron varias distribuciones de Linux: Red Hat 9.0, Mandrake 10.0, Fedora Core 3.0, CentOS 4. Finalmente se instaló el *cluster* haciendo uso de Fedora Core 3.0 utilizando la herramienta de instalación automática de *clusters* OSCAR y CentOS 4.0 incluido en la herramienta de instalación automática de *clusters* NPACI *Rocks*.

5.2.2. INSTALACIÓN DEL *CLUSTER*

Para el presente proyecto se implementaron dos *clusters*: uno basado en OSCAR y otro basado en NPACI *Rocks*.

Los dos *clusters* están conformados por un nodo maestro y dos nodos esclavos.

Se utilizó uno de los discos duros de cada computador personal para instalar el *cluster* basado en NPACI *Rocks*, y el otro disco duro para instalar el otro *cluster* basado en OSCAR.

Se asignó el nombre: **aLeXAnDrE** a los dos *clusters*. El nodo maestro se denomina **dArThSerVer.local** en NPACI *Rocks* y **dArThSerVer.aLeXAnDrE** en OSCAR. Los nodos clientes se denominan **compute-0-0.local** y **compute-0-1.local** en NPACI *Rocks*, y **NodoOscar1.aLeXAnDrE** y **NodoOscar2.aLeXAnDrE** en OSCAR. En la Figura 5-1 se muestra la estructura física del *cluster* **aLeXAnDrE** construido con NPACI *Rocks* y en la Figura 5-2 se muestra el *cluster* construido usando OSCAR.

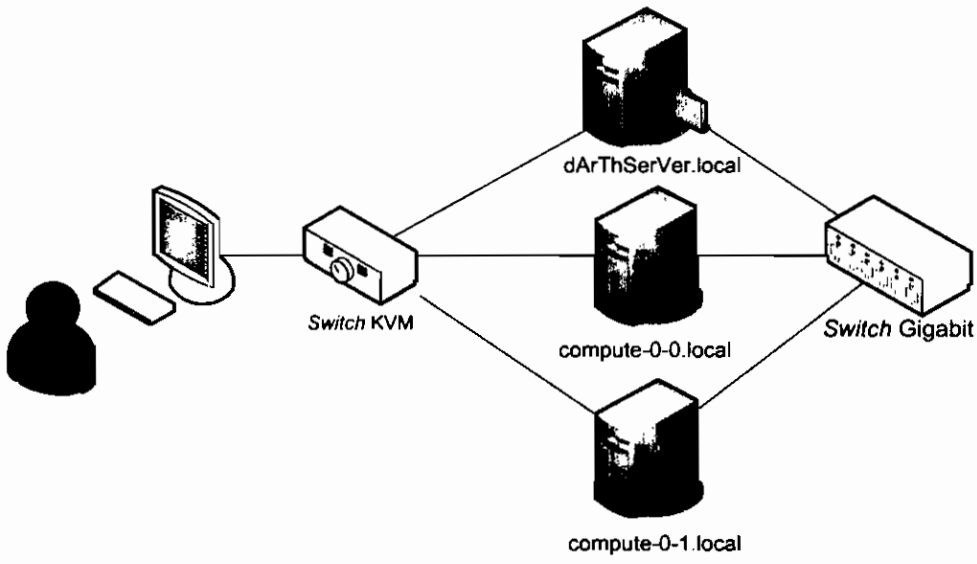


Figura 5-1. Estructura física del *cluster* aLeXAnDrE utilizando NPACI *Rocks*

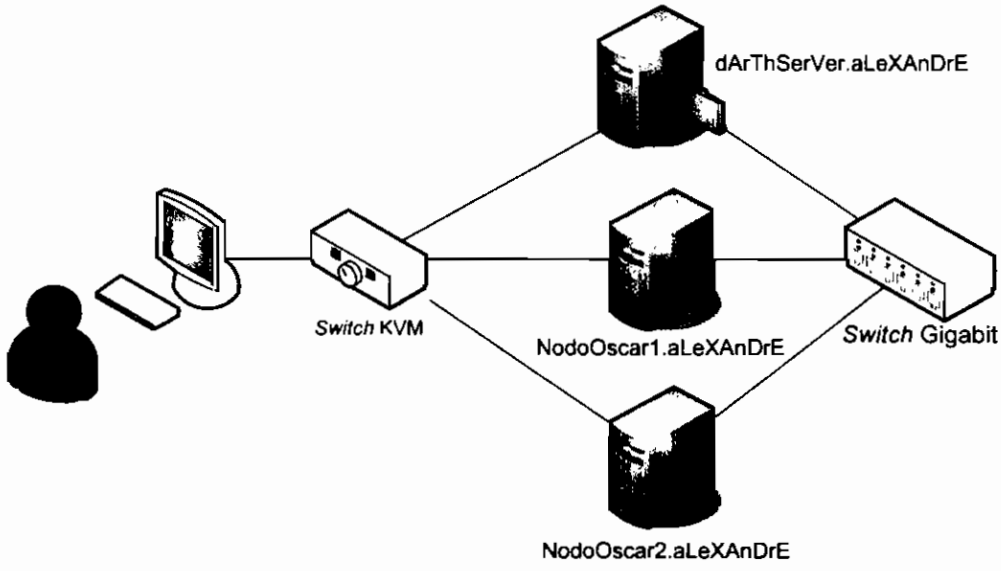


Figura 5-2. Estructura física del *cluster* aLeXAnDrE utilizando OSCAR

La instalación de OSCAR y de NPACI *Rocks* se describe en los ANEXOS 1 y 2, respectivamente.

5.2.2.1. Problemas Encontrados

Se encontraron algunos problemas en la instalación de OSCAR sobre las distribuciones de Linux: Red Hat 9.0, Mandrake 10.0 y Fedora Core 3.0, los cuales se presentan a continuación:

Con la configuración de hardware descrita en la Tabla 5-2, no es posible instalar OSCAR sobre Red Hat 9.0. Esta versión de la distribución Red Hat, no tiene soporte para discos duros SATA, ni tampoco dispone de los controladores para la tarjeta de red Gigabit Ethernet. Sin embargo, los controladores para la tarjeta de red pueden obtenerse desde el sitio Web de CNET, deben compilarse en Red Hat y copiar el módulo `e100.o` obtenido al compilar los controladores a la ubicación `/lib/modules/2.4.7-10/kernel/driver/net`; por el contrario, no es posible el tener soporte para los discos duros SATA, razón por la cual no se pudo utilizar la distribución Red Hat 9.0.

Con la configuración de hardware descrita en la Tabla 5-2 es posible instalar OSCAR sobre Mandrake 10.0 Official y sobre Mandrake 10.0 Community; sin embargo, OSCAR no está totalmente soportado en la versión Community de Mandrake 10.0. En las versiones Community y Official, no es posible instalar los clientes debido a que la imagen SIS no tiene soporte para los discos SATA, ni para las tarjetas de red.

Con la configuración de hardware descrita en la Tabla 5-2 es posible instalar OSCAR sobre Fedora Core 3.0; sin embargo, OSCAR 4.1 no está totalmente soportado, se encuentra en etapa de experimentación. El servidor se puede instalar sin ningún problema utilizando OSCAR 4.1 y un parche que se lo puede descargar desde el sitio Web de OSCAR; pero no es posible instalar los clientes debido a que la imagen SIS no tiene soporte para los discos SATA, ni para las tarjetas de red.

Se estaba utilizando la Versión 3.2.0 de NPACI *Rocks*, sin embargo esta versión no tiene soporte para MPI Versión 2; por lo que se instaló NPACI *Rocks* Versión 4.0.0.

Las soluciones empleadas para resolver los problemas encontrados en la instalación del *cluster* se presentan en las Secciones 5.2.2.2 y 5.2.2.3.

El proceso de instalación se simplifica si se utilizarían discos duros IDE.

5.2.2.2. Consideraciones adicionales en la instalación del Servidor

Tanto OSCAR como NPACI *Rocks* borran y formatean el contenido de los discos duros de los nodos clientes, por lo que es necesario realizar algunos procedimientos adicionales a los descritos en la instalación para conseguir instalar los dos sistemas sobre los equipos. Estos procedimientos se describen a continuación:

En el servidor, se debe desconectar el disco duro SATA ubicado en la segunda ranura para proceder a instalar *Rocks*. Se instala *Rocks* desconectando uno de los discos duros debido a que *Rocks* por defecto formatea todos los discos duros que se encuentran conectados y configura al resto de discos duros que se encuentren conectados para almacenar la información de los usuarios (directorio `/home`).

Una vez instalado Rocks, se conecta el disco de la segunda ranura para proceder a instalar Fedora Core 3.0. Se instala Fedora Core 3.0 en el segundo disco duro (disco secundario) teniendo en cuenta **el no formatear el disco duro primario**. Después de instalar Fedora Core 3.0, se procede a realizar el conjunto de pasos requeridos para instalar OSCAR de acuerdo al ANEXO 1. A continuación se mencionan algunas consideraciones adicionales a los pasos de instalación.

Para iniciar la instalación de OSCAR, debido a que Fedora Core 3.0 no está totalmente soportado, se debe descargar un parche que se encuentra disponible en la dirección Web:

http://www.bcgcs.ca/downloads/oscar/oscar4.1fc3.tar.gz

Antes de proceder con la ejecución del comando: `./install_cluster eth0`, se debe descomprimir el paquete del parche y ejecutarlo mediante el comando:

```
# ./oscar4.1fc3.sh
```

Luego se procede con la instalación. Después de realizar el paso 4, descrito en el ANEXO 1, se debe modificar de forma manual el archivo que contiene la información del particionamiento del disco duro y del sistema de archivos: `/var/lib/systemimager/images/oscarimage/etc/systemimager/autoinstallscript.conf`

Agregando la siguiente línea: `<boel devstyle="devfs" />`, antes de la sentencia `</config>`. `boel`[6] (*Brian's Own Embedded Linux*) es un sistema embebido linux que permite iniciar a los nodos clientes, la opción `devstyle`[7]`="devfs"` permite a SIS iniciar el sistema de archivos de los clientes antes de la instalación del gestor de arranque. El archivo de configuración modificado se presenta en el ANEXO F. Este cambio en el archivo se debe a que el soporte de OSCAR para Fedora Core 3.0 está en etapa de experimentación, mayor información sobre la modificación y ejecución del parche se la puede obtener en el sitio Web:

http://sourceforge.net/mailarchive/message.php?msg_id=11812982

Se debe ejecutar el siguiente comando[8]:

```
# mkautoinstallscript --image oscarimage --force --post-install reboot
```

Finalmente se procede a ejecutar el paso 5 descrito en el ANEXO 1.

La instalación de Fedora Core 3.0 eliminará la información del gestor de arranque que permitía que *Rocks* arranque, por lo que hay que agregar las entradas respectivas en el gestor de arranque de Fedora Core 3.0 para permitir que *Rocks* inicie. Fedora Core 3.0 utiliza por defecto el gestor de arranque denominado **grub**. En el archivo de configuración de **grub** `/etc/grub.conf` se deben agregar las entradas respectivas de *Rocks*.

---

[6] `boel` fue desarrollado por Bryan Finley, mayor información acerca de `boel` se puede obtener en el sitio Web: http://www.linuxjournal.com/article/5684.

[7] Mayor información acerca de la opción `devstyle` se la puede obtener del manual de **SystemImager** escrito por Bryan Finley, disponible en el sitio Web: www.systemimager.org.

[8] `mkautoinstallscript` es un comando de SIS que permite la creación del archivo requerido por SIS para realizar la instalación de los nodos clientes.

En el caso de utilizar el gestor de arranque LILO se deberían realizar los cambios respectivos en el archivo de configuración `/etc/lilo.conf`, y luego éste se debe instalar mediante el comando **lilo –v**.

En la Figura 5-3 se muestra el contenido del archivo `/etc/grub.conf`. Debido a que se utilizaron procesadores con soporte para HyperThreading, el programa de instalación crea automáticamente dos entradas por cada sistema operativo en el gestor de arranque. Una entrada (versión **-up**) tiene toda la funcionalidad excepto soporte para HyperThreading y la otra (versión **smp**) incluye la funcionalidad de HyperThreading. Los cambios realizados en este archivo se presentan en color azul.

```
# grub.conf generated by anaconda
#
# Note that you do not have to rerun grub after making changes to
# this file
# NOTICE:  You have a /boot partition.  This means that
#          all kernel and initrd paths are relative to /boot/, eg.
#          root (hd1,0)
#          kernel /vmlinuz-version ro root=/dev/sdb2
#          initrd /initrd-version.img
#boot=/dev/sda
default=0
timeout=20
splashimage=(hd1,0)/grub/splash.xpm.gz
hiddenmenu
title .:aLeXAnDrE:. - Rocks (CentOS -> 2.6.9-5.0.5.ELsmp)
      root (hd0,0)
      kernel /boot/vmlinuz-2.6.9-5.0.5.ELsmp ro root=LABEL=/ rhgb quiet
      initrd /boot/initrd-2.6.9-5.0.5.ELsmp.img
title .:aLeXAnDrE:. - Rocks-up (CentOS -> 2.6.9-5.0.5.EL)
      root (hd0,0)
      kernel /boot/vmlinuz-2.6.9-5.0.5.EL ro root=LABEL=/ rhgb quiet
      initrd /boot/initrd-2.6.9-5.0.5.EL.img
title .:aLeXAnDrE:. - OSCAR (Fedora Core 3.0 -> 2.6.9-1.667smp)
      root (hd1,0)
      kernel /vmlinuz-2.6.9-1.667smp ro root=LABEL=/1 rhgb quiet
      initrd /initrd-2.6.9-1.667smp.img
title .:aLeXAnDrE:. - OSCAR-up (Fedora Core 3.0 -> 2.6.9-1.667)
      root (hd1,0)
      kernel /vmlinuz-2.6.9-1.667 ro root=LABEL=/1 rhgb quiet
      initrd /initrd-2.6.9-1.667.img
```

Figura 5-3. Archivo de configuración del gestor de arranque grub del servidor

Una vez realizados los cambios en el gestor de arranque se debe resetear el computador para seleccionar el sistema operativo con el cual se desea trabajar: Fedora Core 3.0 para OSCAR o CentOS para NPACI *Rocks*.

Adicionalmente, debido a una capa de seguridad adicional que Fedora Core 3.0 dispone, no es posible desplegar los gráficos que genera Ganglia. Para solucionar este problema, se debe deshabilitar la capa de seguridad denominada SELINUX, modificando el archivo `/etc/selinux/config`. En este archivo se debe cambiar la configuración de SELINUX por `SELINUX=disabled`, y luego reiniciar el computador. Mayor información para desplegar los gráficos de Ganglia sin deshabilitar la capa SELINUX se puede encontrar en la página Web:

http://sourceforge.net/mailarchive/message.php?msg_id=10659480

5.2.2.3. Consideraciones adicionales en la instalación de los nodos cliente

El instalador de los nodos cliente de *Rocks* se encarga de formatear los discos duros que se encuentran en el computador. OSCAR es más flexible en su configuración y permite definir en cual de los discos duros realizar la instalación, sin embargo, también formatea los discos duros existentes. Debido a la facilidad de configuración de OSCAR y para evitar que *Rocks* borre la información de los discos duros es necesario instalar primero OSCAR.

Los computadores en los cuales se realizó la instalación no disponen de PXE, por lo cual para ser inicializados requieren de un CDROM o un disquete.

Antes de ejecutar el paso 5 de la instalación de OSCAR descrito en el ANEXO 1, debido al problema que existe con SIS, se debe compilar un *kernel* con soporte para las tarjetas de red Gigabit y para los discos duros SATA. Se puede encontrar mayor información acerca del problema de SIS y de la compilación del *kernel* en la página Web:

http://wiki.sisuite.org/SystemImager

SIS utiliza un conjunto limitado de controladores de dispositivos para crear un ambiente estable de instalación. SIS provee soporte para diferentes tipos de hardware; sin embargo, los controladores para las tarjetas de red Gigabit y los discos duros SATA no se incluyen en el *kernel* de instalación de SIS.

Una vez que se dispone del nuevo *kernel* se debe generar un disco de inicio para los clientes OSCAR, dependiendo de las características que se agreguen al nuevo *kernel*, es posible que este nuevo *kernel* sea demasiado grande para la capacidad de un disquete, por tal motivo puede ser necesaria la utilización de un CDROM. Una imagen ISO de un *kernel* precompilado y listo para su uso se puede descargar desde la página Web:

http://world.anarchy.com/~peter/systemimager/

Para el presente Proyecto de Titulación, se compiló el *kernel* de Fedora Core 3.0 para proveer soporte a las tarjetas de red y a los discos duros SATA. Se recomienda al lector revisar el Apéndice A del libro "*Red Hat Linux Customization Guide*" publicado por Red Hat, Inc. en el año 2003, y la Unidad VII de libro "*Linux System Administration I: Implementation*" publicado por IBM en el año 2004, para obtener mayor información de la configuración, compilación, e instalación del *kernel*.

El *kernel* compilado es utilizado por la imagen SIS y sólo se utiliza para cargar los controladores necesarios para poder instalar los clientes del sistema *cluster*. El *kernel* que los clientes utilizan una vez concluida la instalación es el mismo *kernel* que se encuentra instalado en el servidor, el cual si dispone de los controladores para discos duros SATA y las tarjetas de red Gigabit.

En el servidor se proceden a modificar los siguientes archivos, antes de instalar los nodos cliente:

- `/var/lib/systemimager/scripts/oscarimage.master`.
- `/var/lib/systemimager/image/oscarimage/etc/systemconfig/systemconfig.conf`.

- `/var/lib/systemimager/image/oscarimage/etc/systemimager/autoinstallscript.conf`.

En cada uno de los archivos mencionados en el párrafo anterior, se cambia la entrada que especifica en que disco duro se realizará la instalación, es decir se cambia **sda** por **sdb**. Estos archivos se presentan en el ANEXO F.

Además, en el archivo `oscarimage.master` se deben comentar las líneas que tengan la sentencia **modprobe**[9], debido a que el nuevo *kernel* no dispone de soporte en módulos por lo que el comando **modprobe** fallará. También se debe comentar la línea que contenga el comando **mkswap**[10], debido a que el *kernel* no incluye este comando. Sin embargo, una vez que se inicien los clientes debe ejecutarse **mkswap** para crear la partición *swap*[11] de los clientes.

Una vez modificados los archivos se procede a ejecutar el paso 6 de la instalación de OSCAR para lo cual se debe utilizar el CDROM creado con el nuevo *kernel* colocando los dos discos duros. OSCAR se instalará en el disco duro secundario.

Una vez que se instala todos los clientes OSCAR, se procede a terminar la instalación de OSCAR ejecutando los pasos 7 y 8 descritos en el ANEXO A.

Luego de la instalación de OSCAR se debe instalar *Rocks* sobre los clientes. Al instalar *Rocks* se deben desconectar los discos secundarios donde se instaló OSCAR y proceder de la forma descrita en el ANEXO B.

Una vez instalado *Rocks*, se vuelven a conectar los discos duros secundarios y se inicia el equipo utilizando el sistema operativo CentOS para proceder a modificar el archivo `/boot/grub/device.map` para agregar el nombre del segundo disco

[9] El comando **modprobe** permite cargar un módulo y sus dependencias. Un módulo es una pieza de código que agrega funcionalidad adicional al *kernel* de Linux en tiempo de ejecución.

[10] El comando **mkswap** se utiliza para preparar una partición para ser utilizada como *swap*.

[11] *Swap* es una particición física donde reside la memoria virtual que el sistema operativo Linux utiliza. Los datos son escritos en dicha partición cuando no hay suficiente espacio para almacenarlos en la memoria RAM.

duro. El archivo `device.map` se muestra en la Figura 5-4, y los cambios realizados en este archivo se presenta en color azul.

```
# this device map was generated by anaconda
(fd0)     /dev/fd0
(hd0)     /dev/sda
(hd1)     /dev/sdb
```

Figura 5-4. Archivo de configuración `device.map` de los clientes

Rocks eliminará la información del gestor de arranque de OSCAR y se debe volver a ingresar las entradas en el archivo de configuración del gestor de arranque `/boot/grub/grub-orig.conf` para disponer de ambos sistemas operativos. El archivo de configuración de los clientes se muestra en la Figura 5-5.

```
# grub.conf generated by anaconda
#
# Note that you do not have to rerun grub after making changes to
# this file
default=0
timeout=20
splashimage=(hd0,0)/boot/grub/splash.xpm.gz
hiddenmenu
title .:aLeXAnDrE:. - Rocks (CentOS -> 2.6.9-5.0.5.ELsmp)
      root (hd0,0)
      kernel /boot/vmlinuz-2.6.9-5.0.5.ELsmp ro root=LABEL=/
      initrd /boot/initrd-2.6.9-5.0.5.ELsmp.img
title .:aLeXAnDrE:. - Rocks-up (CentOS -> 2.6.9-5.0.5.EL)
      root (hd0,0)
      kernel /boot/vmlinuz-2.6.9-5.0.5.EL ro root=LABEL=/
      initrd /boot/initrd-2.6.9-5.0.5.EL.img
title .:aLeXAnDrE:. - Oscar (Fedora Core 3.0 -> 2.6.9-1.667smp)
      root (hd1,0)
      kernel /vmlinuz-2.6.9-1.667smp ro root=/dev/sdb6
      initrd /sc-initrd-2.6.9-1.667smp.gz
title .:aLeXAnDrE:. - Oscar-up (Fedora Core 3.0 -> 2.6.9-1.667)
      root (hd1,0)
      kernel /vmlinuz-2.6.9-1.667 ro root=/dev/sdb6
      initrd /sc-initrd-2.6.9-1.667.gz
```

Figura 5-5. Archivo de configuración `grub-orig.conf` de los clientes

Luego se procede a reiniciar los computadores y los dos sistemas estarán totalmente funcionales.

5.2.3. PRUEBAS DEL *CLUSTER*

5.2.3.1. Pruebas realizadas con OSCAR

En el caso del *cluster* basado en OSCAR, se realizó el paquete de pruebas que incluye OSCAR, en el cual se prueba la funcionalidad de LAM/MPI, MPICH y PVM que se realiza al ejecutar el paso 8 del asistente OSCAR descrito en el ANEXO A. El *cluster* pasó de forma exitosa todas las pruebas realizadas. En la Figura 5-6 se muestran las salidas obtenidas al ejecutar el paso 8 del asistente. Se realizan dos tipos de pruebas, una realizada por el usuario *root* y la otra realiza por un usuario normal. Como usuario *root* se prueban servicios como PBS, Maui y NFS. Como usuario normal se realiza conexiones SSH entre el servidor y cada uno de los nodos, y desde los nodos hacia el servidor; se prueba la funcionalidad de LAM/MPI, MPICH y PVM al ejecutar una aplicación de prueba; y se obtienen algunas métricas mediante Ganglia. Los archivos para realizar estas pruebas se encuentran ubicados en la carpeta `/opt/oscar/testing`.

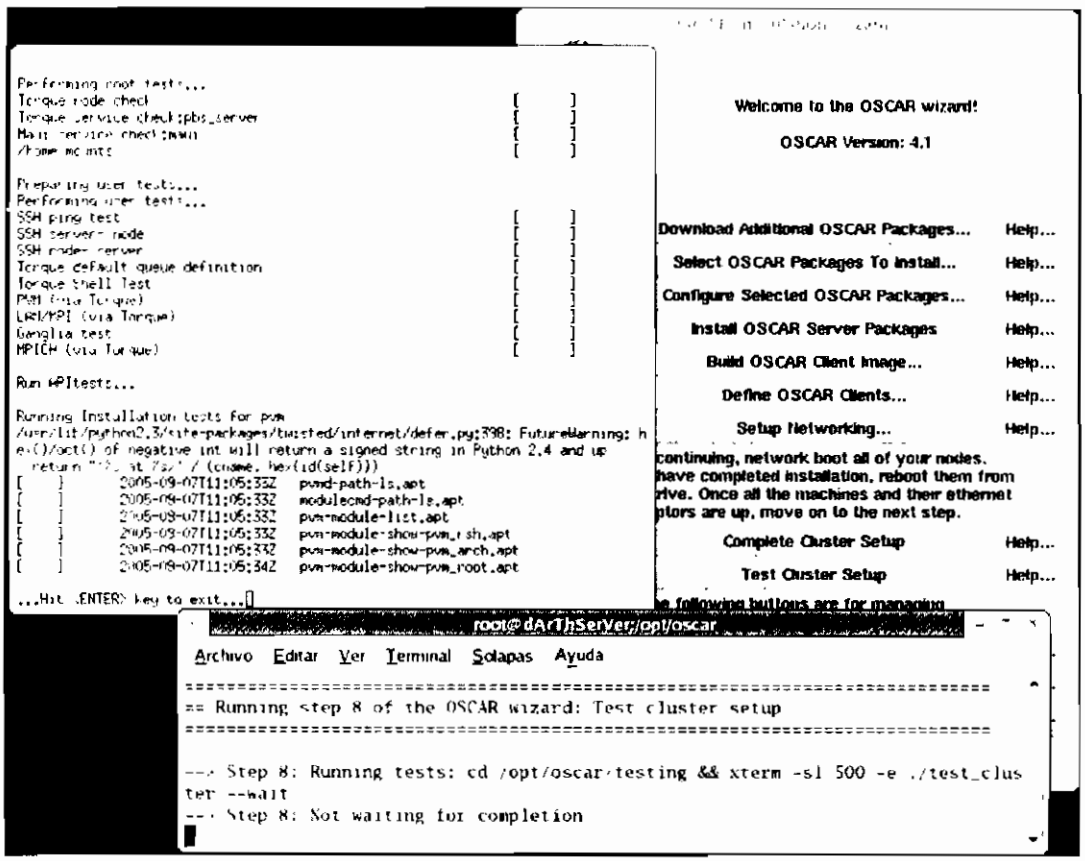


Figura 5-6. Resultados de la ejecución de las pruebas de OSCAR

Además, se realizó una pequeña aplicación con MPI que crea un proceso e informa en que nodo se creó dicho proceso. El código fuente del programa se muestra en la Figura 5-7.

```
#include "mpi.h"
#include <stdio.h>

int main (int argc, char * argv[])
{
        int ProcessID;
        int NoProcess;
        int NameSize;
        char ComputerName[MPI_MAX_PROCESSOR_NAME];

        MPI_Init(&argc, &argv);
        MPI_Comm_size(MPI_COMM_WORLD, &NoProcess);
        MPI_Comm_rank(MPI_COMM_WORLD, &ProcessID);
        MPI_Get_processor_name(ComputerName, &NameSize);
        fprintf(stderr, "HoLa DeSdE eL PrOcEsO: %d DeL nOdO: %s \n",
                ProcessID, ComputerName);
        MPI_Finalize();

        return 0;
}
```

Figura 5-7. Código fuente de una aplicación de MPI que muestra el nombre del proceso y del nodo

Los resultados obtenidos al ejecutar el código mostrado en la Figura 5-7 con 3, 5, 7 y 9 procesos se muestra en la Figura 5-8.

```
root@dArThServVer:/home/oscartst
Archivo  Editar  Ver  Terminal  Solapas  Ayuda
[root@dArThServVer oscartst]# mpirun -np 3 hola
HoLa DeSdE eL PrOcEsO: 0 DeL nOdO: dArThServVer.aLeXAnDrE
HoLa DeSdE eL PrOcEsO: 1 DeL nOdO: NodoOscar1.aLeXAnDrE
HoLa DeSdE eL PrOcEsO: 2 DeL nOdO: NodoOscar2.aLeXAnDrE
[root@dArThServVer oscartst]# mpirun -np 5 hola
HoLa DeSdE eL PrOcEsO: 0 DeL nOdO: dArThServVer.aLeXAnDrE
HoLa DeSdE eL PrOcEsO: 3 DeL nOdO: NodoOscar1.aLeXAnDrE
HoLa DeSdE eL PrOcEsO: 4 DeL nOdO: NodoOscar2.aLeXAnDrE
HoLa DeSdE eL PrOcEsO: 2 DeL nOdO: NodoOscar2.aLeXAnDrE
HoLa DeSdE eL PrOcEsO: 1 DeL nOdO: NodoOscar1.aLeXAnDrE
[root@dArThServVer oscartst]# mpirun -np 7 hola
HoLa DeSdE eL PrOcEsO: 0 DeL nOdO: dArThServVer.aLeXAnDrE
HoLa DeSdE eL PrOcEsO: 1 DeL nOdO: NodoOscar1.aLeXAnDrE
HoLa DeSdE eL PrOcEsO: 4 DeL nOdO: NodoOscar2.aLeXAnDrE
HoLa DeSdE eL PrOcEsO: 5 DeL nOdO: NodoOscar1.aLeXAnDrE
HoLa DeSdE eL PrOcEsO: 2 DeL nOdO: NodoOscar2.aLeXAnDrE
HoLa DeSdE eL PrOcEsO: 6 DeL nOdO: NodoOscar2.aLeXAnDrE
HoLa DeSdE eL PrOcEsO: 3 DeL nOdO: NodoOscar1.aLeXAnDrE
[root@dArThServVer oscartst]# mpirun -np 9 hola
HoLa DeSdE eL PrOcEsO: 0 DeL nOdO: dArThServVer.aLeXAnDrE
HoLa DeSdE eL PrOcEsO: 6 DeL nOdO: NodoOscar2.aLeXAnDrE
HoLa DeSdE eL PrOcEsO: 2 DeL nOdO: NodoOscar2.aLeXAnDrE
HoLa DeSdE eL PrOcEsO: 7 DeL nOdO: NodoOscar1.aLeXAnDrE
HoLa DeSdE eL PrOcEsO: 8 DeL nOdO: NodoOscar2.aLeXAnDrE
HoLa DeSdE eL PrOcEsO: 5 DeL nOdO: NodoOscar1.aLeXAnDrE
HoLa DeSdE eL PrOcEsO: 4 DeL nOdO: NodoOscar2.aLeXAnDrE
HoLa DeSdE eL PrOcEsO: 1 DeL nOdO: NodoOscar1.aLeXAnDrE
HoLa DeSdE eL PrOcEsO: 3 DeL nOdO: NodoOscar1.aLeXAnDrE
[root@dArThServVer oscartst]#
```

Figura 5-8. Resultados obtenidos con OSCAR al ejecutar la aplicación de prueba de MPI

También se ejecutó el código presentado en la Sección 3.4.8. Los resultados obtenidos con 3, 5, 7 y 9 procesos se muestran en la Figura 5-9.

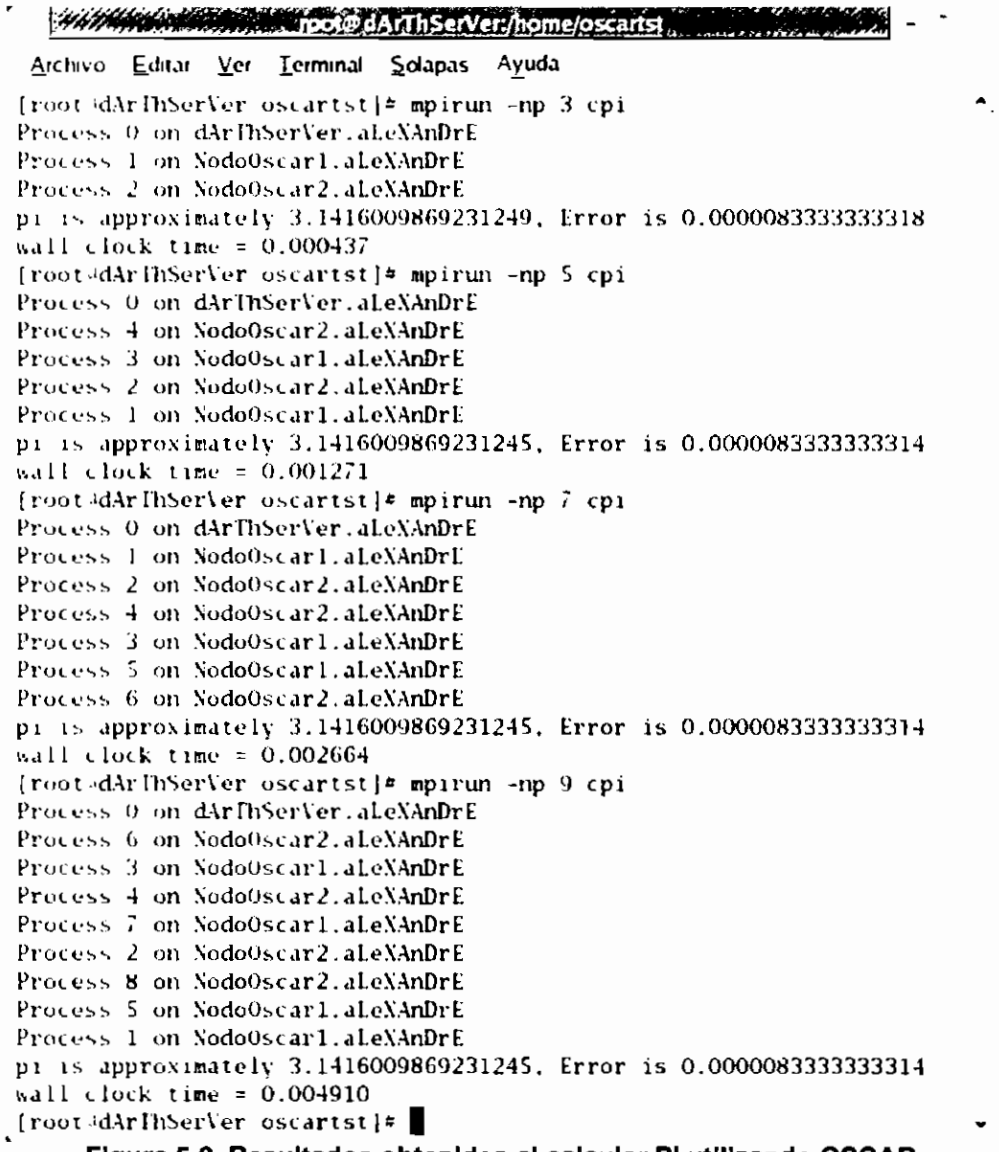


Figura 5-9. Resultados obtenidos al calcular PI utilizando OSCAR

Adicionalmente, se presentan las pantallas obtenidas mediante Ganglia. En la Figura 5-10 se muestra la pantalla principal de Ganglia, en la cual se puede apreciar el uso de la red del *cluster*, de la carga del *cluster*, de la memoria usada y del procesamiento realizado. Adicionalmente, se puede observar la carga de los nodos.

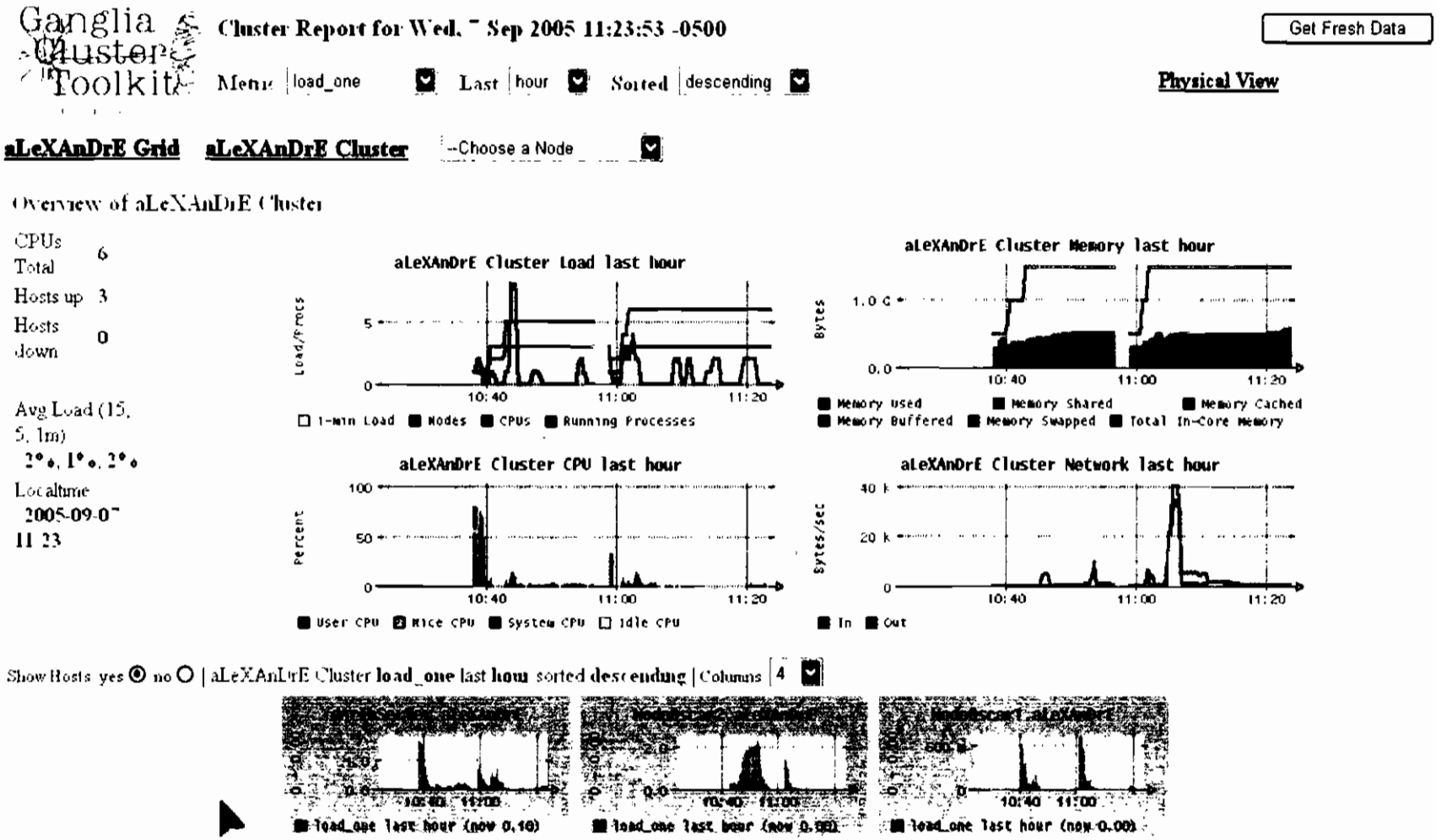


Figura 5-10. Información de Ganglia obtenida en OSCAR

En la Figura 5-11 se muestra una explicación de la información obtenida mediante Ganglia.

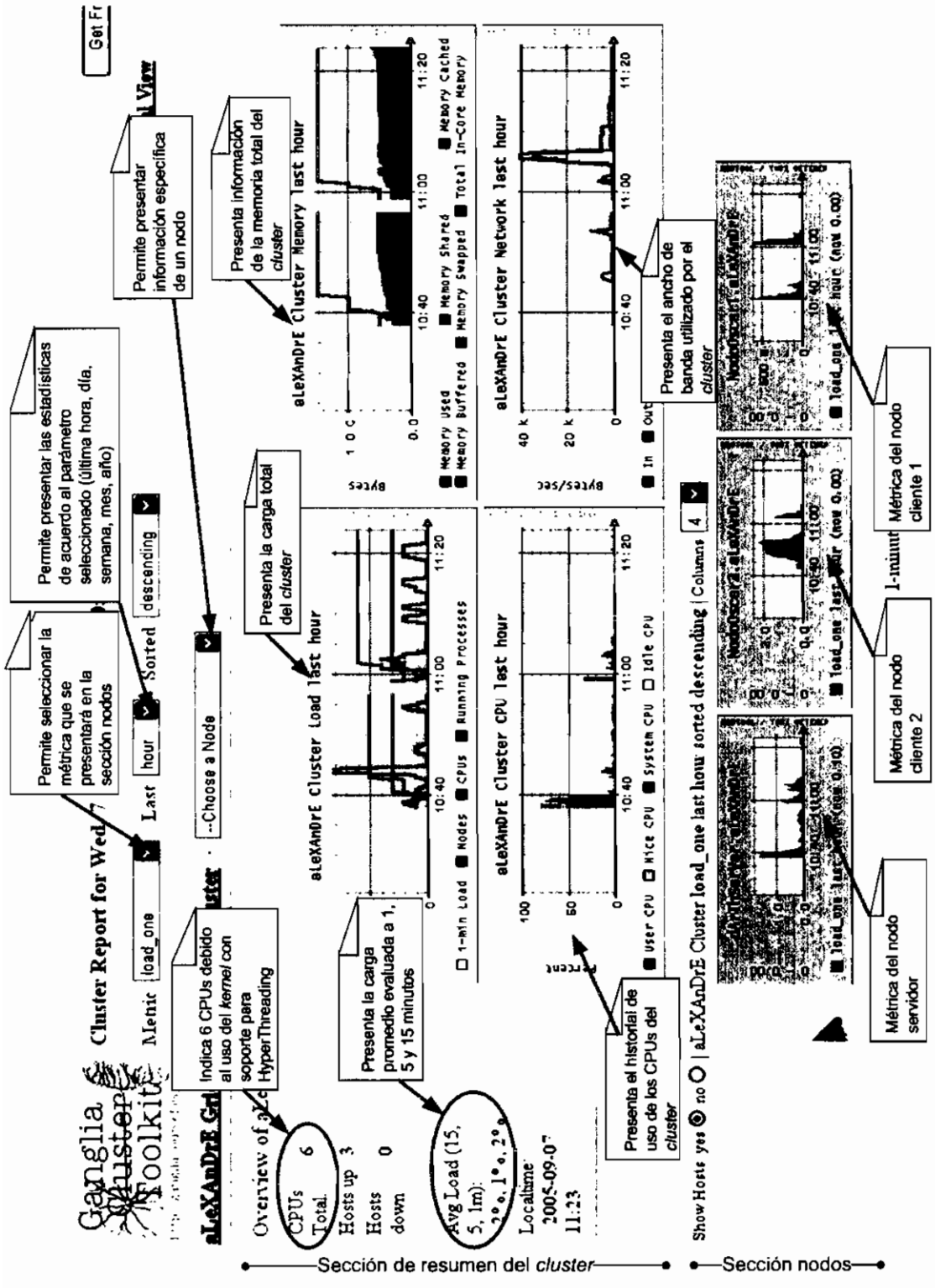


Figura 5-11. Breve explicación de la pantalla de Ganglia obtenida en OSCAR

Por otro lado, se muestran los resultados de la ejecución del comando **cexec**, el cual permite ejecutar un comando Linux en todos los nodos del *cluster*. En la Figura 5-12 se muestra la ejecución del comando **cexec** para listar el contenido de los directorios `/opt` de cada uno de los nodos.

Figura 5-12. Utilización del comando cexec para listar el contenido del directorio `/opt` de los nodos

5.2.3.2. Pruebas realizadas con Rocks

En el caso del *cluster* basado en *Rocks*, se realizó una prueba para comprobar la funcionalidad de MPI. Se utilizó el código fuente mostrado en la Figura 5-7. Los resultados obtenidos con 3, 5, 7 y 9 procesos se muestran en la Figura 5-13.

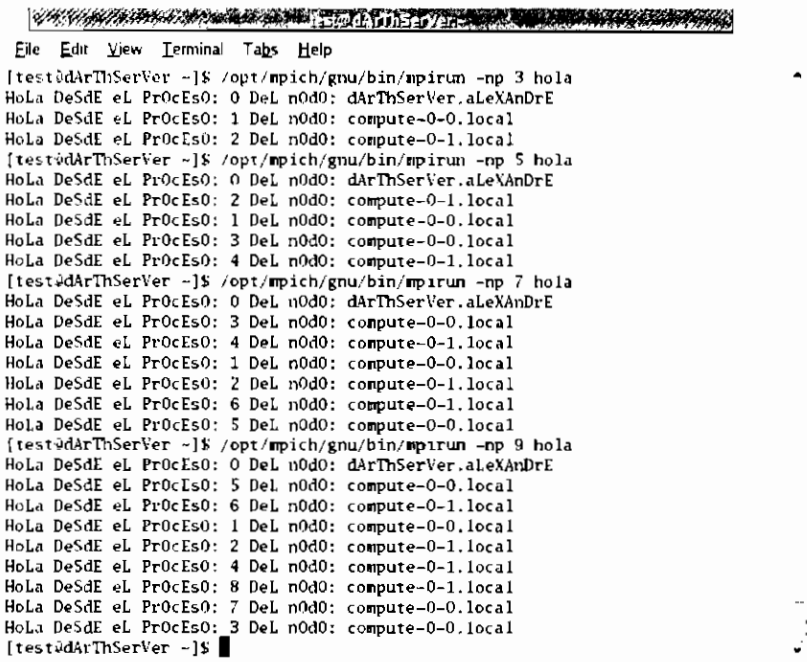


Figura 5-13. Resultados obtenidos con *Rocks* al ejecutar la aplicación de prueba de MPI

También se ejecutó el código presentado en la Sección 3.4.8. Los resultados obtenidos con 3, 5, 7 y 9 procesos se muestran en la Figura 5-14.

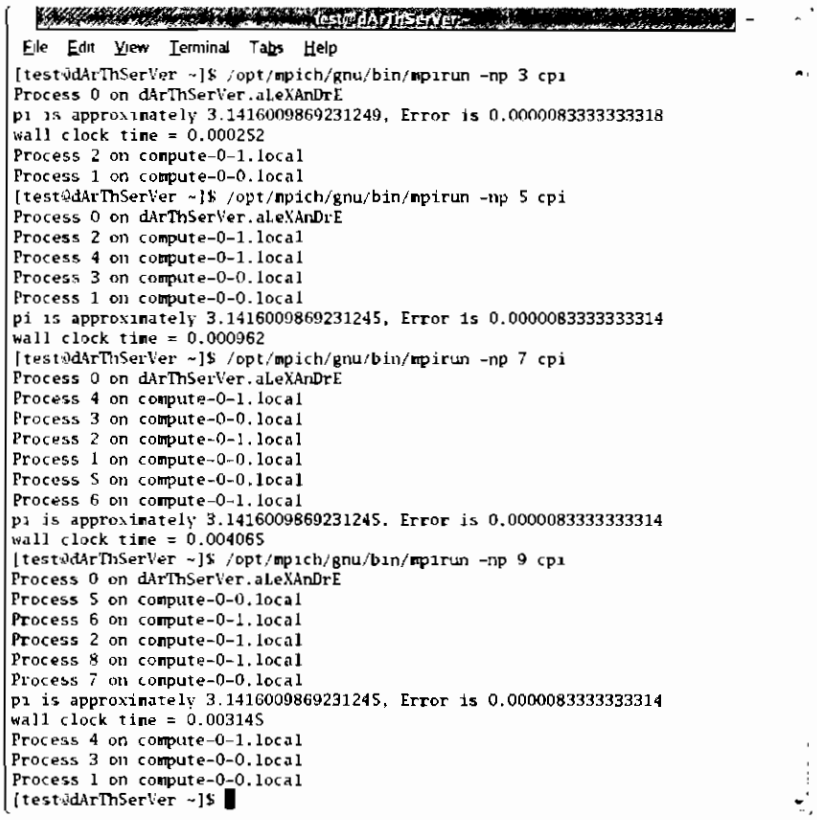


Figura 5-14. Resultados obtenidos al calcular PI utilizando *Rocks*

Además, se ejecutó el *benchmark* de Alto Rendimiento Linpack (HPL - *High Performance Linpack*) que incluye *Rocks*. HPL es un software que permite resolver sistemas lineales aleatorios en aritmética de doble precisión. Para ejecutar HPL, haciendo uso de toda la capacidad de cada computador, se utilizó el *kernel* con soporte para HyperThreading. La ejecución de HPL se realiza en cuatro procesadores (en realidad dos procesadores con tecnología HyperThreading) como sigue:

- Se crea un archivo en el directorio `/home` de un usuario (que no sea *root*) llamado `machines`, en dicho archivo se colocan cuatro entradas que representan a los nodos clientes: `compute-0-0`, `compute-0-1`, `compute-0-0` y `compute-0-1`[12]. El nodo *frontend* no realiza procesamiento, sólo los nodos de cómputo realizan procesamiento en esta prueba, por esta razón, el nombre del nodo *frontend* no se incluye en el archivo `machines`.

- Se debe descargar el archivo de configuración de HPL del sitio Web de *Rocks* y almacenarlo con el nombre `HPL.dat` en el directorio `/home` del usuario.

- Se debe modificar el archivo `HPL.dat` y cambiar las entradas `1 Ps` y `2 Qs` por `2 Ps` y `2 Qs` (`Ps` representa el número de procesadores de cada equipo y `Qs` representa el número de equipos). El número de procesadores que HPL utiliza es calculado mediante la multiplicación de `Ps` veces `Qs`.

- Se deben ejecutar los siguientes comandos en el *frontend*:

```
$ ssh-agent $SHELL
$ ssh-add
$ /opt/mpich/gnu/bin/mpirun -nolocal -np 4 -machinefile machines
/opt/hpl/gnu/bin/xhpl
```

  El comando `ssh-agent` almacena las llaves privadas usadas para autenticar una llave pública, este comando inicia el intérprete definido por la variable

[12] Para disponer de dos procesadores por cada nodo, se debe agregar una entrada por cada procesador.

de entorno `$SHELL` y permite que otros programas se autentiquen cuando realizan conexiones a otros equipos usando SSH. El comando `ssh-add` agrega las llaves privadas usadas por SSH al agente `ssh-agent`. El comando `/opt/mpich/gnu/bin/mpirun` ejecuta una tarea de MPI; la opción `-nolocal` especifica que la ejecución de la aplicación `/opt/hpl/gnu/bin/xhpl` no se realizará en el nodo *frontend*; la opción `-np 4` especifica que se utilizarán 4 procesos para la ejecución y la opción `-machinefile machines` permite especificar el nombre del archivo que contiene los nombres de los nodos sobre los cuales se ejecutará la aplicación.

- HPL se utiliza para generar la lista "TOP 500" de supercomputadores basándose en el rendimiento obtenido.

- Para incrementar la cantidad de trabajo que cada nodo realiza se puede incrementar el parámetro `Ns` del archivo `HPL.dat`. Por defecto el valor de `Ns` es de 1000. El parámetro `Ns` especifica el tamaño del problema que se va a resolver, es decir, la cantidad de elementos del sistema.

Mayor información sobre la configuración y la ejecución de HPL se puede obtener en el sitio Web:

http://www.netlib.org/benchmark/hpl/tuning.html

Modificando la cantidad de trabajo, con un valor `Ns` igual a 6500, el rendimiento obtenido al resolver el HPL fue de 3.17 GFlops y el tiempo que tomó la ejecución de la prueba fue de 57.79 segundos.

En la Figura 5-15 se muestra la pantalla principal de Ganglia, en la cual se puede apreciar el uso de la red del *cluster*, de la carga del *cluster*, de la memoria usada y del procesamiento realizado.

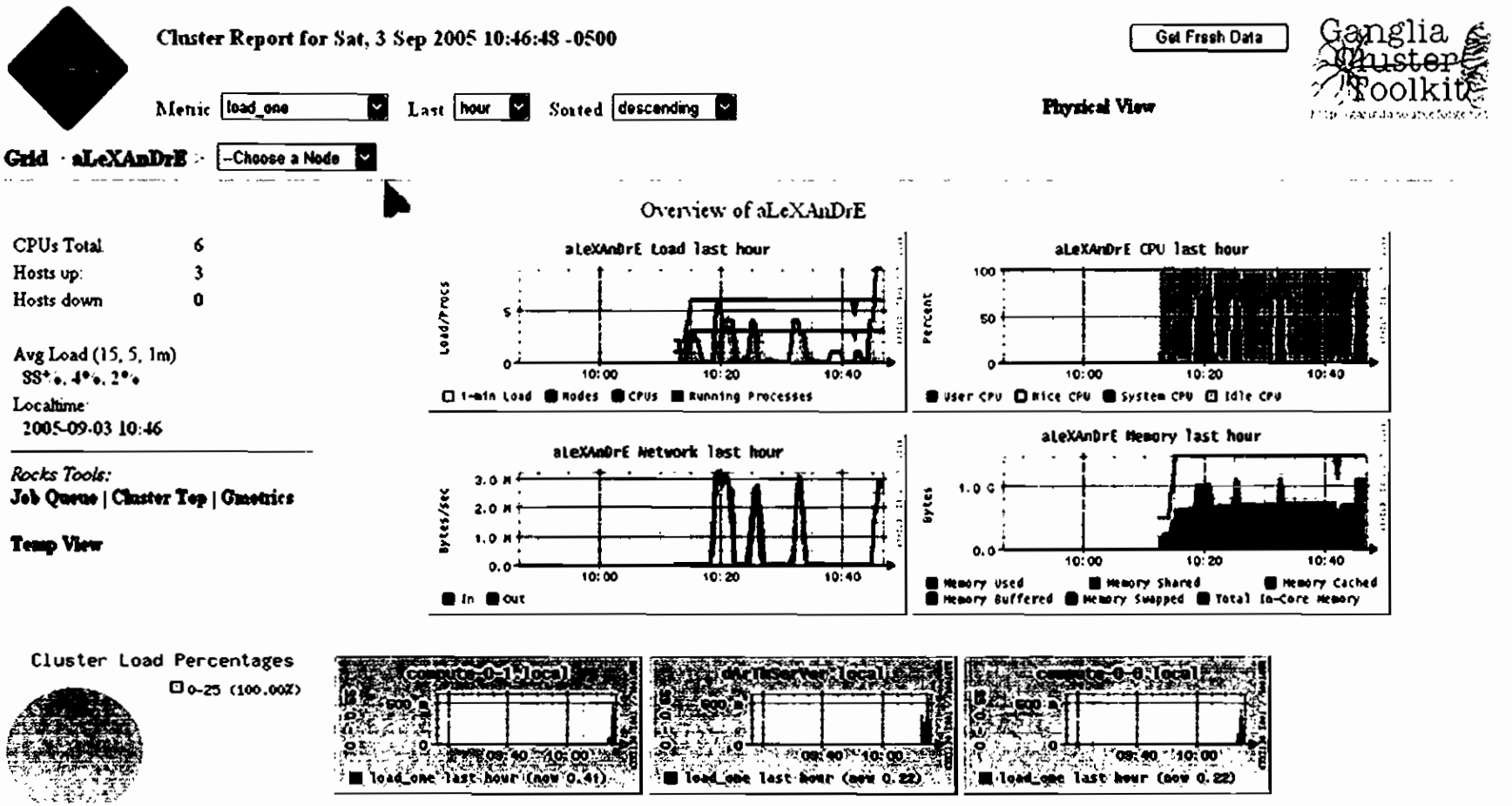


Figura 5-15. Información de Ganglia obtenida en *Rocks*

En la Figura 5-16 se muestra una explicación de la información obtenida mediante Ganglia.

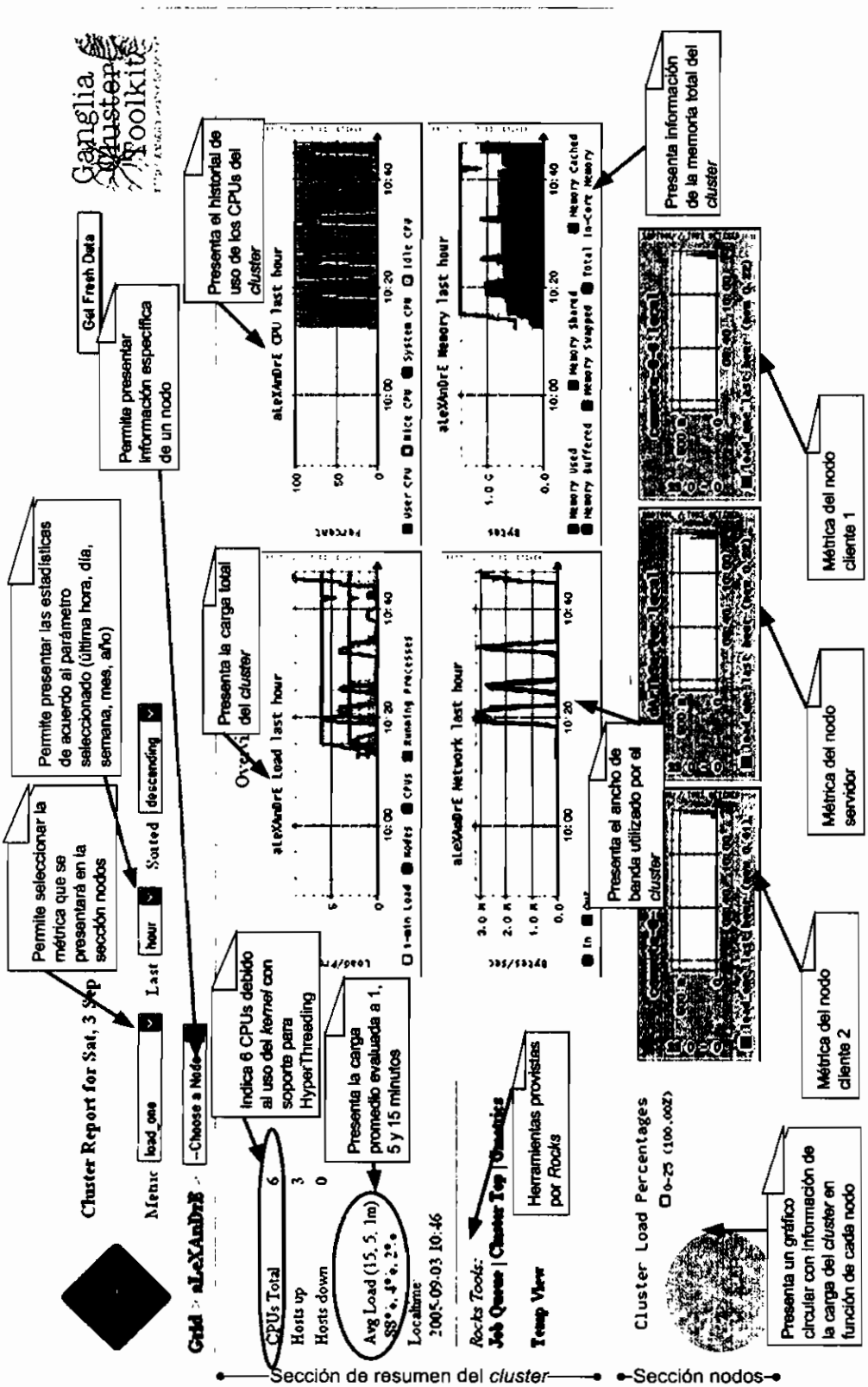


Figura 5-16. Breve explicación de la pantalla de Ganglia obtenida en *Rocks*

En la Figura 5-17 se muestran los resultados que Ganglia presenta para informar la carga de cada nodo. La carga de cada nodo se representa mediante un color específico; además, si un nodo está caído, se presenta la cantidad de segundos que han transcurrido a partir del último informe que se tuvo de dicho nodo.

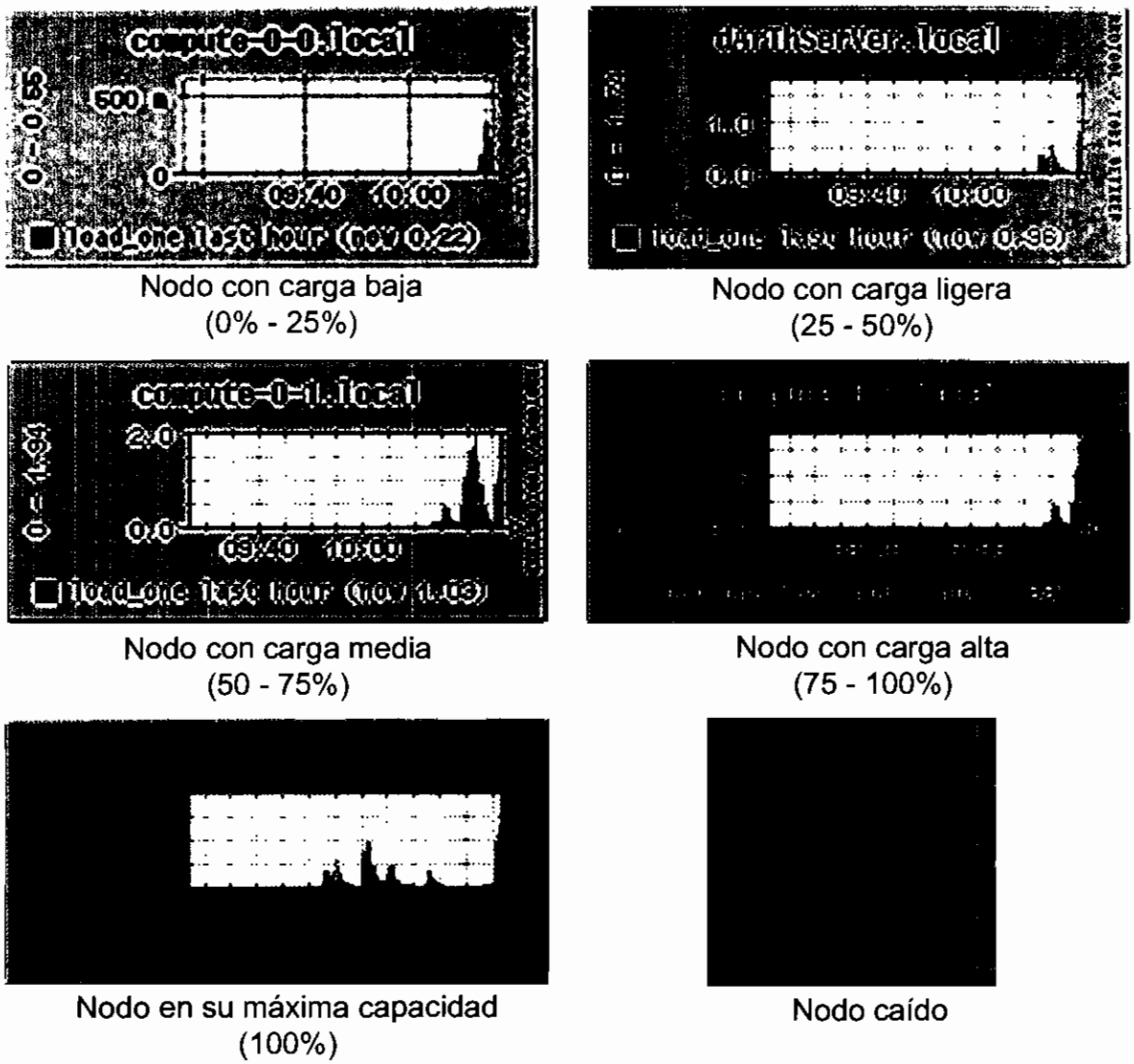


Figura 5-17. Diferentes estados de los nodos

La versión de Ganglia incluida en *Rocks* presenta adicionalmente gráficos circulares (*pie*) en los cuales se resume el estado del *cluster* basándose en la información de los nodos. En la Figura 5-18 se muestran algunos gráficos circulares obtenidos con Ganglia. En cada gráfico circular se puede apreciar la carga de cada nodo; en la Figura 5-14 (a) se observa que los tres nodos (100%) tienen carga baja (0-25); en la Figura 5-14 (b) se observa que un nodo (33,33%)

tiene carga baja (0-25) y los otros dos nodos (66,67%) tienen carga alta (75-100); en la Figura 5-14 (c) se observa que un nodo (33,33%) tiene carga ligera (25-50) y los otros dos nodos (66,67%) tienen carga alta (75-100); en la Figura 5-14 (d) se observa que un nodo (33,33%) tiene carga baja (0-25), otro nodo (33,33%) tiene carga media (50-75) y el nodo restante (33,33%) tienen carga alta (75-100); en la Figura 5-14 (e) se observa que un nodo (33,33%) está en su máxima capacidad (100+) y los nodos restantes (66,67%) tienen carga alta (75-100); y en la Figura 5-14 (f) se observa que un nodo (33,33%) tiene carga baja (0-25), el otro nodo (33,33%) tiene carga ligera (25-50) y el nodo restante (33,33%) está apagado (*down*).

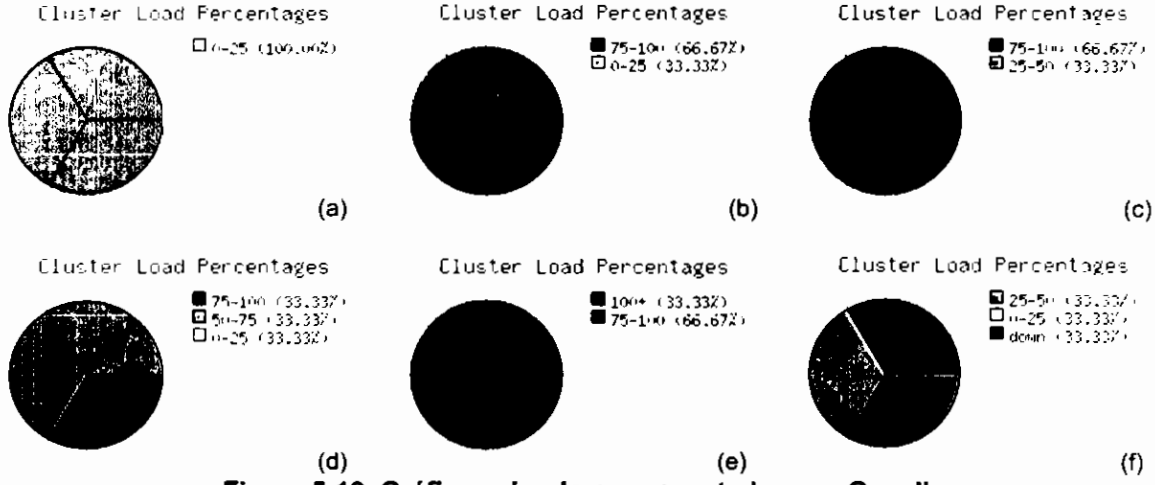


Figura 5-18. Gráficos circulares presentados por Ganglia

Por otro lado, se muestran los resultados de la ejecución del comando **cluster-fork** para listar los archivos de la carpeta `/root` de los nodos cliente en la Figura 5-19 y para reiniciar los nodos cliente en la Figura 5-20.

```
File  Edit  View  Terminal  Tabs  Help
[root@ArThServer ~]# cluster-fork ls
compute-0-0:
anaconda-ks.cfg
install.log
install.log.syslog
RCS
compute-0-1:
anaconda-ks.cfg
install.log
install.log.syslog
RCS

[root@ArThServer ~]#
```

Figura 5-19. Utilización del comando `cluster-fork` para listar el contenido del directorio `/root` de los nodos

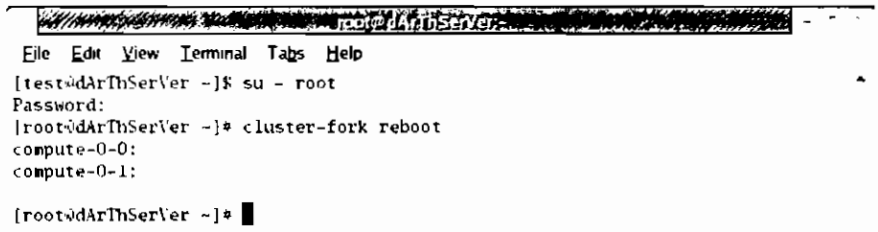


Figura 5-20. Utilización del comando `cluster-fork` para reiniciar los nodos

5.3. DISEÑO E IMPLEMENTACIÓN DE UNA APLICACIÓN CON MPI PARA LA RESOLUCIÓN DE PROCESOS DE MARKOV A TIEMPO DISCRETO Y CONTINUO

El programa desarrollado permite probar la funcionalidad del sistema *cluster* **aLeXaNDrE** que fue instalado, configurado y administrado bajo un ambiente Linux; sin embargo, fue desarrollado en su totalidad en un sistema operativo Windows usando la herramienta de desarrollo Visual Studio .NET 2003 integrada con MPICH Versión 1 y 2. En el Anexo E se muestran los criterios usados para integrar MPICH con Visual Studio. Se utilizó Visual Studio .NET 2003 para desarrollar la aplicación debido a los conocimientos obtenidos en los años de pregrado, además de la facilidad que brinda para depurar aplicaciones y de la cantidad de información que dispone en su colección de ayuda.

La aplicación desarrollada permite resolver Cadenas de Markov. Se escogió como aplicación la resolución de Cadenas de Markov, debido a que se puede demostrar las ventajas del *cluster* al paralelizar las operaciones involucradas en su solución. Las Cadenas de Markov pueden hacer uso de matrices de grandes dimensiones, por lo que su solución y las operaciones asociadas pueden requerir una gran capacidad de procesamiento computacional.

Las operaciones más relevantes utilizadas en la solución de Cadenas de Markov son:

- Multiplicación de matrices.
- Resolución de sistemas de ecuaciones lineales.

5.3.1. PRESENTACIÓN DE LA APLICACIÓN DESARROLLADA

En términos generales la aplicación se diseñó en base al modelo Maestro–Esclavo. De acuerdo a los conceptos presentados anteriormente, este modelo permite que un proceso denominado Maestro, el cual se ejecuta en el nodo principal, coordine las tareas que realizan un grupo de procesos denominados Esclavos, los cuales se encuentran en los nodos de cómputo. En la Figura 5-21 se describe la arquitectura básica del sistema implementado.

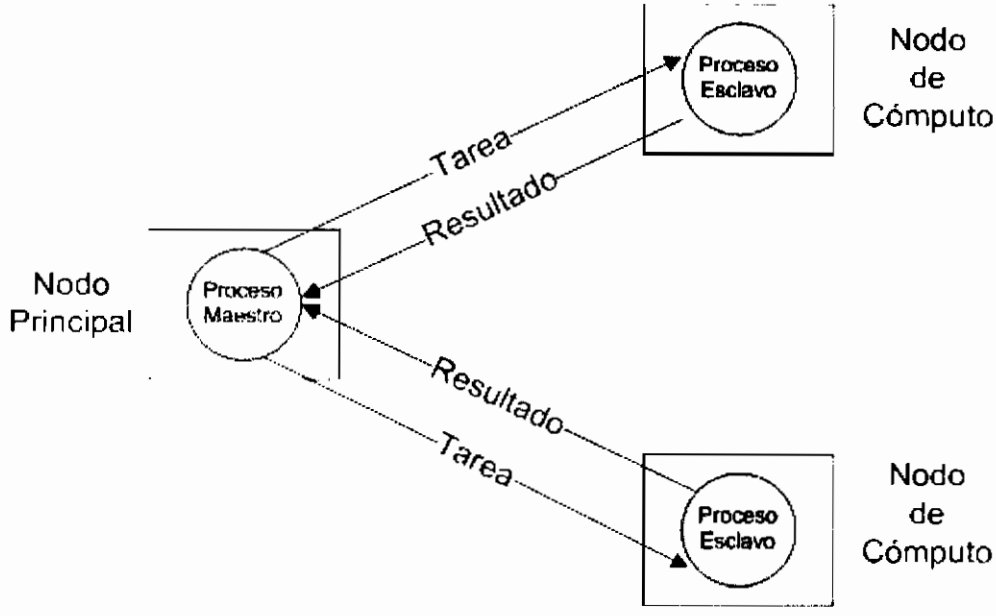


Figura 5-21. Arquitectura de la aplicación implementada

A continuación se describe el análisis matemático de las operaciones entre matrices, y la solución de sistemas de ecuaciones y Cadenas de Markov; además, se presentan las ideas básicas utilizadas en el desarrollo de los algoritmos paralelos para implementar los métodos matemáticos necesarios.

5.3.2. OPERACIONES ENTRE MATRICES

5.3.2.1. Suma de matrices

La suma de una matriz **A** de `l` filas y `m` columnas, y una matriz **B** de `l` filas y `m` columnas es una matriz **C**, cuyos elementos están definidos por:

$$c_{i,j} = a_{i,j} + b_{i,j}$$

(5-1)

Este algoritmo se basa en un lazo, durante cada iteración del lazo basada en un índice `i`, se lee el elemento `(i,j)` de la matriz **A** y el elemento `(i,j)` de la matriz **B**, y se escriben los datos en el elemento `(i,j)` de la matriz **C**.

5.3.2.2. Multiplicación de matrices

El producto de una matriz **A** de `l` filas y `m` columnas, por una matriz **B** de `m` filas y `n` columnas es una matriz **C**, cuyos elementos están definidos por:

$$c_{i,j} = \sum_{k=0}^{m-1} a_{i,k} \cdot b_{k,j}$$

(5-2)

Mediante operaciones sucesivas basadas en las filas de una matriz se obtiene un algoritmo que permite efectuar la multiplicación de matrices. Este algoritmo se basa en un lazo, durante cada iteración del lazo basada en un índice `i`, se lee la fila `i` de la matriz **A** y los elementos de la columna `j` de la matriz **B**, y se escriben los datos en la fila `i` de la matriz **C**.

5.3.2.3. Potencia de matrices

A partir de la multiplicación de matrices, se define la potencia *n*-ésima de una matriz **A** como:

$$\mathbf{C} = \mathbf{A}^n$$
$$\mathbf{C} = (\mathbf{A} \cdot \mathbf{A}) \cdot \mathbf{A}^{n-2}$$

(5-3)

5.3.2.3.1. Paralelización de la potencia de matrices

En la operación potencia descrita anteriormente, cada elemento de la matriz **C** es una función de los elementos de la matriz **A**. Debido a que los valores de **A** no serán modificados en el algoritmo, se puede calcular de manera simultánea cada elemento de **C**. Para paralelizar el proceso, el proceso maestro crea una copia de la matriz **A** denominada **B**.

El proceso maestro, calcula el número de filas que cada proceso va a utilizar basado en:

Porción a procesar = Filas de matriz **A** / Número de procesos

Luego de conocer la cantidad de filas que un proceso va a dar tratamiento, el proceso maestro, envía las filas de la matriz **A** correspondientes y toda la matriz **B** a cada uno de los procesos.

Cada proceso es responsable de realizar la potencia de la porción asignada de filas de la matriz **A** usando la matriz **B** como la base de la operación potencia, generando los elementos de la matriz **C** correspondientes.

Después de realizar el procesamiento necesario para obtener la porción de la matriz $\mathbf{A}^n$ los procesos envían esta porción de regreso al proceso maestro.

5.3.3. SISTEMAS DE ECUACIONES LINEALES

A continuación se describe el método utilizado para resolver sistema de ecuaciones lineales y luego se presenta una breve descripción del algoritmo implementado para paralelizar dicho método.

5.3.3.1. Sistemas Lineales: Eliminación de Gauss-Jordan

Un conjunto de `m` ecuaciones con `m` incógnitas, puede representarse con una ecuación de la forma:

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{y}$$

(5-4)

donde **A** se denomina matriz de conexión y contiene los coeficientes del sistema de ecuaciones, **x** se denomina vector incógnita y está conformado por las incógnitas de dicho sistema y **y** es el vector conocido, conformado por los términos independientes del sistema de ecuaciones.

El método de Gauss Jordan consiste en la eliminación sucesiva de incógnitas. Se obtiene una matriz identidad mediante la sustitución de los elementos bajo y sobre la diagonal principal de la matriz de conexión por valores iguales a cero. Esto se logra mediante operaciones de suma y resta entre los elementos de las filas de la matriz y la utilización de una fila como pívot para realizar las operaciones correspondientes. Al obtener la matriz identidad, **y** o el vector conocido contendrá la solución del sistema de ecuaciones.

5.3.3.1.1. Paralelización del método de eliminación de Gauss Jordan

El proceso maestro, calcula el número de filas que cada proceso va a utilizar basado en:

Porción a procesar = Filas de matriz **A** / Número de procesos

Luego de conocer la cantidad de filas que un proceso va a dar tratamiento, el proceso maestro, envía la porción de filas de la matriz **A** y la porción de filas de la matriz **y** correspondientes a cada uno de los procesos. Cada proceso es responsable de normalizar la fila pívot que le corresponda y de reducir los elementos de las otras filas; para esto, de forma ordenada, cada proceso envía la fila pívot a los otros procesos, es responsable de normalizarla y de hacer ceros los elementos de las otras filas que le fueron asignadas, los otros procesos realizarán las operaciones necesarias para que en base al pívot recibido, se encarguen de reducir sus filas. De esta forma, todos los procesos colaboran en la reducción de los elementos de la matriz **A** y obtienen los valores de la porción de la matriz **y** que les fue asignada. Una vez reducida la matriz **A**, cada proceso

envía sus resultados, su porción de la matriz **y**, al proceso maestro para que arme la matriz **y**.

Por ejemplo, si se tiene una matriz **A** de 6 filas y 6 columnas y una matriz **y** de 6 filas y 1 columna, y existen 3 procesos, denominados P0, P1, P2, cada proceso será responsable de dar tratamiento a 2 filas de cada matriz. Un diagrama del proceso que se realiza para paralelizar la solución se muestra en la Figura 5-22.

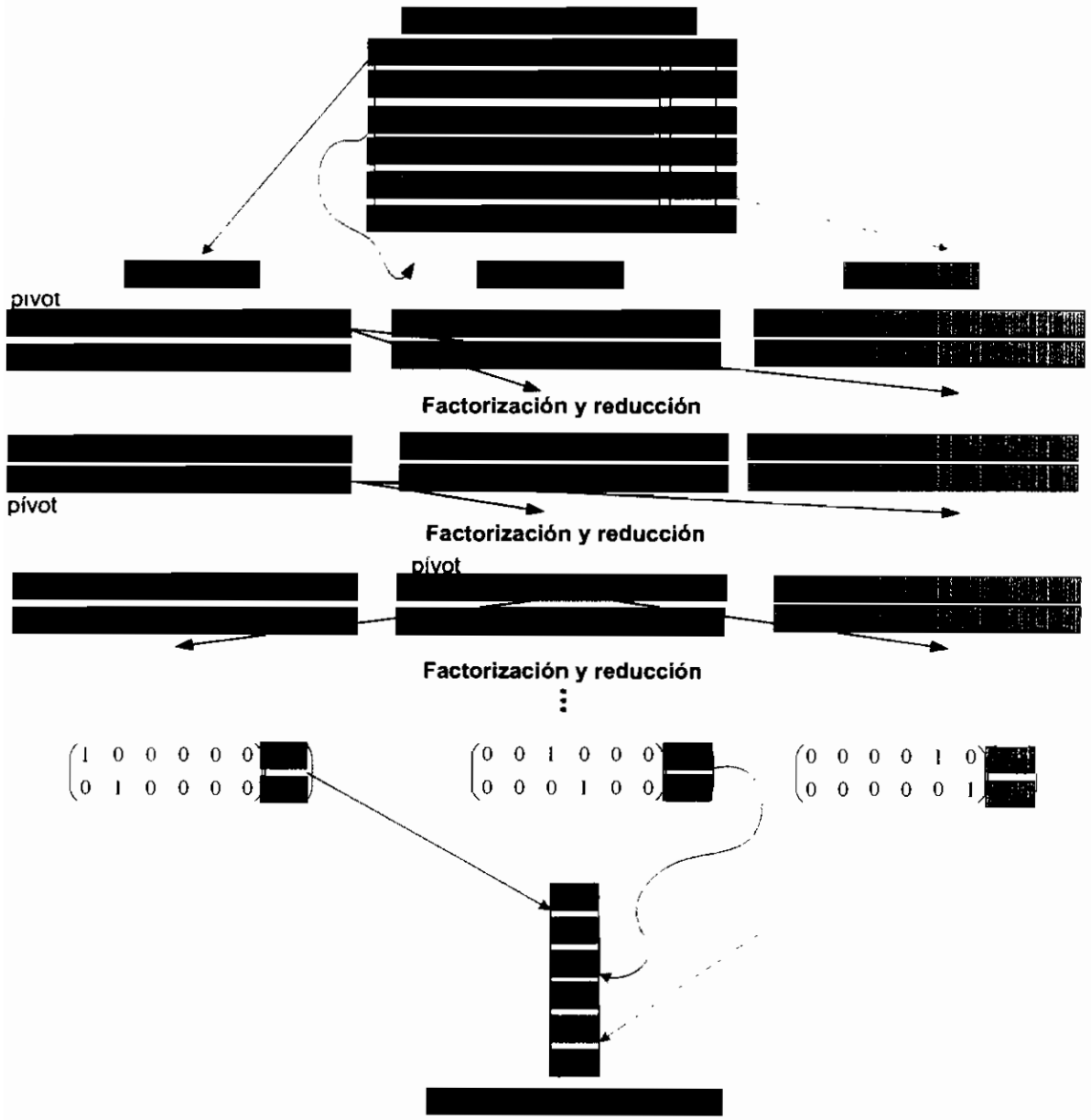


Figura 5-22. Eliminación de Gauss-Jordan paralelizada

El proceso P0 será responsable de trabajar con la fila 1 y 2 de las matrices **A** y **y**; el proceso P1 será responsable de trabajar con la fila 3 y 4 de las matrices **A** y **y**; y el proceso P2 será responsable de trabajar con la fila 5 y 6 de las matrices **A** y **y**. Primero, el proceso P0, usa la fila 1 como pívot, la envía a los otros procesos, la normaliza y reduce la fila 2 en base al pívot. Los procesos P1 y P2 obtienen el pívot y reducen sus filas en base a este pívot. El proceso P0, procede a usar la fila 2 como pívot, la envía a los otros procesos, la normaliza y reduce la fila 1 en base al pívot; los otros procesos obtienen el pívot y reducen sus filas en base al pívot. Luego, el proceso P1 usa la fila 3 como pívot, la envía a los otros procesos, la normaliza y reduce la fila 4 en base a su pívot; los otros procesos obtienen el pívot y proceden a reducir sus filas. Y se continúa hasta que todos los procesos han reducido sus filas dando como resultado de este proceso una matriz **y** que contiene los resultados parciales de la solución del sistema de ecuaciones.

5.3.4. CADENAS DE MARKOV

En general, se puede definir a un proceso estocástico como un modelo matemático probabilístico, utilizado con frecuencia para describir fenómenos aleatorios, representables como funciones de un parámetro que usualmente es el tiempo.

Por otra parte los Procesos de Markov son procesos estocásticos que permiten definir modelos probabilísticos para describir el comportamiento de redes de telecomunicaciones.

Si los Procesos de Markov se consideran en espacio discreto, se denotan Cadenas de Markov.

Las Cadenas de Markov pueden ser a tiempo continuo o a tiempo discreto, dependiendo de los valores que el parámetro tiempo puede asumir.

Sin embargo, se restringe la solución de las Cadenas de Markov a aquellas que cumplen con la propiedad de homogeneidad y de ser irreducibles. Una Cadena de

Markov se considera homogénea si la evolución futura de un proceso está completamente especificada, en términos probabilísticos, a partir del estado actual. Una Cadena de Markov se considera irreducible si no existe un estado en el cual el proceso se quede de forma indefinida, un estado trampa[13].

Se recomienda al lector revisar el libro "*Markov Chains*" de James Norris, publicado por la Universidad de Cambridge, Inglaterra; el libro "*Markov Chains – a Quick Introduction*" de Andrew Lyassof, publicado por la Universidad de Boston, Estados Unidos y las páginas Web:

- http://www.cms.wisc.edu/~cvg/course/491/modules/Markov/Markov/node2.html,
- http://mathforum.org/workshops/sum96/discrete/markov.html,
- http://people.hofstra.edu/faculty/Stefan_Waner/RealWorld/Summary8.html

para profundizar la información aquí presentada acerca de las Cadenas de Markov, así como del análisis matemático necesario para la obtención de las ecuaciones descritas en las siguientes secciones.

5.3.4.1. Cadenas de Markov: Tiempo Discreto

En el caso de las Cadenas de Markov de Tiempo Discreto (CMTD), uno de los objetivos es evaluar la probabilidad de que el proceso se encuentre en un estado específico en un paso dado, es decir obtener la **distribución en el paso *n***; el otro objetivo es evaluar la **distribución de régimen**.

La **distribución en el paso *n*** describe la evolución del sistema al paso *n*. El comportamiento a largo término de la Cadena de Markov puede exhibir características muy importantes del sistema. La **distribución de régimen** permite describir el comportamiento a largo término de la Cadena y determinar el estado de equilibrio del sistema.

[13] Un estado trampa (*absorbing*) es un estado en el cual un proceso se queda de forma indefinida, es decir, que la probabilidad de abandonar este estado es 0. La probabilidad de *llegar* a este estado es 1.

Se puede utilizar notación matricial para simplificar la representación de las CMTDs. Se define la matriz de probabilidades de transición **P**:

$$\mathbf{P} = [p_{ij}]$$

(5-5)

donde p_{ij} es la probabilidad de estar en el estado *j* en el siguiente paso, dado que el estado actual es *i*. Además, se cumple que:

$$\sum_j p_{ij} = 1$$

(5-6)

En la Figura 5-23 se presenta la matriz de probabilidades de transición para una Cadena de Markov de 3 estados y en la Figura 5-24 se presenta un ejemplo del diagrama de transición para una Cadena de Markov de 3 estados.

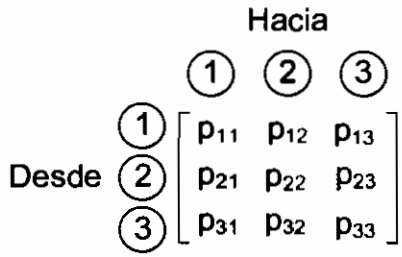


Figura 5-23. Matriz de Probabilidades de Transición para una Cadena de Markov de 3 estados

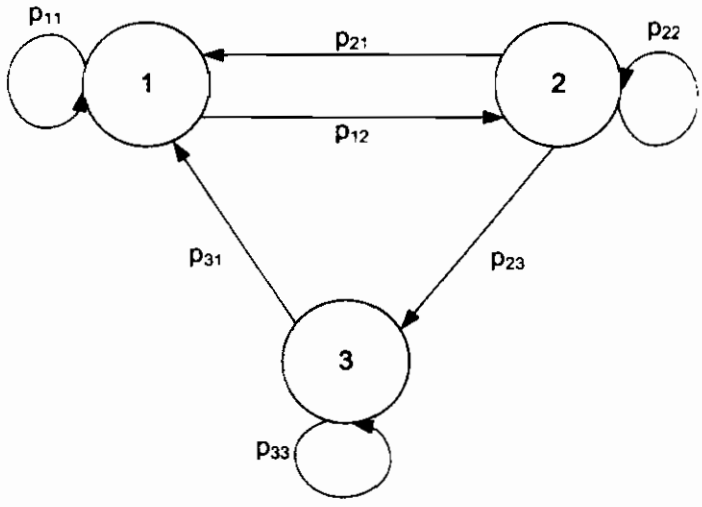


Figura 5-24. Diagrama de Transición para una Cadena de Markov de 3 estados

Para obtener la **distribución en el paso *n***, se define el vector $\mathbf{p}(n)$ como la distribución de estado, cuyos elementos son las probabilidades de estado al paso *n*:

$$\mathbf{p}(n) = [\pi_1(n) \quad \pi_2(n) \quad ...]$$

(5-7)

Además, la matriz de probabilidad de transición en el paso *n* puede expresarse como la potencia *n*-ésima de la matriz **P** y la distribución de estado puede expresarse como:

$$\mathbf{p}(n) = \mathbf{p}(0) \cdot \mathbf{P}^n$$

(5-8)

La **distribución de régimen** puede obtenerse a partir de la ecuación:

$$\mathbf{p} = \mathbf{p} \cdot \mathbf{P}$$

(5-9)

donde $\mathbf{p} = [\pi_1 \quad \pi_2 \quad ...]$

El sistema de ecuaciones obtenido tiene infinitas soluciones. Para obtener un sistema de ecuaciones con solución única, se requiere utilizar la condición de normalización:

$$\sum_j \pi_j = 1$$

(5-10)

5.3.4.2. Cadenas de Markov: Tiempo Continuo

En el caso de las Cadenas de Markov de Tiempo Continuo (CMTC), uno de los objetivos es evaluar que un proceso se encuentre en un estado específico, a un tiempo dado, es decir, obtener la **distribución en el tiempo *t***; el otro objetivo es evaluar la **distribución de régimen**.

La **distribución en el tiempo *t*** describe la evolución del sistema al tiempo *t*. El comportamiento a largo término de la Cadena de Markov puede exhibir

características muy importantes del sistema. La **distribución de régimen** permite describir el comportamiento a largo término de la Cadena de Markov y determinar el estado de equilibrio del sistema.

Se puede utilizar notación matricial para simplificar la representación de las CMTCs. Se define la matriz de probabilidades de transición **P(*t*)**:

$$\mathbf{P}(t) = \left[p_{ij}(t)\right]$$

(5-11)

Además,

$$\mathbf{P}(0) = \mathbf{I}$$

(5-12)

Para obtener la **distribución en el tiempo *t***, se define el vector **p**(*t*) como la distribución de estado, cuyos elementos son las probabilidades de estado al tiempo *t*:

$$\mathbf{p}(t) = \left[\pi_1(t) \quad \pi_2(t) \quad ...\right]$$

(5-13)

Además, se puede definir el generador infinitesimal de la matriz de probabilidades de transición **P(*t*)** o matriz de tasas de transición como:

$$\mathbf{Q} = \left[q_{ij}\right]$$

(5-14)

donde q_{ij} es la tasa con la cual el proceso se mueve del estado *i* al estado *j* y cumple con la condición: $q_{ij} \geq 0 \quad si \quad i \neq j$; y además, se cumple que:

$$\sum_j q_{ij} = 0$$

(5-15)

La matriz de probabilidades de transición puede obtenerse utilizando:

$$\mathbf{P}(t) = e^{\mathbf{Q} \cdot t}$$

(5-16)

que a su vez puede resolverse mediante el uso de la Serie de Taylor[14] para las funciones exponenciales:

$$\mathbf{P}(t) = e^{\mathbf{Q}\cdot t} = \mathbf{I} + \sum_{k=1}^{\infty} \frac{t^{k}}{k!} \cdot \mathbf{Q}^{k}$$

(5-17)

Y la distribución de estado puede expresarse como:

$$\mathbf{p}(t) = \mathbf{p}(0) \cdot \mathbf{P}(t)$$

(5-18)

La **distribución de régimen** puede obtenerse a partir de la ecuación:

$$\mathbf{p} \cdot \mathbf{Q} = 0$$

(5-19)

donde $\mathbf{p} = [\pi_1 \quad \pi_2 \quad ...]$

El sistema de ecuaciones obtenido tiene infinitas soluciones. Para obtener un sistema de ecuaciones con solución única, se requiere utilizar la condición de normalización:

$$\sum_{j} \pi_j = 1$$

(5-20)

5.3.4.2.1. Solución de Cadenas de Markov

Como se puede ver del análisis matemático descrito anteriormente, el resolver los parámetros de las Cadenas de Markov involucra el utilizar los algoritmos descritos para resolver los sistemas de ecuaciones y las operaciones entre matrices.

En el caso de CMTD, para obtener la **distribución en el paso *n***, se eleva a la potencia *n*-ésima la matriz de probabilidades de transición, lo cual involucra realizar *n* multiplicaciones sucesivas de dicha matriz, para luego multiplicar la matriz obtenida por el vector de estado de la distribución inicial; para obtener la

[14] BROOK TAYLOR, matemático inglés que introdujo las series que llevan su nombre.

distribución de régimen se obtiene el sistema de ecuaciones lineales mediante la multiplicación de la matriz de probabilidades de transición por el vector de estado y haciendo uso de la condición de normalización para obtener un sistema de ecuaciones cuya solución sea única.

En el caso de CMTC, para obtener la **distribución al tiempo *t***, se utiliza las Series de Taylor para la función exponencial, para obtener la matriz de probabilidades de transición, para luego multiplicar la matriz obtenida por el vector de estado de la distribución inicial; para obtener la **distribución de régimen** se obtiene el sistema de ecuaciones lineales mediante la multiplicación de la matriz de probabilidades de transición por el vector de estado y haciendo uso de la condición de normalización para obtener un sistema de ecuaciones cuya solución sea única.

5.3.5. ESTRUCTURA Y FUNCIONALIDAD DE LA APLICACIÓN

El código fuente y los archivos asociados a la aplicación desarrollada se presentan en un CD, que se adjunta al presente Proyecto de Titulación. A continuación se ofrece una breve descripción de la aplicación desarrollada y se presenta con mayor detalle de las funciones más relevantes.

Se diseñaron clases utilizando el lenguaje C++ para poder facilitar la solución. La clase llamada `CMatrix` define todas las operaciones secuenciales entre matrices. La clase `CMatrix_Handler` maneja las operaciones matriciales de forma paralela, y manipula las comunicaciones mediante el paso de mensajes. La clase `CMarkovTD` permite obtener los valores de parámetros involucrados en las Cadenas de Markov de Tiempo Discreto, y la clase `CMarkovTC` para determinar los valores de los parámetros involucrados en las Cadenas de Markov de Tiempo Continuo.

En la Figura 5-25 se muestra el diagrama UML de clases y la interrelación entre las clases creadas y sus principales funciones.

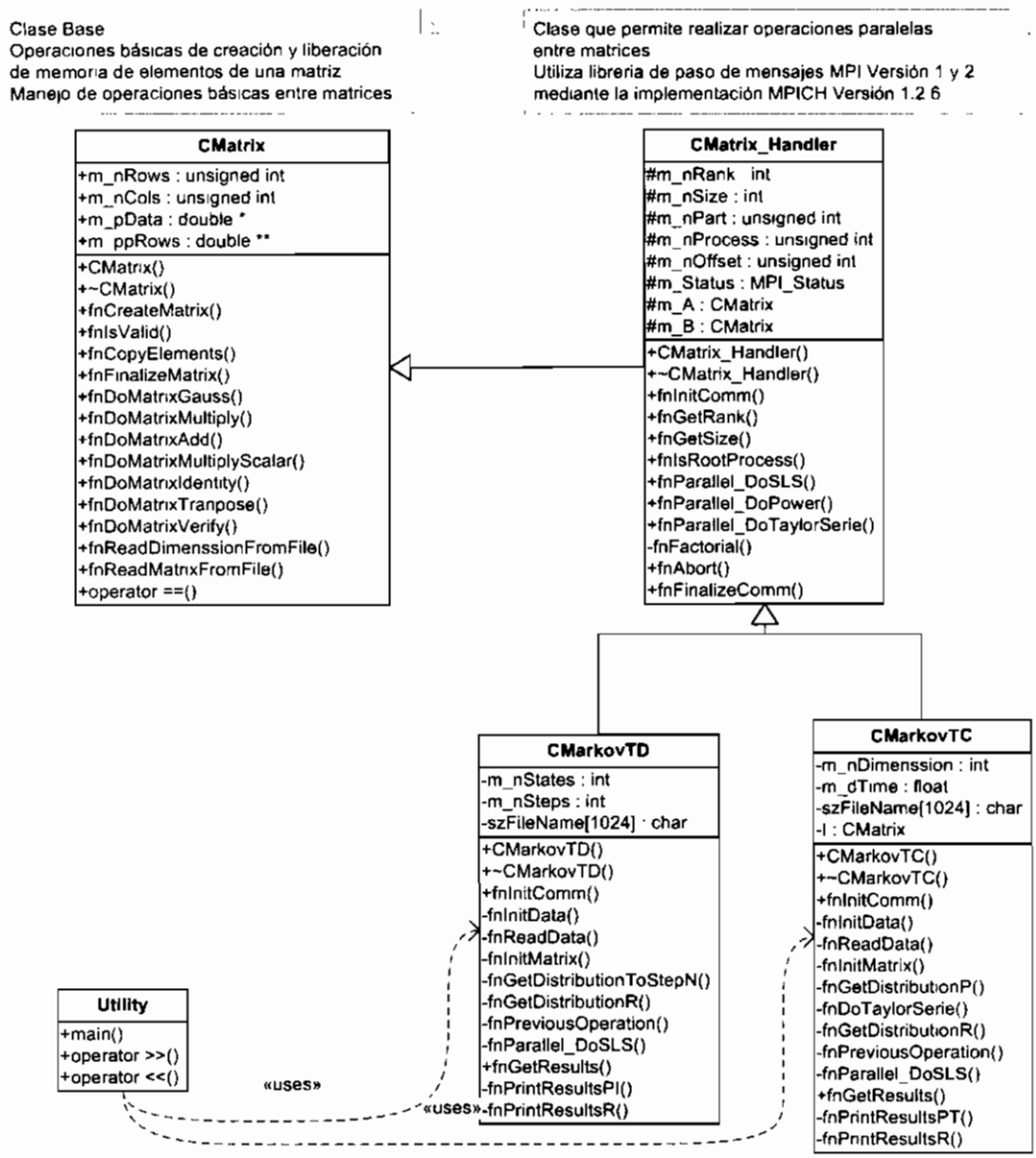


Figura 5-25. Diagrama UML de clases

5.3.5.1. Funcionalidad de la aplicación

En síntesis, la aplicación realiza las siguientes funciones:

1. El proceso maestro lee los datos de configuración y de las matrices.
2. El proceso maestro inicializa la librería de MPI.
3. El proceso maestro envía el paso al que se va a evaluar la CMTD o el tiempo al que se va a evaluar la CMTC y la dimensión de la matriz **P** o **Q**(t) a los otros procesos.

4. El proceso maestro comprueba que la matriz cumpla con las condiciones especificadas en la Sección 5.3.4.1.1 y 5.3.4.1.2.
5. Se obtiene la distribución al paso *n* o al tiempo *t*. Para lo cual el proceso maestro calcula la porción de trabajo que le corresponde realizar a cada proceso y envía la porción correspondiente de la matriz (**P** en CMTD o **Q**(*t*) en CMTC) para que los otros procesos ayuden en las operaciones requeridas. Luego de realizar las operaciones, los procesos envían sus resultados al proceso maestro para que los presente en consola y los almacene en un archivo.
6. Se obtiene la distribución de régimen. Para lo cual el proceso maestro forma la matriz de conexión y el vector conocido, envía las porciones correspondientes a cada proceso para su resolución y luego recupera los resultados para presentarlos en consola y almacenarlos en un archivo.
7. Cada proceso libera la memoria utilizada y el proceso maestro se encarga además de liberar las librerías de MPI.

5.3.6. CLASE `CMatrix`

`CMatrix` es una clase de C++. En la clase **`CMatrix`** se implementa la función **`fnDoMatrixMultiplyScalar(double dValue, CMatrix& A)`**, donde el parámetro **dValue** representa el escalar a ser multiplicado por la matriz **A**. En la Figura 5-26 se muestra la implementación de esta función.

```
for (unsigned int i = 0; i < A.m_nRows; i++) // Filas
        for (unsigned int j = 0; j < A.m_nCols; j++) // Columnas
                A.m_ppRows[i][j] = dValue * A.m_ppRows[i][j];
```

Figura 5-26. Código de la función `fnDoMatrixMultiplyScalar`

En esta clase se implementan las funciones **`fnDoMatrixMultiply(CMatrix& A, CMatrix& B, CMatrix& Resultado)`** y **`fnDoMatrixAdd(CMatrix& A, CMatrix& B, CMatrix& Resultado)`**, donde los parámetros **A** y **B** representan las matrices sobre las cuales se va a ejecutar la operación y **Resultado** almacena el resultado de dicha operación. En la Figura 5-27 se muestra la implementación de

la función **fnDoMatrixMultiply** y en la Figura 5-28 se muestra la implementación de la función **fnDoMatrixAdd**.

```
double sum = 0.0;
for (unsigned int i = 0; i < A.m_nRows; i++) // Filas
{
	for (unsigned int j = 0; j < B.m_nCols; j++) // Columnas
	{
		sum = 0;
		for (unsigned int k = 0; k < A.m_nCols; k++)
			sum += A.m_ppRows[i][k] * B.m_ppRows[k][j];
		Resultado.m_ppRows[i][j] = sum;
	}
}
```

Figura 5-27. Código de la función fnDoMatrixMultiply

```
for (unsigned int i = 0; i < A.m_nRows; i++) // Filas
{
	for (unsigned int j = 0; j < B.m_nCols; j++) // Columnas
	{
		Resultado.m_ppRows[i][j] = A.m_ppRows[i][j] + B.m_ppRows[i][j];
	}
}
```

Figura 5-28. Código de la función fnDoMatrixAdd

También, se implementa en esta clase la función **fnDoMatrixPower(CMatrix& A, CMatrix & B, CMatrix& Resultado, int nPotencia)** donde la matriz **B** representa la matriz base de la operación potencia y la matriz **A** es una copia (total o parcial[15]) de la matriz **B**, **Resultado** almacena el resultado (total o parcial) de dicha operación y **nPotencia** representa el exponente al cual se eleva la matriz **A**. En la Figura 5-29 se muestra la implementación de la función **fnDoMatrixPower**.

```
CMatrix tempo;
tempo = fnCreateMatrix(A.m_nRows, A.m_nCols);
fnCopyValues(A, tempo);
for (int x = 0; x < nPotencia - 1; x++)
{
	fnDoMatrixMultiply(tempo, B, Resultado);
	fnCopyValues(Resultado, tempo);
}
```

Figura 5-29. Código de la función fnDoMatrixPower

[15] Al paralelizar la potencia, un proceso sólo realiza la potencia de una parte de la matriz, por lo que la matriz A puede ser sólo una parte de la matriz B.

Se implementa la función **fnDoMatrixIdentity(CMatrix& A)**, donde el parámetro **A** almacena la matriz identidad. Y además, se implementa la función **fnDoMatrixTranspose(CMatrix& A)**, en la cual el parámetro **A** almacena la matriz transpuesta. En la Figura 5-30 se muestra la implementación de la función **fnDoMatrixIdentity** y en la Figura 5-31 se muestra la implementación de la función **fnDoMatrixTranspose**.

```
if(A.m_nRows == A.m_nCols)
{
        for(int i = 0; i < (int) A.m_nRows; i++)
        {
                for(int j = 0; j < (int) A.m_nCols; j++)
                {
                        if(i == j)      A.m_ppRows[i][j] = 1;
                        else            A.m_ppRows[i][j] = 0;
                }
        }
        return true;
}
return false;
```

Figura 5-30. Código de la función fnDoMatrixIdentity

```
CMatrix Temp;
Temp = fnCreateMatrix(A.m_nRows, A.m_nCols);
fnCopyValues(A, Temp);
for(int i = 0; i < (int) A.m_nRows; i++)
{
        for(int j = 0; j < (int) A.m_nCols; j++)
                A.m_ppRows[i][j] = Temp.m_ppRows[j][i];
}
if(fnIsValid(A)) return true;
return false;
```

Figura 5-31. Código de la función fnDoMatrixTranspose

Las matrices se crean a partir de los elementos que se leen de un archivo, debido a que éstas podrían ser de grandes dimensiones, y sería difícil para el usuario ingresar todos los elementos de la matriz desde consola. La función que se implementa con el fin de leer la matriz desde un archivo es: **fnReadMatrixFromFile(char*, CMatrix&)** cuyos parámetros permiten especificar el nombre del archivo y la matriz que contendrá los elementos que se leen desde el archivo.

También, implementa la función **fnDoMatrixGauss(CMatrix& A, CMatrix& B, int nPivotRow, int nPivotCol, double* dRowA, double* dRowB)**, que permite utilizar el algoritmo de Gauss-Jordan para resolver sistemas de ecuaciones lineales, donde los parámetros **A** y **B** representan a la matriz de conexión y al vector conocido, respectivamente; los parámetros **nPivotRow** y **nPivotCol** permiten identificar la fila pívot y el elemento de la diagonal principal; y los parámetros **dRowA** y **dRowB** son arreglos que contienen las filas pívot de la matriz **A** y **B**, respectivamente, que se utilizan para reducir la matriz. En la Figura 5-32 se muestra la implementación de la función **fnDoMatrixGauss**.

```
unsigned int i = 0, j = 0;
double dTemp = 0.0;
// Se obtiene el valor del elemento para proceder a normalizar
// la fila dRow definida por nPivotCol
dTemp = dRowA[nPivotCol];
// En caso de que sea 0 el valor del elemento, se termina la
// llamada a la funcion debido a que no existe division para 0
if (dTemp == 0)
	return;
// Se normaliza los elementos de la fila dRow de A y de B
for (j = 0; j < A.m_nCols; j++)
	dRowA[j] /= dTemp;
for (j = 0; j < B.m_nCols; j++)
	dRowB[j] /= dTemp;
// Se reducen las filas en base al pivot
for (i = 0; i < A.m_nRows; i++)
{
	// si es la fila pivot, se procede con la siguiente fila
	if(i == nPivotRow)
		continue;
	// Se obtiene el valor del elemento de la columna
	// identificado por nPivotCol
	// (Elementos sobre o bajo la diagonal principal)
	dTemp = A.m_ppRows[i][nPivotCol];
	// En caso de ser 0, significa que no se necesita
	// realizar operaciones en esta fila
	if(dTemp == 0)
		continue;
	// Se realiza operaciones sobre la fila actual, basandose
	// en el pivot para hacer ceros a los elementos
	for (j = 0; j < A.m_nCols; j++)
		A.m_ppRows[i][j] -= dTemp * dRowA[j];
	for (j = 0; j < B.m_nCols; j++)
		B.m_ppRows[i][j] -= dTemp * dRowB[j];
}
```

Figura 5-32. Código de la función fnDoMatrixGauss

5.3.7. CLASE `CMatrix_Handler`

`CMatrix_Handler` es una clase de C++ derivada de **`CMatrix`**. En la clase **`CMatrix_Handler`** se implementan las funciones que permiten inicializar, liberar y obtener parámetros de la librería de paso de mensajes MPI: **`fnInitComm(int argc, char** argv)`** llama la función **`MPI_Init`** para iniciar la librería de paso de mensajes; la función **`fnFinalizeComm(void)`** llama a la función **`MPI_Finalize`** para liberar la librería de paso de mensajes; la función **`fnGetSize(void)`**, llama a la función **`MPI_Comm_size`** para obtener el número de procesos involucrados; la función **`fnGetRank(void)`**, llama a la función **`MPI_Comm_rank`** para obtener el identificador de proceso; y la función **`fnIsRootProcess(void)`** permite verificar que cierta operación se realice sobre el proceso maestro.

Además implementa la función **`fnParallel_DoSLS(CMatrix& A, CMatrix& B)`** que permite paralelizar la solución del sistema de ecuaciones lineales, donde los parámetros **A** y **B** representan a la matriz de conexión y al vector conocido, respectivamente. En la Figura 5-33 se muestra la implementación de la función **`fnParallel_DoSLS`**.

```
CMatrix C, D;
int i0 = 0;

if(fnIsRootProcess())
{
        // C y D permiten almacenar la fila pivot que envian
        //otros proceso
        C = fnCreateMatrix(1, A.m_nCols);
        D = fnCreateMatrix(1, B.m_nCols);

        m_nSizeT[0] = A.m_nRows;
        m_nSizeT[1] = A.m_nCols;
        m_nSizeT[2] = B.m_nRows;
        m_nSizeT[3] = B.m_nCols;

        // Operacion Broadcast (enviar) el tamaño de la matriz
        MPI_Bcast((void *)&m_nSizeT, 4, MPI_INT, 0,
                MPI_COMM_WORLD);

        // Particion que corresponde a cada proceso
        m_nPart = A.m_nRows / m_nSize;

        // Numero de filas que les corresponde a cada proceso
        m_nOffset = A.m_nRows - m_nPart * (m_nSize - 1);

        // Enviar a cada proceso la parte de A y B que le
        // corresponda segun el m_nOffset
        for(int nDest = 1; nDest < m_nSize; nDest++)
        {
                MPI_Send((void *)A.m_ppRows[m_nOffset],
                        m_nPart*A.m_nCols, MPI_DOUBLE,
                        nDest, TAG_MATRIX_PARTITION,
                        MPI_COMM_WORLD);

                MPI_Send((void *)B.m_ppRows[m_nOffset],
                        m_nPart*B.m_nCols, MPI_DOUBLE,
                        nDest, TAG_MATRIX_PARTITION,
                        MPI_COMM_WORLD);

                m_nOffset += m_nPart;
        }
        // Cambio el numero de filas de A y B para procesarlas en
        // el proceso maestro (Ej: si son 10 procesos, y A tiene
        // 20 filas,la cantidad que se envia son 18 filas,
        //  y el proceso maestro se queda con 2...
        A.m_nRows -= m_nPart * (m_nSize - 1);
        B.m_nRows -= m_nPart * (m_nSize - 1);
}
```

Figura 5-33. (Parte 1) Código de la función `fnParallel_DoSLS`

```
else
{

//////////////////////////////////////////////////////////
///////////////Operaciones en otros procesos//////////////
//////////////////////////////////////////////////////////

        // Operación Broadcast (recibir) el tamaño de la matriz
        MPI_Bcast((void *)&m_nSizeT, 4, MPI_INT, 0,
                MPI_COMM_WORLD);

        if (m_nSizeT[0] == 0) return;

        // Cantidad de filas a procesar
        m_nPart = m_nSizeT[0] / m_nSize;

        // i0 permite encontrar el elemento de la diagonal
        // principal
        // Filas de A - (n procesos - id proceso) * porcion
        i0 = m_nSizeT[0] - (m_nSize - m_nRank) * m_nPart;

        // Inicializar matrices en memoria
        A = fnCreateMatrix(m_nPart, m_nSizeT[1]);
        B = fnCreateMatrix(m_nPart, m_nSizeT[3]);
        C = fnCreateMatrix(1, m_nSizeT[1]);
        D = fnCreateMatrix(1, m_nSizeT[3]);

        // Operación Broadcast (recibir) porciones de las
        // matrices A y B
        MPI_Recv((void *)A.m_pData, m_nPart*A.m_nCols,
                MPI_DOUBLE, 0,
                TAG_MATRIX_PARTITION, MPI_COMM_WORLD, &m_Status);

        MPI_Recv((void *)B.m_pData, m_nPart*B.m_nCols,
                MPI_DOUBLE, 0,
                TAG_MATRIX_PARTITION, MPI_COMM_WORLD, &m_Status);
}
```

Figura 5-33. (Parte 2) Código de la función `fnParallel_DoSLS`

```
// m_nOffset nos permite conocer la cantidad de filas que cada
// proceso maneja
m_nOffset = m_nSizeT[0] - m_nPart * (m_nSize - 1);

// Barrido de matriz
for (unsigned int i = 0; i < m_nSizeT[0]; i++)
{
        // Si i es menor que offset, estamos en las
        // filas que el proceso maestro procesa
        if (i < m_nOffset)
                m_nProcess = 0;
        // m_nProcess contiene el identificador de proceso que
        // realiza operaciones y se calcula en base a la
        //  porcion de filas que le corresponde y al offset
        else m_nProcess = (i - m_nOffset) / m_nPart + 1;

        // si el rango es igual al lugar en el que le corresponde
        // dar procesamiento
        if (m_nRank == m_nProcess)
        {
                // Enviar fila pivot de A
                MPI_Bcast((void *)A.m_ppRows[i - i0], A.m_nCols,
                        MPI_DOUBLE, m_nProcess, MPI_COMM_WORLD);


                // Enviar fila pivot de B
                MPI_Bcast((void *)B.m_ppRows[i - i0], B.m_nCols,
                        MPI_DOUBLE, m_nProcess, MPI_COMM_WORLD);

                // Se realiza la sustitucion de elementos
                // correspondiente
                fnDoMatrixGauss(A, B, i-i0, i, A.m_ppRows[i - i0],
                                B.m_ppRows[i - i0]);
        }
        else
        {
                // Recibir fila pivot de A
                MPI_Bcast((void *)C.m_ppRows[0], C.m_nCols,
                        MPI_DOUBLE, m_nProcess, MPI_COMM_WORLD);

                // Recibir fila pivot de B
                MPI_Bcast((void *)D.m_ppRows[0], D.m_nCols,
                        MPI_DOUBLE, m_nProcess, MPI_COMM_WORLD);

                // Se realiza la sustitucion de elementos
                // correspondiente como no es pivot, -1 evita
                //  que se de tratamiento de pivot
                // a la fila que debe procesar este proceso
                fnDoMatrixGauss(A, B, -1, i, C.m_ppRows[0],
                                D.m_ppRows[0]);
        }
}
```

Figura 5-33. (Parte 3) Código de función la `fnParallel_DoSLS`

```
////////////////////////////////////////////////////////////////
/////////////////            Resultados       ////////////////
////////////////////////////////////////////////////////////////

if(fnIsRootProcess())
{
        // Restaura el tamaño original de matriz A y B
        A.m_nRows = m_nSizeT[0];
        B.m_nRows = m_nSizeT[2];

        // Se obtiene de cada proceso el resultado de la matriz B
        // que le correspondio procesar
        m_nOffset = A.m_nRows - m_nPart * (m_nSize - 1);
        for(int nSource = 1; nSource < m_nSize; nSource++)
        {
                MPI_Recv((void *)B.m_ppRows[m_nOffset],
                        m_nPart*B.m_nCols, MPI_DOUBLE,
                        nSource, TAG_MATRIX_PARTITION,
                        MPI_COMM_WORLD, &m_Status);
                m_nOffset += m_nPart;
        }
}
else
{
////////////////////////////////////////////////////////////
///////////////Operaciones en otros procesos///////////////
////////////////////////////////////////////////////////////

        // Enviar resultado de regreso
        MPI_Send((void *)B.m_pData, B.m_nRows*B.m_nCols,
                        MPI_DOUBLE, 0, TAG_MATRIX_PARTITION,
                        MPI_COMM_WORLD);
}
// Esperar que todos los procesos finalicen sus operaciones STOP
MPI_Barrier(MPI_COMM_WORLD);
```

Figura 5-33. (Parte 4) Código de la función `fnParallel_DoSLS`

Implementa también la función **`fnParallel_DoPower(CMatrix& A, CMatrix& B, CMatrix& Result, int nPower = 1)`**, que permite elevar a una potencia la matriz **B**. La matriz **B** representa la matriz base de la operación potencia y la matriz **A** es una copia de la matriz **B**; **Result** almacena el resultado parcial de dicha operación y **nPower** representa el exponente al cual se eleva la matriz **B**. En la Figura 5-34 se muestra el código fuente de esta función.

```
if(fnIsRootProcess())
{
        m_nSizeT[0] = A.m_nRows;
        m_nSizeT[1] = A.m_nCols;
        m_nSizeT[2] = B.m_nRows;
        m_nSizeT[3] = B.m_nRows;

        // Operacion Broadcast (enviar) tamaño de matriz,
        MPI_Bcast((void*)&m_nSizeT, 4, MPI_INT, 0, MPI_COMM_WORLD);

        // Enviar a cada proceso una parte de A
        m_nPart = A.m_nRows / m_nSize;
        m_nOffset = A.m_nRows - m_nPart * (m_nSize - 1);

        // Se envia a cada proceso su porcion correspondiente
        // de la matriz A
        for (int nDest = 1; nDest < m_nSize; nDest++)
        {
                MPI_Send((void *)A.m_ppRows[m_nOffset],
                        m_nPart*A.m_nCols, MPI_DOUBLE,
                        nDest, TAG_MATRIX_PARTITION,
                        MPI_COMM_WORLD);
                m_nOffset += m_nPart;
        }

        // En el proceso maestro se redimensiona el tamaño de A
        // para realizar las operaciones
        A.m_nRows = A.m_nRows - m_nPart * (m_nSize - 1);
        Result.m_nRows = Result.m_nRows - m_nPart * (m_nSize -1);

        // Operacion Broadcast (enviar) matriz B
        MPI_Bcast((void *)B.m_pData, B.m_nRows*B.m_nCols,
                MPI_DOUBLE, 0, MPI_COMM_WORLD);

        // Realizar la operacion potencia
        fnDoMatrixPower(A, B, Result, nPower);

        // Se vuelve a la dimension original de A
        A.m_nRows = m_nSizeT[0];
        Result.m_nRows = m_nSizeT[0];

        // Se obtiene el offset para recibir la porcion
        // asignada a cada proceso
        m_nOffset = A.m_nRows - m_nPart * (m_nSize - 1);

        // Se recibe de cada proceso la porcion del resultado
        for (int nSource = 1; nSource < m_nSize; nSource++)
        {
                MPI_Recv((void *)Result.m_ppRows[m_nOffset],
                        m_nPart*Result.m_nCols, MPI_DOUBLE,
                        nSource, TAG_MATRIX_PARTITION,
                        MPI_COMM_WORLD, &m_Status);
                m_nOffset += m_nPart;
        }
```

Figura 5-34. (Parte 1) Código de la función `fnParallel_DoPower`

```
else
{
////////////////////////////////////////////////////////
//////////////Operaciones en otros procesos//////////////
////////////////////////////////////////////////////////

        // Operacion Broadcast (recibir) el tamaño de la matriz
        MPI_Bcast((void *)&m_nSizeT, 4, MPI_INT, 0,
                MPI_COMM_WORLD);
        m_nPart = m_nSizeT[0] / m_nSize;

        // Inicializar matrices en memoria
        A = fnCreateMatrix(m_nPart, m_nSizeT[1]);
        B = fnCreateMatrix(m_nSizeT[2], m_nSizeT[3]);

        // Se debe redimensionar la matriz Result
        Result.m_nRows = A.m_nRows;

        // Operacion Send (recibir) una parte de la matriz A
        MPI_Recv((void *)A.m_pData, m_nPart*A.m_nCols,
                MPI_DOUBLE, 0,
                TAG_MATRIX_PARTITION, MPI_COMM_WORLD, &m_Status);

        // Operacion Broadcast (recibir) matriz B
        MPI_Bcast((void *)B.m_pData, B.m_nRows*B.m_nCols,
                MPI_DOUBLE, 0, MPI_COMM_WORLD);

        // Realizar la operacion potencia
        fnDoMatrixPower(A, B, Result, nPower);

        // Se envia la matriz resultado al proceso raiz
        MPI_Send((void *)Result.m_pData, m_nPart*Result.m_nCols,
                MPI_DOUBLE, 0,
                TAG_MATRIX_PARTITION, MPI_COMM_WORLD);
}

// Esperar que todos los procesos finalicen sus operaciones STOP
MPI_Barrier(MPI_COMM_WORLD);
```

Figura 5-34. (Parte 2) Código de la función `fnParallel_DoPower`

Además, implementa la función **`fnParallel_DoTaylorSerie(CMatrix &I, CMatrix& A, CMatrix& B, CMatrix& Result, double dTime)`**, que permite obtener la Serie de Taylor. La matriz **I** representa la matriz identidad. Las matrices **A** y **B** permiten realizar la potencia de los diferentes términos involucrados en la serie. **Result** almacena el resultado de la serie y **dTime** es el escalar involucrado en la serie. Esta función realiza una aproximación de la serie de Taylor debido a que sólo hace uso de los primeros 25 términos de la serie. En la Figura 5-35 se muestra el código fuente de esta función.

```
CMatrix Temp1, Temp2;
float nFactorial = 1;
double dTemp = 0.0;
if(fnIsRootProcess())
{
        m_nSizeT[0] = A.m_nRows;
        m_nSizeT[1] = A.m_nCols;
        m_nSizeT[2] = B.m_nRows;
        m_nSizeT[3] = B.m_nRows;

        // Operacion Broadcast (enviar) tamaño de matriz,
        MPI_Bcast((void *)&m_nSizeT, 4, MPI_INT, 0,
                MPI_COMM_WORLD);


         // Calcular el tamaño que se envia de A y de I a cada
         // proceso
         m_nPart = A.m_nRows / m_nSize;
         m_nOffset = A.m_nRows - m_nPart * (m_nSize - 1);

        // Se envia a cada proceso su procion correspondiente de
        // la matriz A y de la matriz I
        for (int nDest = 1; nDest < m_nSize; nDest++)
        {
                MPI_Send((void *)A.m_ppRows[m_nOffset],
                        m_nPart*A.m_nCols, MPI_DOUBLE,
                        nDest, TAG_MATRIX_PARTITION,
                        MPI_COMM_WORLD);
                MPI_Send((void *)I.m_ppRows[m_nOffset],
                        m_nPart*I.m_nCols, MPI_DOUBLE,
                        nDest, TAG_MATRIX_PARTITION,
                        MPI_COMM_WORLD);
                m_nOffset += m_nPart;
        }

        // En el proceso maestro se redimensiona el tamaño de A,
        // de I y Result y se crean las matrices temporales
        // para realizar las operaciones
        A.m_nRows = A.m_nRows - m_nPart * (m_nSize - 1);
        Result.m_nRows = Result.m_nRows - m_nPart * (m_nSize - 1);
        I.m_nRows = I.m_nRows - m_nPart * (m_nSize - 1);
        Temp1 = fnCreateMatrix(A.m_nRows, A.m_nCols);
        Temp2 = fnCreateMatrix(A.m_nRows, A.m_nCols);

        // Operacion Broadcast (enviar) matriz B
        MPI_Bcast((void *)B.m_pData, B.m_nRows*B.m_nCols,
                MPI_DOUBLE, 0, MPI_COMM_WORLD);
```

Figura 5-35. (Parte 1) Código de la función `fnParallel_DoTaylorSerie`

```
        // Realizando la Serie de Taylor
        for(int k = MIN_THRESHOLD; k < MAX_THRESHOLD; k++)
        {
                // Termino k!
                nFactorial = fnGetFactorial(k);
                // Termino t^k
                dTemp = dTime;
                for(int t = 1; t < k; t++)  dTemp *= dTime;
                // Termino (t^k / k!)
                dTemp = ( dTemp / (double) nFactorial );
                // Termino Q^k
                if(k == MIN_THRESHOLD)
                        fnCopyElements(A, Templ);
                else
                        fnDoMatrixPower(A, B, Templ, k);
                // (( t^k ) / k!) * ( Q^k )
                fnDoMatrixMultiplyScalar(dTemp, Templ);

                // Termino k + sumatorio terminos anteriores
                fnDoMatrixAdd(Templ, Temp2, Result);
                fnCopyElements(Result , Temp2);
        }
        // I + Q*t + ((t^k)/k!) * (Q^k)
        fnDoMatrixAdd(I, Temp2, Result);

        // Se vuelve a la dimension original de A y de Result
        A.m_nRows = m_nSizeT[0];
        Result.m_nRows = m_nSizeT[0];

        // Se obtiene el offset para recibir la porcion
        // asignada a cada proceso
        m_nOffset = A.m_nRows - m_nPart * (m_nSize - 1);

        // Se recibe de cada proceso la porcion de la matriz
        // resultado
        for (int nSource = 1; nSource < m_nSize; nSource++)
        {
                MPI_Recv((void *)Result.m_ppRows[m_nOffset],
                        m_nPart*Result.m_nCols, MPI_DOUBLE,
                        nSource, TAG_MATRIX_PARTITION,
                        MPI_COMM_WORLD, &m_Status);
                m_nOffset += m_nPart;
        }
}
```

Figura 5-35. (Parte 2) Código de la función `fnParallel_DoTaylorSerie`

```
else
{
//////////////Operaciones en otros procesos//////////////

        // Operacion Broadcast (recibir) el tamaño de la matriz
        MPI_Bcast((void *)&m_nSizeT, 4, MPI_INT, 0,
                MPI_COMM_WORLD);

        m_nPart = m_nSizeT[0] / m_nSize;
        // Inicializar matrices en memoria
        A = fnCreateMatrix(m_nPart, m_nSizeT[1]);
        B = fnCreateMatrix(m_nSizeT[2], m_nSizeT[3]);
        I = fnCreateMatrix(A.m_nRows, A.m_nCols);
        Temp1 = fnCreateMatrix(A.m_nRows, A.m_nCols);
        Temp2 = fnCreateMatrix(A.m_nRows, A.m_nCols);
        // Se debe redimensionar la matriz Result
        Result.m_nRows = A.m_nRows;
        // Operacion Send (recibir) una parte de la matriz A
        MPI_Recv((void *)A.m_pData, m_nPart*A.m_nCols,
                MPI_DOUBLE, 0,
                TAG_MATRIX_PARTITION, MPI_COMM_WORLD, &m_Status);
        // Operacion Send (recibir) una parte de la matriz I
        MPI_Recv((void *)I.m_pData, m_nPart*I.m_nCols,
                MPI_DOUBLE, 0,
                TAG_MATRIX_PARTITION, MPI_COMM_WORLD, &m_Status);
        // Operacion Broadcast (recibir) matriz B
        MPI_Bcast((void *)B.m_pData, B.m_nRows*B.m_nCols,
                MPI_DOUBLE, 0, MPI_COMM_WORLD);
        for(int k = MIN_THRESHOLD; k < MAX_THRESHOLD; k++)
        {
                // Termino k!
                nFactorial = fnGetFactorial(k);
                // Termino t^k
                dTemp = dTime;
                for(int t = 1; t < k; t++)  dTemp *= dTime;
                // Termino (t^k / k!)
                dTemp = ( dTemp / (double) nFactorial );
                // Termino Q^k
                if(k == MIN_THRESHOLD)  fnCopyElements(A, Temp1);
                else   fnDoMatrixPower(A, B, Temp1, k);
                // (( t^k ) / k!) * ( Q^k )
                fnDoMatrixMultiplyScalar(dTemp, Temp1);
                // Termino k + sumatorio terminos anteriores
                fnDoMatrixAdd(Temp1, Temp2, Result);
                fnCopyElements(Result , Temp2);
        }
        // I + Q*t + ((t^k)/k!) * (Q^k)
        fnDoMatrixAdd(I, Temp2, Result);
        // Se envia la matriz resultado al proceso raiz
        MPI_Send((void *)Result.m_pData, m_nPart*Result.m_nCols,
                MPI_DOUBLE, 0, TAG_MATRIX_PARTITION,
                MPI_COMM_WORLD);
}
// Esperar que todos los procesos finalicen sus operaciones STOP
MPI_Barrier(MPI_COMM_WORLD);
```

Figura 5-35. (Parte 3) Código de la función `fnParallel_DoTaylorSerie`

La función **fnGetFactorial(int k)** permite obtener el factorial de un número **k**, esta función es utilizada en el cálculo de la Serie de Taylor.

La función **fnAbort()** permite finalizar MPI si existiese algún inconveniente. Esta función llama a la función **MPI_Abort**.

Esta clase utiliza el parámetro **MPI_Status** como variable miembro para que las llamadas de la librería de paso de mensajes conozcan el estado de la aplicación.

Además, la clase **CMatrix_Handler** es la clase base de **CMarkovTD** y **CMarkovTC**.

5.3.8. CLASE CMarkovTD

CMarkovTD es una clase de C++ derivada de **CMatrix_Handler**. En la clase **CMarkovTD** se implementan las funciones que permiten realizar operaciones para determinar la solución de Cadenas de Markov a Tiempo Discreto, así como funciones para presentar en consola los resultados y almacenarlos en un archivo.

La función **fnGetDistributionToStepN(int)** permite determinar la distribución al paso *n* y almacenar esta distribución en un arreglo **MatrixArray**, para luego poder presentar en consola y almacenarla en un archivo. El parámetro de entrada permite especificar el paso al cual se quiere obtener la distribución. En la Figura 5-36 se muestra el código fuente de esta función.

```
if(fnIsRootProcess())
{
        fnCopyElements(m_A, Temp2);
        // Almacenar tiempo de inicializacion
        m_dTime0 = MPI_Wtime();
}
// Paraleliza el calculo de valor potencia de
// de matrices
fnParallel_DoPower(m_A, Temp2, Temp1, nPaso);
if(fnIsRootProcess())
{
        // Almacenar tiempo de finalizacion
        m_dTime1 = MPI_Wtime();
        m_dTimeArray.push_back(m_dTime1 - m_dTime0);
}
Temp1.m_nRows = 1;
return Temp1;
```

Figura 5-36. Código de la función `fnGetDistributionToStepN`

Para especificar el nombre del archivo que contiene la matriz, se creó el archivo `config.txt`; adicionalmente, en este archivo se debe especificar el paso al cual se quiere evaluar la CMTD.

La función **`fnPreviousOperation(CMatrix& A, CMatrix& B)`** permite generar el sistema de ecuaciones a partir de la ecuación:

$$\mathbf{p} = \mathbf{p} \cdot \mathbf{P} \tag{5-21}$$

donde $\mathbf{p} = [\pi_1 \quad \pi_2 \quad ...]$, y utilizando la condición de normalización para que el sistema de ecuaciones resultante tenga solución. Las matrices **A** y **B** sirven para almacenar la matriz de conexión y el vector conocido que se generarán después de ejecutar esta función. El código de esta función se muestra en la Figura 5-37.

```
int nRowActual = (int) m_A.m_nRows - 1;

m_B = fnCreateMatrix(m_A.m_nRows, 1);

// Realizar Transpuesta de A
if(!fnDoMatrixTranpose(m_A)) return;

// Fila i = j restar 1
for(int i = 0; i < (int) m_A.m_nRows; i++)
{
	for(int j = 0; j < (int) m_A.m_nCols; j++)
	{
		if( i == j )
		{
			m_A.m_ppRows[i][j] = m_A.m_ppRows[i][j] - 1;
		}
		// Ultima fila todos 1
		m_A.m_ppRows[nRowActual][j] = 1.0;
	}
}

for(int x = 0; x < (int) m_B.m_nRows; x++)
	m_B.m_ppRows[x][0] = 0.0;

m_B.m_ppRows[nRowActual][0] = 1.0;
```

Figura 5-37. Código de la función `fnPreviousOperation`

La función **`fnGetDistributionR()`** permite determinar la distribución de régimen. El código de esta función se muestra en la Figura 5-38.

```
if(fnIsRootProcess())
	// Almacenar tiempo de inicializacion
	m_dTime0 = MPI_Wtime();
fnParallel_DoSLS();
if(fnIsRootProcess())
{
	// Almacenar tiempo de finalizacion
	m_dTime1 = MPI_Wtime();
	m_dTimeArray.push_back(m_dTime1 - m_dTime0);
}
```

Figura 5-38. Código de la función `fnGetDistributionR`

Adicionalmente, esta clase tiene las funciones **`fnPrintResultsPI(ostream &output)`**, que permite presentar la distribución obtenida en diferentes pasos en consola y también permite almacenar en archivo los resultados obtenidos, y **`fnPrintResultsR(ostream &output)`** que permite presentar la distribución de

régimen en consola y almacenar los resultados obtenidos en un archivo denominado `ResultadosTD.txt`.

Las funciones **`fnGetDistributiontoStepsN(int)`**, **`fnPrintResultsPI()`**, **`fnGetDistributionR()`** y **`fnPrintResultsR()`** son llamadas en la función **`fnGetResults()`**.

5.3.9. CLASE `CMarkovTC`

`CMarkovTC` es una clase de C++ derivada de **`CMatrix_Handler`**. En la clase **`CMarkovTC`** se implementan las funciones que permiten realizar operaciones para determinar la solución de Cadenas de Markov a Tiempo Continuo, así como funciones para presentar en consola los resultados y almacenarlos en un archivo.

La función **`fnDoTaylorSerie(CMatrix& Result)`** permite obtener la Serie de Taylor llamando a la función de la clase base. Esta función prepara las matrices que van a ser paralelizadas. En el parámetro **`Result`** se almacena el resultado de la serie. El código de esta función se presenta en la Figura 5-39.

```
CMatrix mQcopy;
mQcopy = fnCreateMatrix(m_A.m_nRows, m_A.m_nCols);
fnCopyElements(m_A, mQcopy);

fnParallel_DoTaylorSerie(I, m_A, mQcopy, Result, m_dTime);

I.fnFinalizeMatrix();
mQcopy.fnFinalizeMatrix();
```

Figura 5-39. Código de la función `fnDoTaylorSerie`

Esta clase tiene una variable miembro de la clase **`CMatrix`** que representa a la matriz identidad **`I`**.

Para especificar el nombre del archivo que contiene la matriz, se creo el archivo `config.txt`; adicionalmente, en este archivo se debe especificar el tiempo al

cual se quiere evaluar la CMTC. La función **`fnGetDistributionP(CMatrix& Result)`** permite determinar la matriz de probabilidades de transición. Esta función obtiene la matriz identidad y calcula la Serie de Taylor. El código de está función se presenta en Figura 5-40.

```
if(fnIsRootProcess())
        // Almacenar tiempo de inicializacion
        m_dTime0 = MPI_Wtime();

// Obtiene la matrix identidad
fnDoMatrixIdentity(I);
// Calcular la serie de Taylor
fnDoTaylorSerie(Result);
// Almacenar tiempo de finalizacion
if(fnIsRootProcess())
{
        m_dTime1 = MPI_Wtime();
        m_dTimeArray.push_back(m_dTime1 - m_dTime0);
}
```

Figura 5-40. Código de la función `fnGetDistributionP`

La función **`fnPreviousOperation(CMatrix& A, CMatrix& B)`** permite generar el sistema de ecuaciones a partir de la ecuación:

$$\mathbf{p} \cdot \mathbf{Q} = 0 \tag{5-22}$$

donde $\mathbf{p} = [\pi_1 \quad \pi_2 \quad ...]$, y utilizando la condición de normalización para que el sistema de ecuaciones resultante tenga solución. Las matrices **A** y **B** sirven para almacenar la matriz de conexión y el vector conocido, respectivamente, que se generarán después de ejecutar esta función. El código de esta función se muestra en la Figura 5-41.

```
int nRowActual = (int) m_A.m_nRows - 1;

m_B = fnCreateMatrix(m_A.m_nRows, 1);

// Realizar Transpuesta de A
if(!fnDoMatrixTranpose(m_A)) return;

for(int j = 0; j < (int) m_A.m_nCols; j++)
{
        // Ultima fila todos 1
        m_A.m_ppRows[nRowActual][j] = 1.0;
}

// Valores de matriz ampliada iguales a 0
for(int x = 0; x < (int) m_B.m_nRows; x++)
        m_B.m_ppRows[x][0] = 0.0;

// Cambiar los valores de ultima fila a 1
m_B.m_ppRows[nRowActual][0] = 1.0;
```

Figura 5-41. Código de la función `fnPreviousOperation`

Para obtener la distribución de régimen, se utiliza la función **`fnGetDistributionR()`**. El código de esta función se muestra en la Figura 5-42.

```
if(fnIsRootProcess())
        // Almacenar tiempo de inicializacion
        m_dTime0 = MPI_Wtime();

fnParallel_DoSLS();

if(fnIsRootProcess())
{
        // Almacenar tiempo de finalizacion
        m_dTime1 = MPI_Wtime();
        m_dTimeArray.push_back(m_dTime1 - m_dTime0);
}
```

Figura 5-42. Código de la función `fnGetDistributionR`

Adicionalmente, esta clase tiene las funciones **`fnPrintResultsPT(ostream &output)`**, que permite presentar el vector de estado al tiempo *t* y la matriz de probabilidades de transición **P**(*t*), tanto en consola como en archivo; y **`fnPrintResultsR(ostream &output)`** que permite presentar la distribución de régimen en consola y almacenar los resultados obtenidos en un archivo denominado `ResultadosTC.txt`.

Las funciones **`fnGetDistributionP()`**, **`fnPrintResultsPT()`**, **`fnGetDistributionR()`** y **`fnPrintResultsR()`** son llamadas en la función **`fnGetResults()`**.

5.3.10. INTERFAZ DE USUARIO

Para poder ejecutar la aplicación desarrollada se implementó una interfaz gráfica creada bajo **Qt Designer**, una herramienta que permite diseñar e implementar interfaces con el conjunto de herramientas GUI multiplataforma de **Qt**. **Qt** ha sido desarrollado usando C++ y provee portabilidad de código en varios sistemas operativos como Windows, Mac OS X, Linux y Unix.

Se recomienda al lector revisar el sitio Web de **Qt**: http://www.trolltech.com/qt así como el libro *"C++ GUI Programming with Qt 3"* escrito por Jasmin Blanchette y Mark Summerfield y publicado por la editorial Pearson Education, Inc., para obtener mayor información acerca del diseño e implementación de interfaces utilizando la herramienta **Qt**.

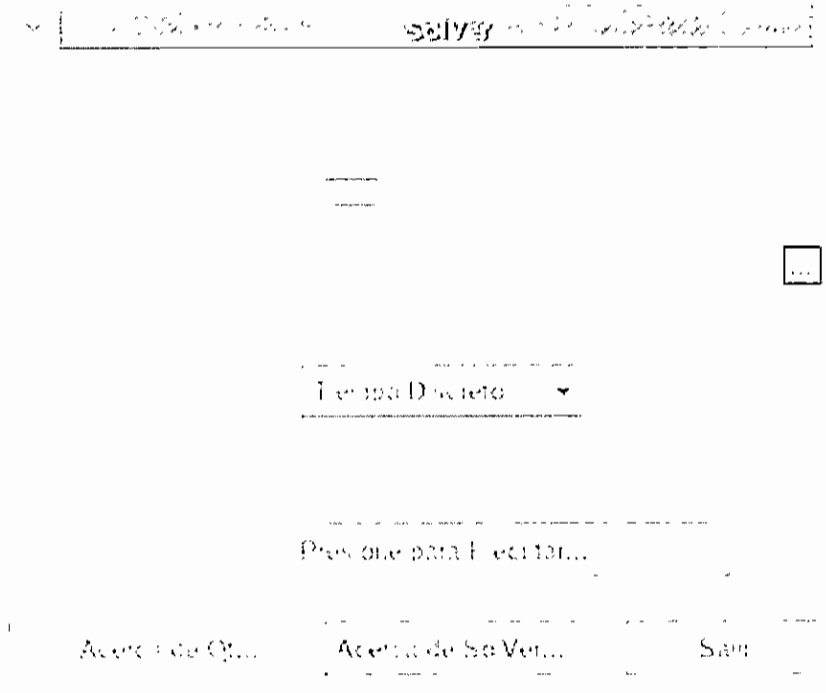


Figura 5-43. Interfaz gráfico de la aplicación desarrollada

Se asignó el nombre **SolVer** al interfaz gráfico que permite interactuar con **mpirun** y la aplicación que permite resolver las Cadenas de Markov en consola. En la Figura 5-43 se muestra el interfaz gráfico implementado con **Qt Designer**.

SolVer permite guardar en el archivo `config.txt` el nombre del archivo que contiene la matriz **P** o la matriz **Q**(*t*); además, almacena el número de pasos o el tiempo en el que se evalúa la Cadena de Markov. Al presionar el botón <Presione para Ejecutar...>, se ejecuta el comando **mpirun** con la opción **–np** igual a la cantidad de procesos especificada mediante el control deslizador (*slider*).

En la Figura 5-44 se muestra una breve descripción de la funcionalidad de cada elemento del interfaz gráfico desarrollado.

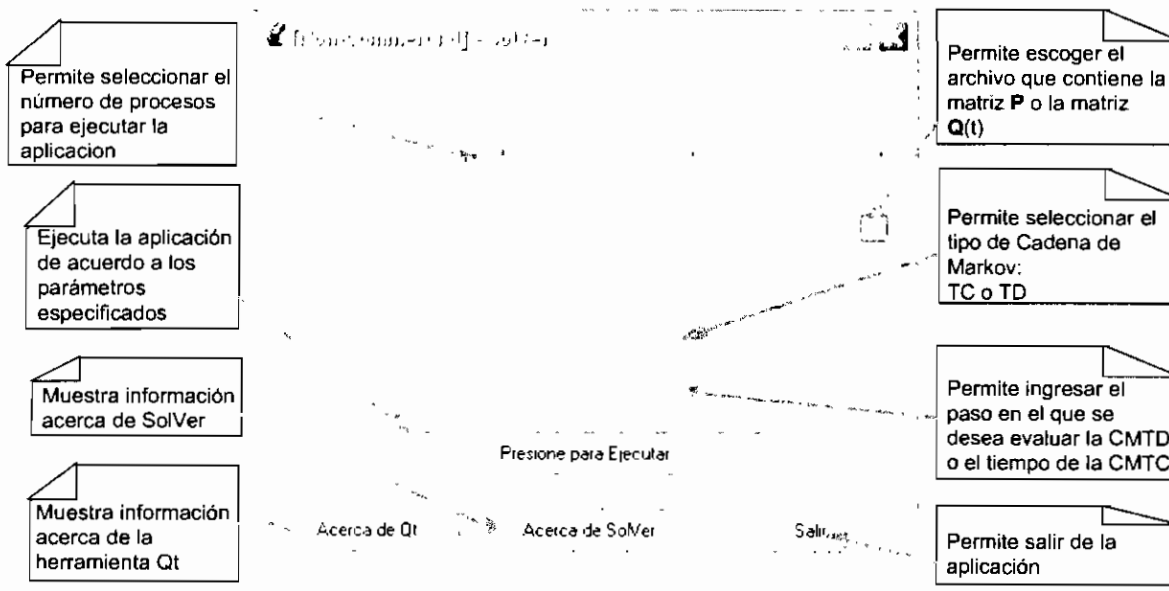


Figura 5-44. Descripción del interfaz gráfico de la aplicación desarrollada

5.3.11. COMPILACIÓN DE LA APLICACIÓN

Para compilar la aplicación utilizando OSCAR, se utilizó el comando de MPICH **mpiCC**[16] el cual tiene soporte para C++.

[16] **mpiCC** es una utilidad de MPICH que simplifica la compilación utilizando el compilador de C++.

El comando utilizado para compilar la aplicación que resuelve Cadenas de Markov a Tiempo Discreto es:

```
$ mpiCC -o tesistd main.cpp Matrix.cpp Matrix_Handler.cpp MarkovTD.cpp
```

El comando utilizado para compilar la aplicación que resuelve Cadenas de Markov a Tiempo Continuo es:

```
$ mpiCC -o tesistc main.cpp Matrix.cpp Matrix_Handler.cpp MarkovTC.cpp
```

Se compiló la aplicación en *Rocks* utilizando el comando de MPICH **mpicxx**[17] el cual tiene soporte para C++.

El comando utilizado para compilar la aplicación que resuelve Cadenas de Markov a Tiempo Discreto es:

```
$ mpicxx -o tesistd main.cpp Matrix.cpp Matrix_Handler.cpp MarkovTD.cpp
```

El comando utilizado para compilar la aplicación que resuelve Cadenas de Markov a Tiempo Continuo es:

```
$ mpicxx -o tesistc main.cpp Matrix.cpp Matrix_Handler.cpp MarkovTC.cpp
```

Para compilar la interfaz gráfica que fue desarrollada con **Qt**, se utilizó el comando **qmake**[18] y el comando **make**[19].

---

[17] **mpicxx** es una utilidad de MPICH que permite compilar programas con MPI Versión 2.

[18] **qmake** crea un archivo denominado `makefile` en base al nombre del proyecto desarrollado en **Qt** con información para la construcción del ejecutable.

[19] **make** lee la información del archivo `makefile` para construir la aplicación.

```
# qmake interfaz.pro
# make
```

5.4. COMPARACIÓN DE LOS RESULTADOS OBTENIDOS SOBRE EL *CLUSTER* Y SOBRE UN SÓLO COMPUTADOR

A continuación se presentan los resultados obtenidos al ejecutar la aplicación desarrollada usando OSCAR y Rocks. La aplicación fue ejecutada utilizando un sólo computador (1 procesador), y sobre el *cluster* usando 2 computadores (2 procesadores) y 3 computadores (3 procesadores).

Para probar la funcionalidad de la aplicación, se realizaron pruebas con matrices de orden 3, 30, 60, 90, 120, 150, 210, 270, 330, 390, 450, 510 y 600. Se realizaron pruebas resolviendo Cadenas de Markov a Tiempo Discreto y a Tiempo Continuo. Para facilitar la generación de matrices se desarrollaron dos programas simples que generan matrices basándose en las restricciones descritas en las Secciones 5.3.4.1.1 y 5.3.4.1.2. El código de la función que permite generar la matriz de probabilidades de transición se muestra en la Figura 5-45, y el código de la función que permite generar la matriz de tasas de transición se presenta en la Figura 5-46.

```
CMatrix CMatrix::fnCreateMatrix(int nRows, int nCols)
{
        m_nRows = nRows;
        m_nCols = nCols;
        m_pData = new double[nRows * nCols];
        m_ppRows = new double*[nRows];
        for(int i = 0; i < (int) m_nRows; i++)
        {
                for(int j = 0; j < (int) m_nCols; j++)
                {
                        m_ppRows[i] = m_pData + (i * m_nCols);
                        m_ppRows[i][j] = 1.0 / nRows;
                }
        }
    return *this;
}
```

Figura 5-45. Código para generar la matriz de probabilidades de transición

```
CMatrix CMatrix::fnCreateMatrix(int nRows, int nCols)
{
        m_nRows = nRows;
        m_nCols = nCols;
        m_pData = new double[nRows * nCols];
        m_ppRows = new double*[nRows];
        for(int i = 0; i < (int) m_nRows; i++)
        {
                for(int j = 0; j < (int) m_nCols; j++)
                {
                        m_ppRows[i] = m_pData + (i * m_nCols);
                        if(i == j)
                                m_ppRows[i][j] = - (m_nRows - 1.0) / nRows;
                        else
                                m_ppRows[i][j] = 1.0 / nRows;
                }
        }
    return *this;
}
```

Figura 5-46. Código para generar la matriz de tasas de transición

En las Secciones 5.4.1 y 5.4.2 se presentan los resultados obtenidos al ejecutar la aplicación sobre OSCAR y *Rocks*; y en la Sección 5.4.3 se presenta el análisis de los resultados obtenidos.

5.4.1. RESULTADOS OBTENIDOS CON OSCAR

Para evaluar las Cadenas de Markov a Tiempo Discreto, se obtuvo la distribución en el paso 25 y la distribución de régimen. Se tomó el tiempo de ejecución de la aplicación, el tiempo que tardó en obtener la distribución en el paso 25 y el tiempo que tomó en calcular la distribución de régimen.

En la Tabla 5-5 se presentan los resultados obtenidos al calcular la distribución en el paso 25 y en la Figura 5-47 se realiza una comparación de dichos resultados con 1, 2 y 3 procesadores.

En la Tabla 5-6 se presentan los resultados obtenidos al calcular la distribución de régimen y en la Figura 5-48 se realiza una comparación de dichos resultados con 1, 2 y 3 procesadores.

En la Tabla 5-7 se presentan los tiempos de ejecución que tomó la resolución de la CMTD y en la Figura 5-49 se realiza una comparación de dichos resultados.

Para visualizar de mejor manera las gráficas del tiempo requerido para obtener la distribución de régimen, la distribución al tiempo o al paso y el tiempo de ejecución en función del orden de la matriz (**P** o **Q**(t)), se presentan dos figuras por cada tabla. La primera figura presenta los valores obtenidos al utilizar matrices de orden bajo (3 a 150); y la segunda figura muestra los resultados obtenidos para las matrices de mayor orden (210 a 600).

Orden de la matriz P	Procesos 1	2	3
3	2,80E-3	2,94E-4	3,67E-4
30	8,61E-3	4,89E-3	3,52E-3
60	6,12E-2	3,17E-2	2,23E-2
90	1,88E-1	1,04E-1	1,07E-1
120	4,38E-1	2,25E-1	1,53E-1
150	8,29E-1	4,42E-1	3,00E-1
210	3,09E+0	1,63E+0	1,07E+0
270	6,90E+0	3,47E+0	2,34E+0
330	1,33E+1	6,76E+0	4,55E+0
390	2,87E+1	1,49E+1	9,99E+0
450	4,63E+1	2,33E+1	1,56E+1
510	6,79E+1	3,42E+1	2,30E+1
600	9,89E+1	4,96E+1	3,34E+1

Tabla 5-5. Tiempo requerido en segundos para obtener la distribución en el paso 25 con 1, 2 y 3 procesadores usando OSCAR

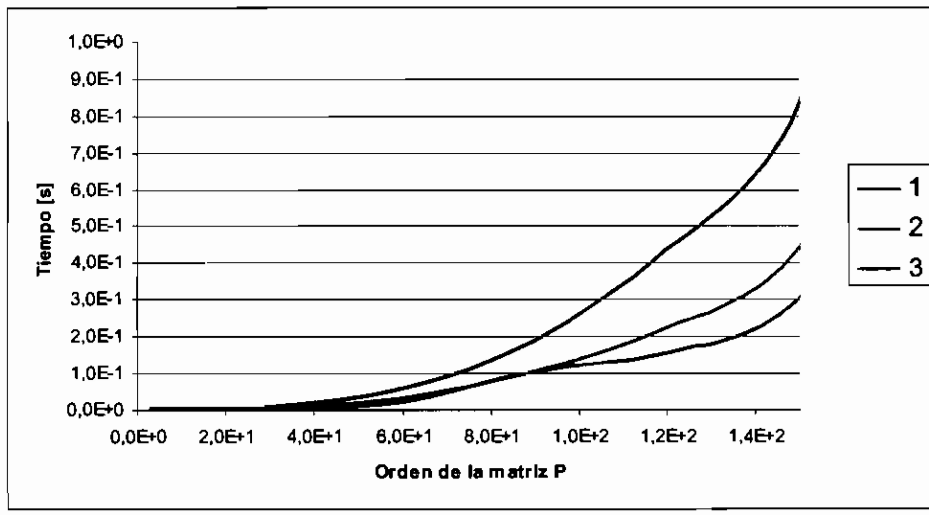


Figura 5-47. (Parte I) Comparación del tiempo requerido con 1, 2 y 3 procesadores para obtener la distribución en el paso 25 usando OSCAR

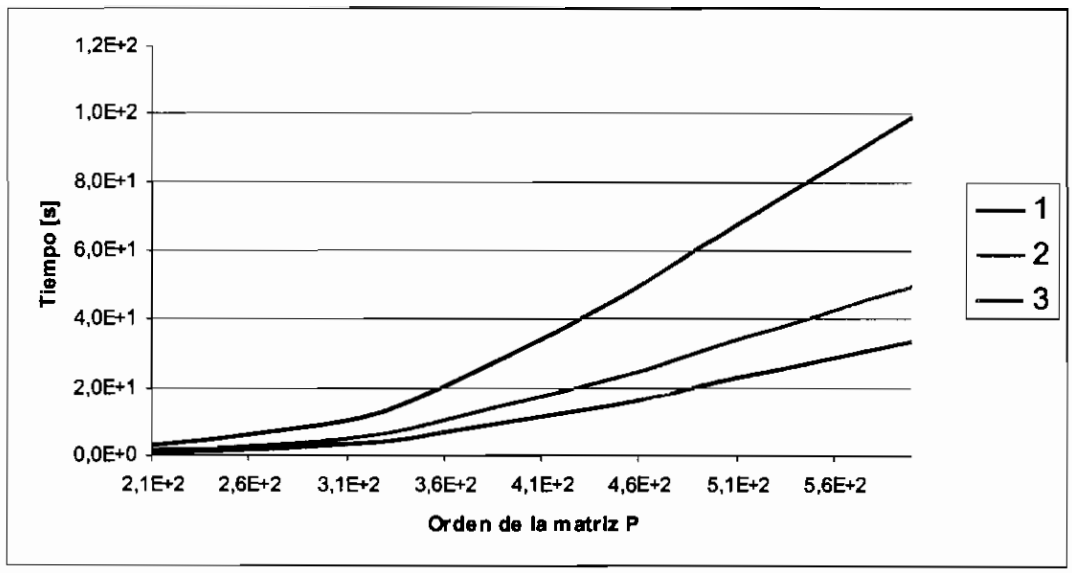


Figura 5-47. (Parte II) Comparación del tiempo requerido con 1, 2 y 3 procesadores para obtener la distribución en el paso 25 usando OSCAR

De los resultados obtenidos, al calcular la distribución en el paso 25, se puede concluir que el tiempo requerido se reduce en aproximadamente un 50% al utilizar dos procesadores y en un valor cercano al 60% al utilizar tres procesadores, si se realiza la comparación con el tiempo requerido al utilizar un sólo procesador.

		Procesos		
		1	2	3
Orden de la matriz P	3	1,30E-5	4,04-e2	2,16E-3
	30	3,72E-4	4,26E-2	8,25E-2
	60	2,53E-3	5,25E-3	7,29E-3
	90	8,10E-3	1,04E-2	1,20E-2
	120	1,86E-2	1,83E-2	2,01E-2
	150	3,59E-2	3,04E-2	3,09E-2
	210	9,70E-2	7,07E-2	9,89E-2
	270	2,07E-1	1,31E-1	1,11E-1
	330	3,92E-1	2,29E-1	1,82E-1
	390	6,72E-1	3,70E-1	2,86E-1
	450	1,06E+0	5,53E-1	4,24E-1
	510	1,50E+0	8,54E-1	5,90E-1
	600	2,45E+0	1,38E+0	9,79E-1

Tabla 5-6. Tiempo requerido en segundos para obtener la distribución de régimen (CMTD) con 1, 2 y 3 procesadores usando OSCAR

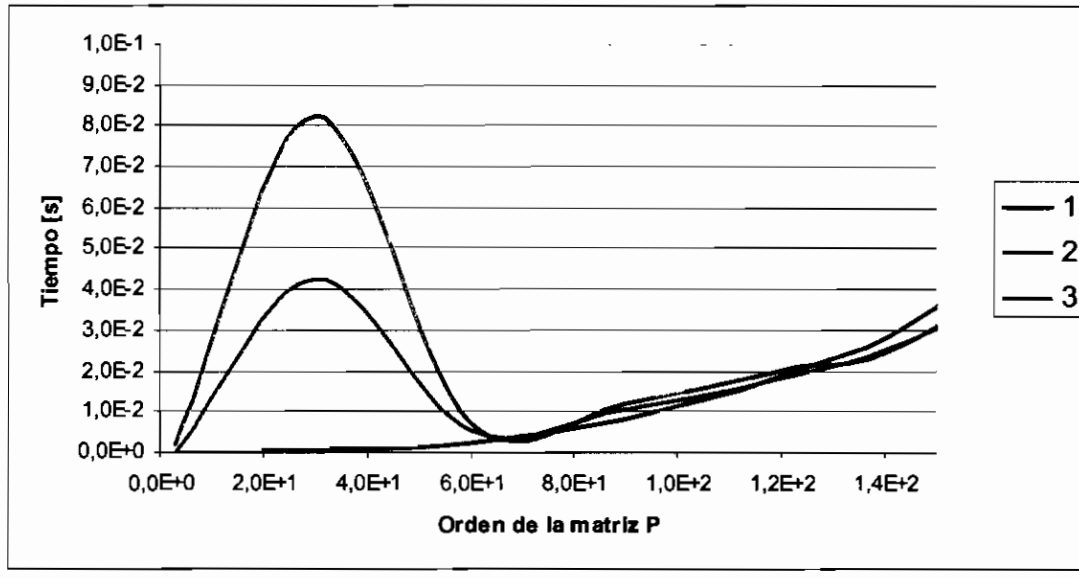


Figura 5-48. (Parte I) Comparación del tiempo requerido con 1, 2 y 3 procesadores para obtener la distribución de régimen (CMTD) usando OSCAR

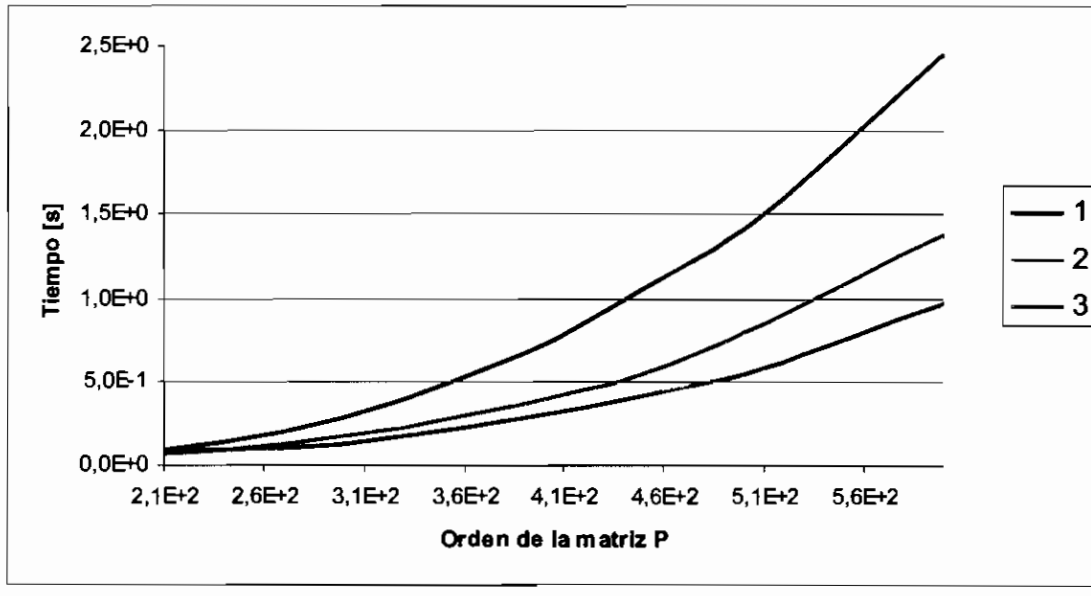


Figura 5-48. (Parte II) Comparación del tiempo requerido con 1, 2 y 3 procesadores para obtener la distribución de régimen (CMTD) usando OSCAR

De la Figura 5-48 (parte II), se puede concluir que el rendimiento mejora en un 50% si se utilizan dos procesadores en lugar de uno, y en un 60% si se utilizan los tres procesadores en lugar de tan sólo uno. En la Figura 5-48 (parte I), se puede apreciar un resultado diferente al obtenido en la Figura 5-48 (parte II), debido a que para las matrices de orden bajo (3 a 150), la cantidad de procesamiento es

baja comparada con las operaciones requeridas para enviar y recibir los datos desde cada proceso.

Orden de la matriz P	Procesos 1	2	3
3	0,00E+0	0,00E+0	0,00E+0
30	1,00E-2	0,00E+0	0,00E+0
60	6,00E-2	3,00E-2	3,00E-2
90	2,00E-1	1,10E-1	8,00E-2
120	4,70E-1	2,90E-1	1,80E-1
150	8,80E-1	7,90E-1	5,56E-1
210	3,21E+0	1,70E+0	1,14E+0
270	7,17E+0	3,63E+0	2,48E+0
330	1,37E+1	7,01E+0	4,77E+0
390	2,95E+1	1,54E+1	1,03E+1
450	4,75E+1	2,39E+1	1,60E+1
510	6,96E+1	3,53E+1	2,36E+1
600	1,02E+2	5,11E+1	3,44E+1

Tabla 5-7. Tiempo en segundos para ejecutar la aplicación que permite resolver CMTD con 1, 2 y 3 procesadores usando OSCAR

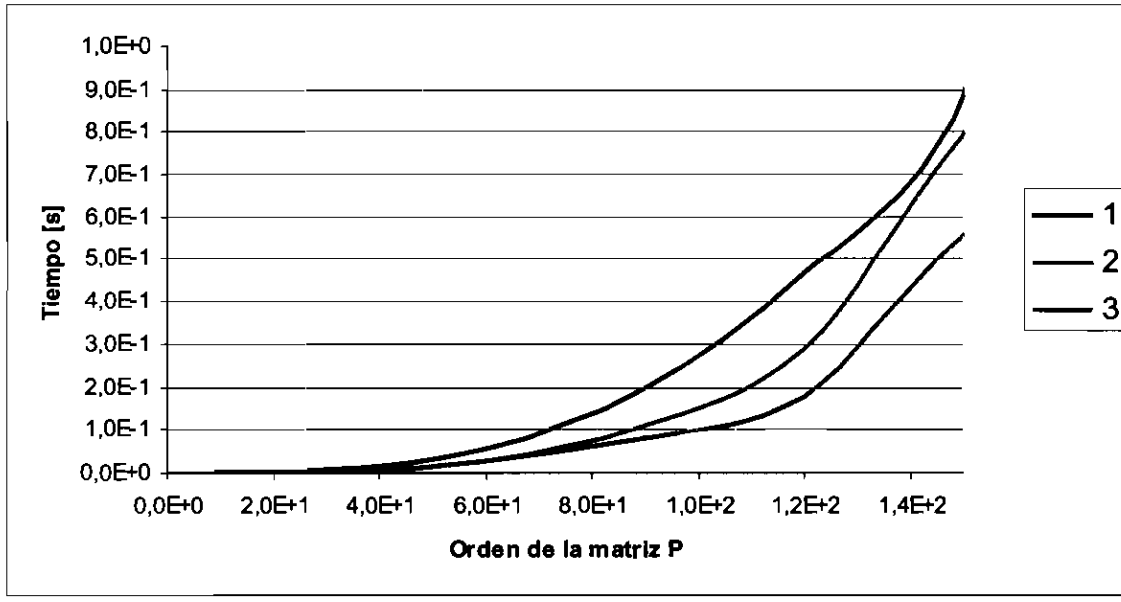


Figura 5-49. (Parte I) Comparación del tiempo requerido con 1, 2 y 3 procesadores para ejecutar la aplicación para resolver CMTD usando OSCAR

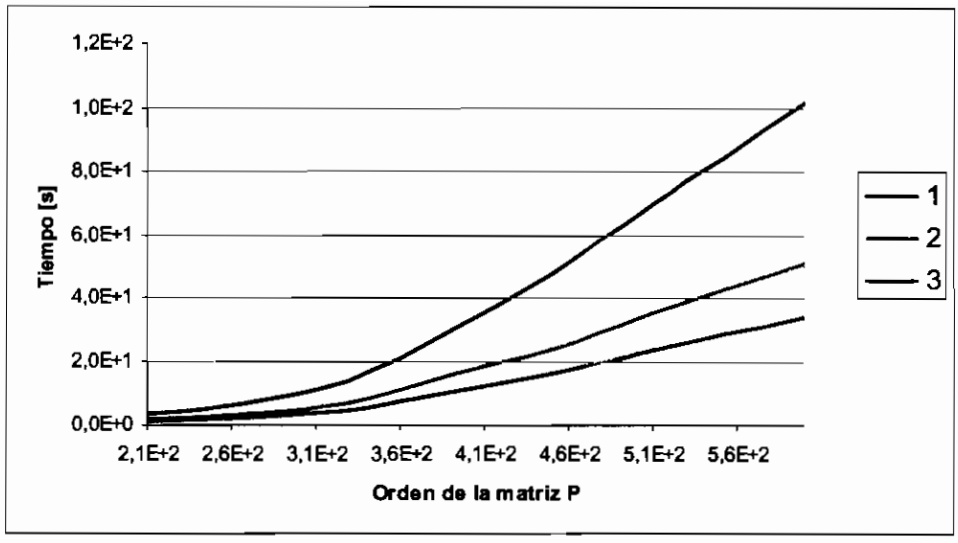


Figura 5-49. (Parte II) Comparación del tiempo requerido con 1, 2 y 3 procesadores para ejecutar la aplicación para resolver CMTD usando OSCAR

De las partes I y II de la Figura 5-49, se puede mencionar que el tiempo de ejecución de la aplicación que permite evaluar CMTDs, se reduce aproximadamente en un 50% si se utilizan dos procesadores y en un valor cercano al 60% al utilizar tres procesadores si se compara con el tiempo de ejecución de la aplicación en un solo procesador.

Para evaluar las Cadenas de Markov a Tiempo Continuo, se obtuvo la distribución en el tiempo 0,25 y la distribución de régimen. Se tomó el tiempo de ejecución de la aplicación, el tiempo que tardó en obtener la distribución en el tiempo 0,25 y el tiempo que tomó el calcular la distribución de régimen.

En la Tabla 5-8 se presentan los resultados obtenidos al calcular la distribución en el tiempo 0,25 y en la Figura 5-50 se realiza una comparación de dichos resultados con 1, 2 y 3 procesadores.

En la Tabla 5-9 se presentan los resultados obtenidos al calcular la distribución de régimen y en la Figura 5-51 se realiza una comparación de dichos resultados con 1, 2 y 3 procesadores.

En la Tabla 5-10 se presentan los tiempos que tomó la resolución de la CMTC y en la Figura 5-52 se realiza una comparación de dichos resultados.

Al igual que en la resolución de CMTD, descrita en los párrafos anteriores, también se presentan dos figuras por cada tabla, para presentar de mejor manera los resultados obtenidos.

		Procesos		
		1	2	3
Orden de la matriz Q(t)	3	2,27E-4	3,94E-4	4,53E-4
	30	9,87E-2	5,06E-2	3,40E-2
	60	7,00E-1	3,55E-1	2,38E-1
	90	2,20E+0	1,10E+0	7,38E-1
	120	4,99E+0	2,55E+0	1,69E+0
	150	9,63E+0	4,84E+0	3,24E+0
	210	3,64E+1	1,81E+1	1,24E+1
	270	8,05E+1	4,18E+1	2,66E+1
	330	1,88E+2	9,40E+1	5,93E+1
	390	3,15E+2	1,72E+2	1,15E+2
	450	5,16E+2	2,67E+2	1,78E+2
	510	7,84E+2	3,84E+2	2,61E+2
	600	1,14E+3	5,68E+2	3,85E+2

Tabla 5-8. Tiempo en segundos requerido para obtener la distribución en el tiempo 0,25 con 1, 2 y 3 procesadores usando OSCAR

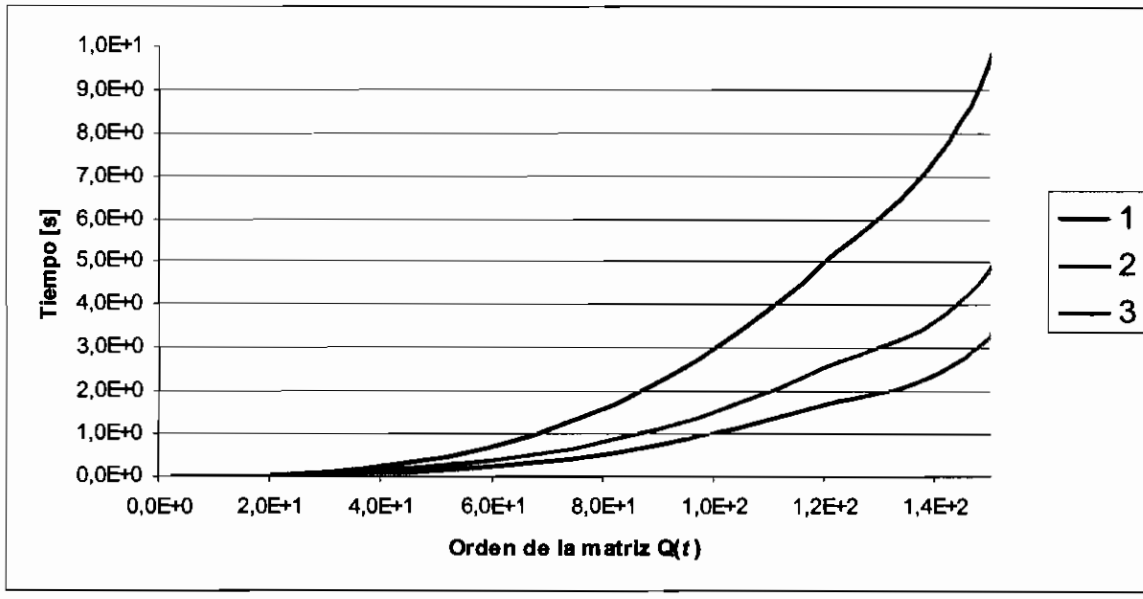


Figura 5-50. (Parte I) Comparación del tiempo requerido con 1, 2 y 3 procesadores para obtener la distribución en el tiempo 0,25 usando OSCAR

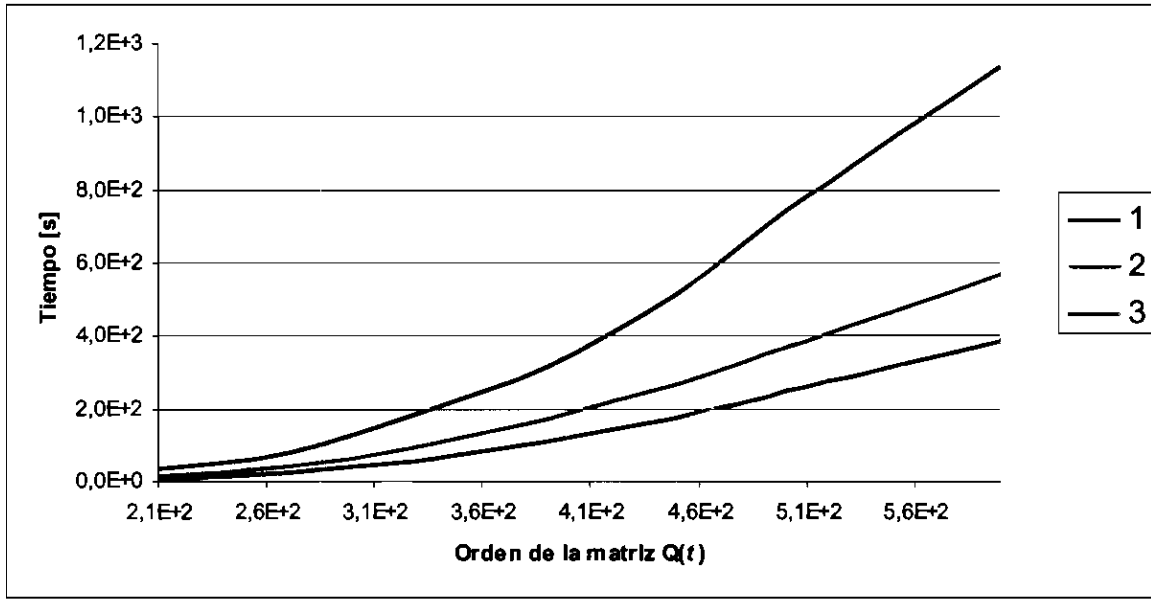


Figura 5-50. (Parte II) Comparación del tiempo requerido con 1, 2 y 3 procesadores para obtener la distribución en el tiempo 0,25 usando OSCAR

El tiempo necesario para obtener la distribución en el tiempo 0,25, utilizando un sólo procesador aproximadamente es dos veces el tiempo requerido si se emplean dos procesadores y casi tres veces si se emplean tres procesadores.

		Procesos		
		1	2	3
Orden de la matriz Q(t)	3	1,20E-5	3,90E-2	2,38E-3
	30	3,50E-4	4,20E-2	4,20E-2
	60	2,46E-3	5,20E-3	7,49E-3
	90	7,98E-3	1,03E-2	1,39E-2
	120	1,85E-2	1,80E-2	1,98E-2
	150	3,57E-2	2,95E-2	3,08E-2
	210	9,65E-2	6,90E-2	6,36E-2
	270	2,12E-1	1,34E-1	1,12E-1
	330	3,86E-1	2,28E-1	1,81E-1
	390	6,81E-1	3,70E-1	2,87E-1
	450	1,05E+0	5,76E-1	4,20E-1
	510	1,50E+0	8,43E-1	5,88E-1
	600	2,45E+0	1,38E+0	9,71E-1

Tabla 5-9. Tiempo en segundos requerido para obtener la distribución de régimen (CMTC) con 1, 2 y 3 procesadores usando OSCAR

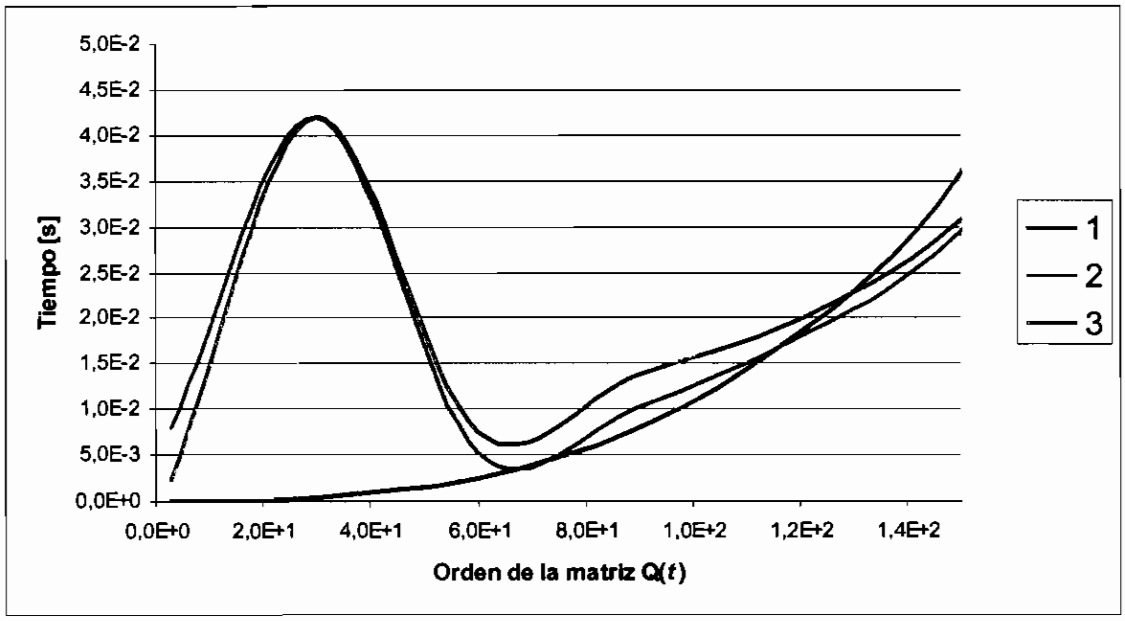


Figura 5-51. (Parte I) Comparación del tiempo requerido con 1, 2 y 3 procesadores para obtener la distribución de régimen (CMTC) usando OSCAR

Figura 5-51. (Parte II) Comparación del tiempo requerido con 1, 2 y 3 procesadores para obtener la distribución de régimen (CMTC) usando OSCAR

De igual manera que en CMTD, las partes I y II de la Figura 5-51 presentan resultados diferentes. La parte I muestra que no existen mejoras en el rendimiento si se utilizan matrices de tamaño bajo (3 a 150), debido al procesamiento requerido para su solución y al tiempo empleado en comunicaciones entre

procesos; mientras que la parte II muestra mejoras en el rendimiento al utilizar matrices de tamaño superior (210 a 600).

Orden de la matriz Q(t)	Procesos		
	1	2	3
3	0,00E+0	0,00E+0	0,00E+0
30	1,10E-1	6,00E-2	4,00E-2
60	7,40E-1	4,00E-1	2,80E-1
90	2,31E+0	1,20E+0	8,40E-1
120	5,18E+0	2,73E+0	1,88E+0
150	9,94E+0	5,10E+0	3,50E+0
210	3,70E+1	1,85E+1	1,27E+1
270	8,16E+1	4,28E+1	2,76E+1
330	1,89E+2	9,55E+1	6,07E+1
390	3,17E+2	1,57E+2	1,11E+2
450	5,20E+2	2,59E+2	1,75E+2
510	7,88E+2	3,97E+2	2,75E+2
600	1,15E+3	5,74E+2	3,84E+2

Tabla 5-10. Tiempo en segundos para ejecutar la aplicación que permite resolver CMTC con 1, 2 y 3 procesadores usando OSCAR

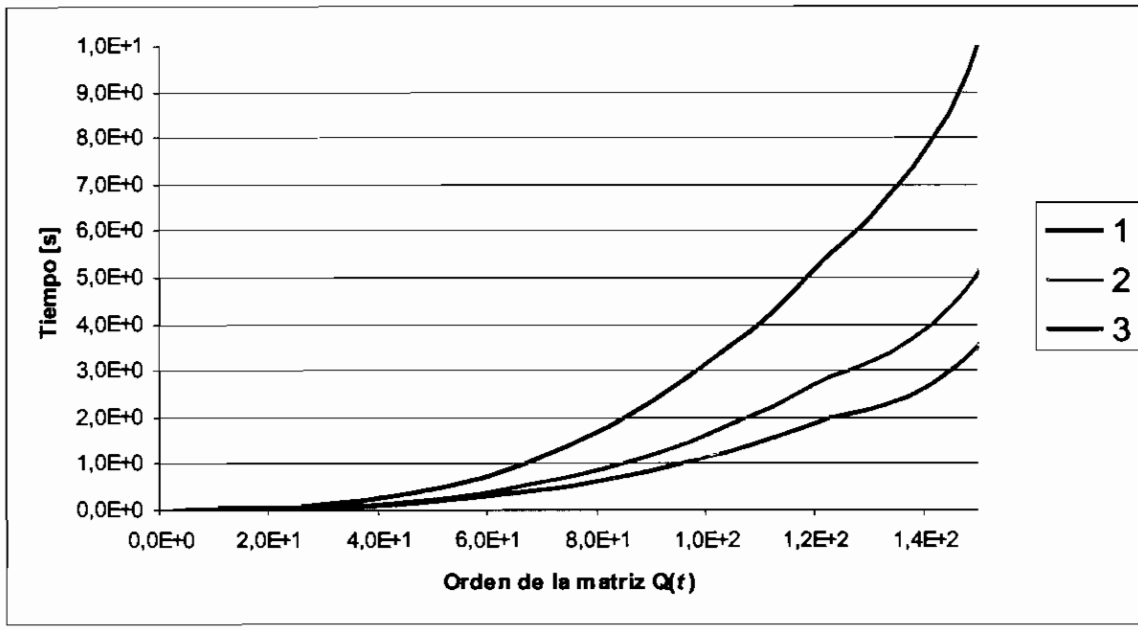


Figura 5-52. (Parte I) Comparación del tiempo requerido con 1, 2 y 3 procesadores para ejecutar la aplicación para resolver CMTC usando OSCAR

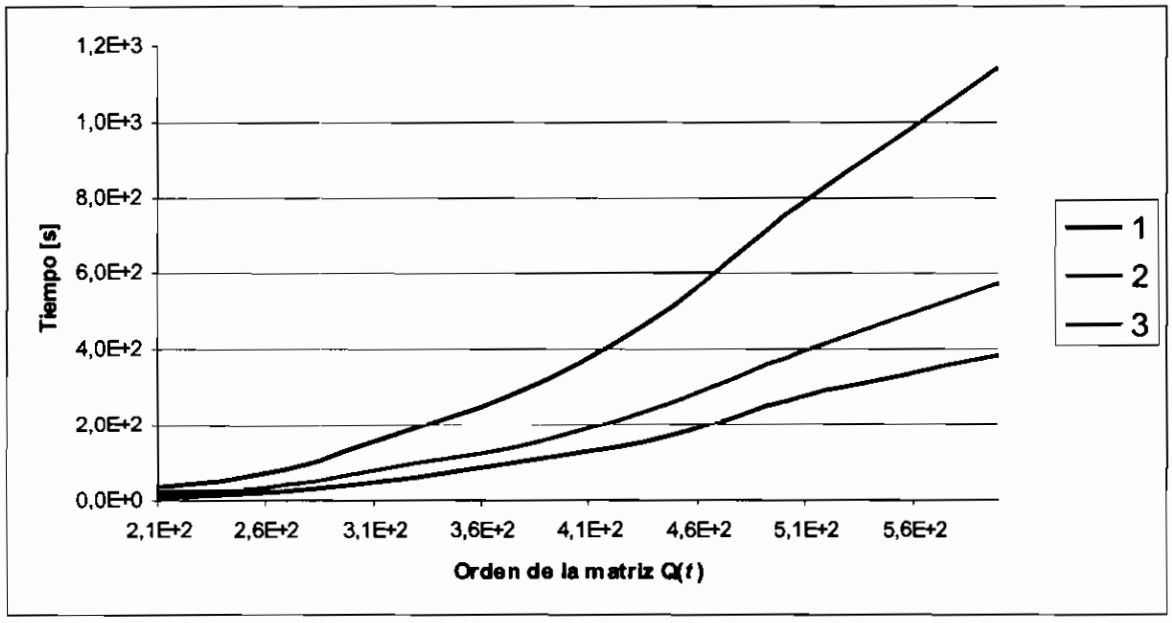


Figura 5-52. (Parte II) Comparación del tiempo requerido con 1, 2 y 3 procesadores para ejecutar la aplicación para resolver CMTC usando OSCAR

De forma similar que en CMTD, el rendimiento mejora en un valor cercano al 50% con dos procesadores y casi un 60% al utilizar tres procesadores en lugar de uno.

5.4.2. RESULTADOS OBTENIDOS CON *Rocks*

Para evaluar las Cadenas de Markov a Tiempo Discreto, utilizando *Rocks*, se obtuvo la distribución en el paso 25 y la distribución de régimen.

En la Tabla 5-11 se presentan los resultados obtenidos al calcular la distribución en el paso 25 y en la Figura 5-53 se realiza una comparación de dichos resultados con 1, 2 y 3 procesadores.

En la Tabla 5-12 se presentan los resultados obtenidos al calcular la distribución de régimen y en la Figura 5-54 se realiza una comparación de dichos resultados con 1, 2 y 3 procesadores.

En la Tabla 5-13 se presentan los tiempos de ejecución que tomó la resolución de la CMTD y en la Figura 5-55 se realiza una comparación de dichos resultados.

Para visualizar de mejor manera las gráficas del tiempo requerido para obtener la distribución de régimen, la distribución al tiempo o al paso y el tiempo de ejecución en función del orden de la matriz (**P** o **Q**(t)), se presentan dos figuras por cada tabla. La primera figura presenta los valores obtenidos al utilizar matrices de orden bajo (3 a 150); y la segunda figura muestra los resultados obtenidos para las matrices de mayor orden (210 a 600).

Orden de la matriz P	Procesos 1	2	3
3	2,00E-3	1,87E-4	2,15E-4
30	8,50E-3	7,72E-3	3,39E-3
60	6,13E-2	3,15E-2	2,20E-2
90	1,88E-1	9,72E-2	6,97E-2
120	4,37E-1	2,22E-1	1,52E-1
150	8,29E-1	4,20E-1	2,87E-1
210	2,87E-1	1,60E+0	1,06E+0
270	6,94E+0	2,35E+0	2,36E+0
330	1,32E+1	6,74E+0	4,52E+0
390	2,97E+1	1,49E+1	1,00E+1
450	4,62E+1	2,32E+1	1,55E+1
510	6,81E+1	3,42E+1	2,28E+1
600	9,85E+1	4,94E+1	3,31E+1

Tabla 5-11. Tiempo requerido en segundos para obtener la distribución en el paso 25 con 1, 2 y 3 procesadores usando *Rocks*

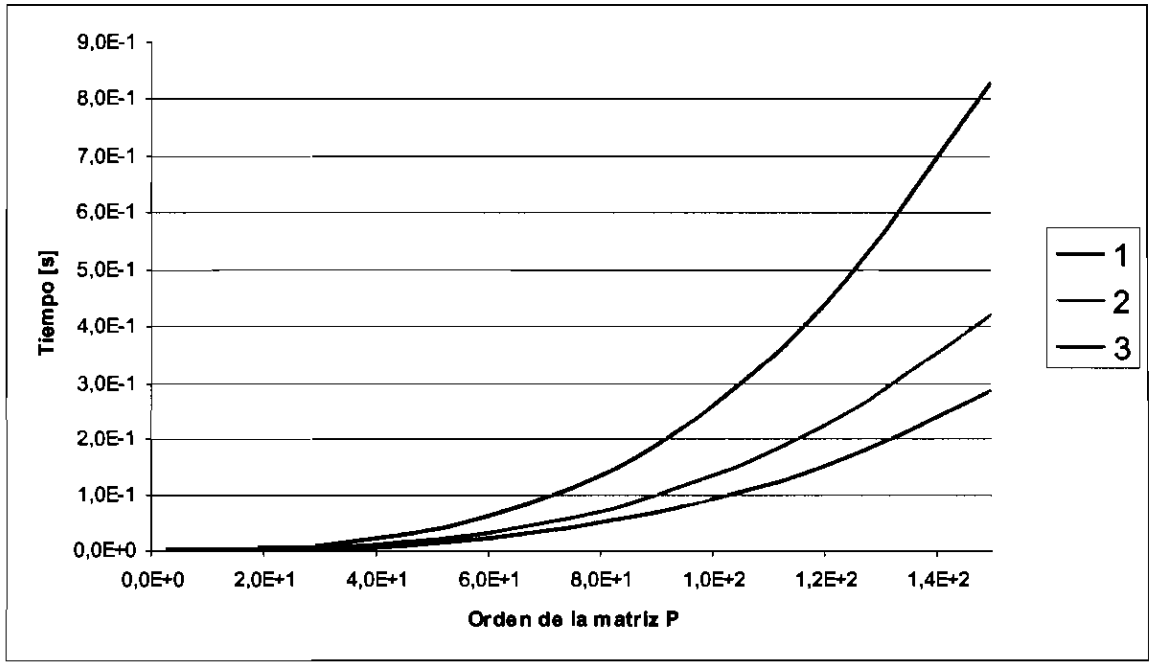


Figura 5-53. (Parte I) Comparación del tiempo requerido con 1, 2 y 3 procesadores para obtener la distribución en el paso 25 usando *Rocks*

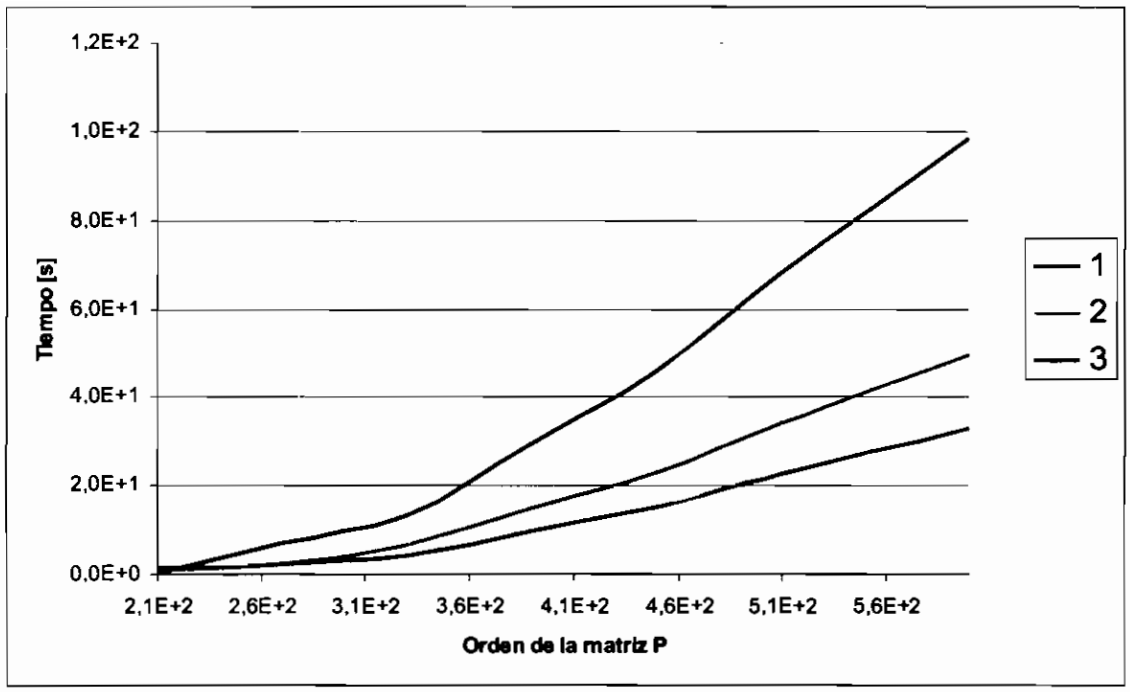


Figura 5-53. (Parte II) Comparación del tiempo requerido con 1, 2 y 3 procesadores para obtener la distribución en el paso 25 usando *Rocks*

Los resultados obtenidos son muy similares a los obtenidos utilizando OSCAR. El tiempo requerido para obtener la distribución en el paso 25 en un procesador, se reduce en 1/2 al usar dos procesadores y en 1/3 si se usan los tres.

		Procesos		
		1	2	3
Orden de la matriz P	3	7,00E-6	3,95E-2	7,28E-4
	30	3,66E-4	4,10E-2	4,15E-2
	60	2,54E-3	3,52E-3	4,59E-3
	90	8,10E-3	7,74E-3	8,82E-3
	120	1,86E-2	1,48E-2	1,43E-2
	150	3,58E-2	2,55E-2	2,29E-2
	210	9,69E-2	6,22E-2	8,60E-2
	270	2,03E-1	1,22E-1	9,76E-2
	330	3,84E-1	2,14E-1	1,60E-1
	390	6,69E-1	3,52E-1	2,64E-1
	450	1,04E+0	5,43E-1	3,85E-1
	510	1,50E+0	8,17E-1	5,54E-1
	600	2,44E+0	1,35E+0	9,18E-1

Tabla 5-12. Tiempo requerido en segundos para obtener la distribución de régimen (CMTD) con 1, 2 y 3 procesadores usando *Rocks*

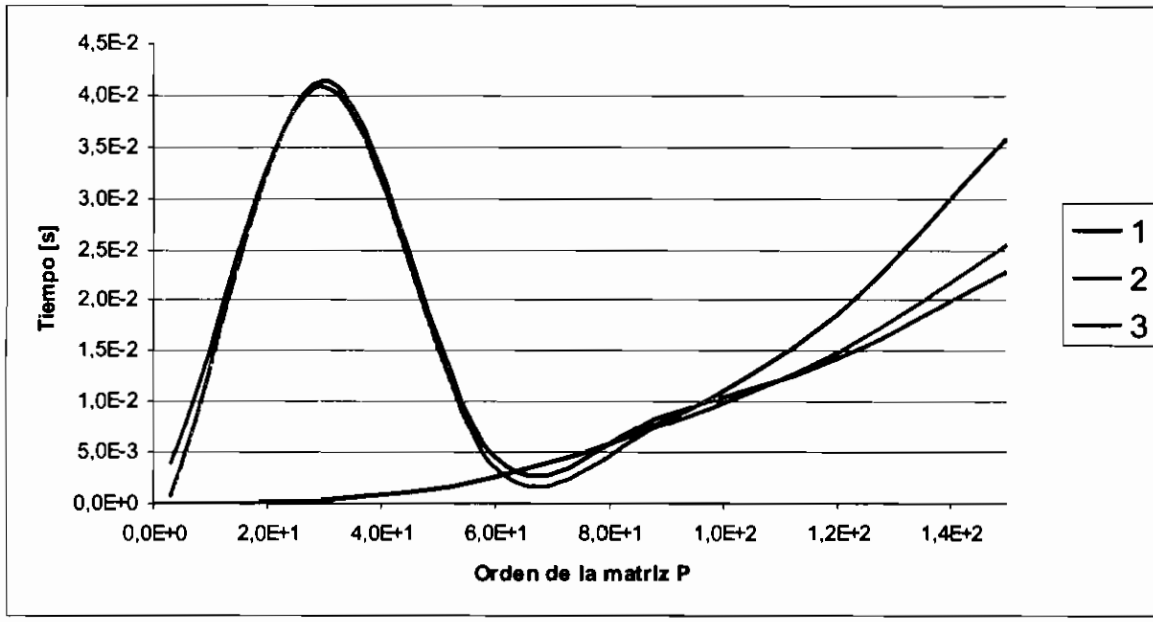


Figura 5-54. (Parte I) Comparación del tiempo requerido con 1, 2 y 3 procesadores para obtener la distribución de régimen (CMTD) usando *Rocks*

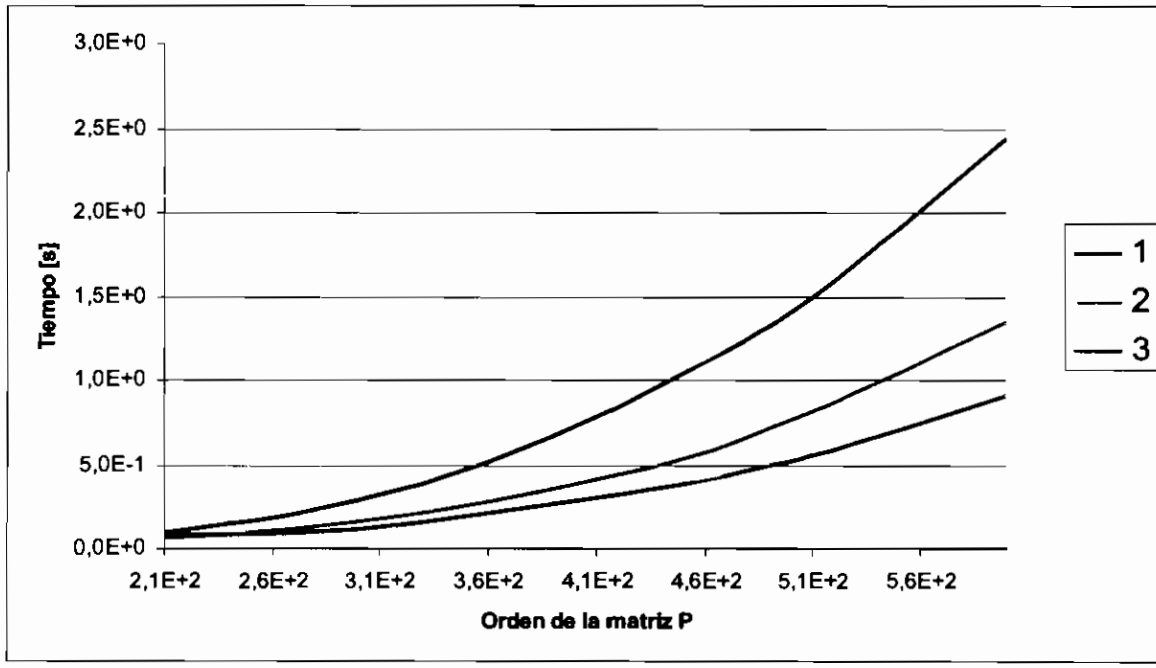


Figura 5-54. (Parte II) Comparación del tiempo requerido con 1, 2 y 3 procesadores para obtener la distribución de régimen (CMTD) usando *Rocks*

De igual manera que en OSCAR, de la Figura 5-54 (parte II) se puede concluir que el rendimiento mejora en aproximadamente 50% si se utilizan dos procesadores y en casi 60% si se utilizan los tres procesadores en lugar de uno. En la Figura 5-54 (parte I) se obtuvo un resultado distinto al obtenido en la Figura

5-54 (parte II), y esto se debe a que en las matrices de menor orden (3 a 150), la cantidad de procesamiento es baja comparada con las operaciones requeridas para enviar y recibir los datos desde cada proceso.

Orden de la matriz P	Procesos 1	2	3
3	0,00E+0	0,00E+0	0,00E+0
30	1,00E-2	0,00E+0	0,00E+0
60	6,00E-2	3,00E-2	2,00E-2
90	2,00E-1	1,00E-1	8,00E-2
120	4,60E-1	2,40E-1	1,70E-1
150	8,80E-1	4,60E-1	3,20E-1
210	3,30E+0	1,69E+0	1,13E+0
270	7,19E+0	3,66E+0	2,48E+0
330	1,36E+1	6,96E+0	4,69E+0
390	3,05E+1	1,53E+1	1,03E+1
450	4,73E+1	2,38E+1	1,59E+1
510	6,98E+1	3,52E+1	2,35E+1
600	1,01E+2	5,09E+1	3,41E+1

Tabla 5-13. Tiempo en segundos para ejecutar la aplicación que permite resolver CMTD con 1, 2 y 3 procesadores usando *Rocks*

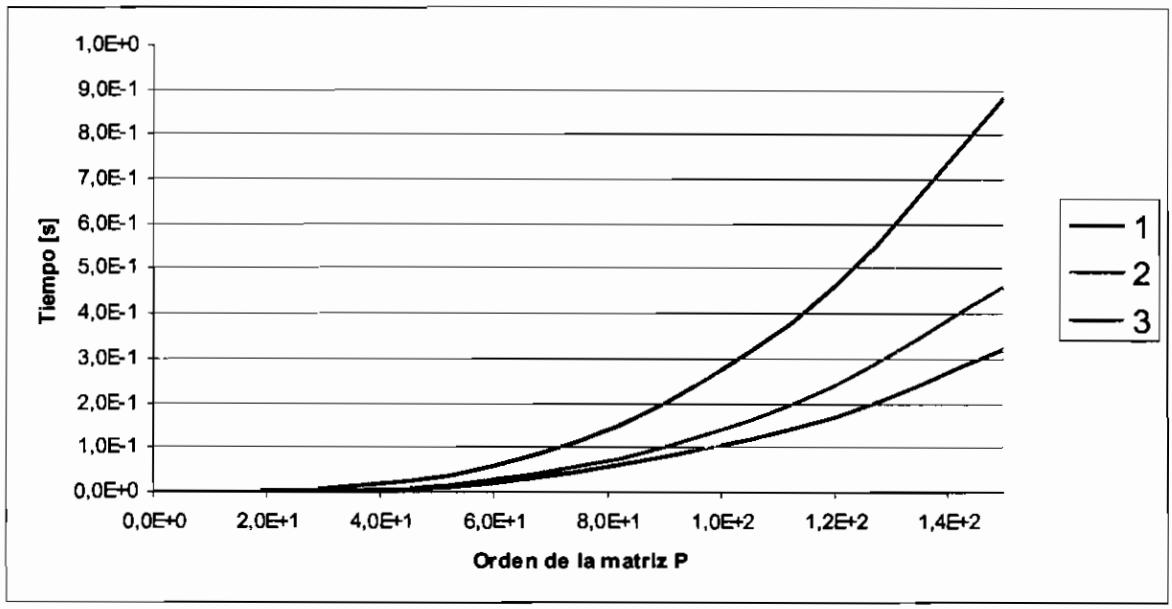


Figura 5-55. (Parte I) Comparación del tiempo requerido con 1, 2 y 3 procesadores para ejecutar la aplicación para resolver CMTD usando *Rocks*

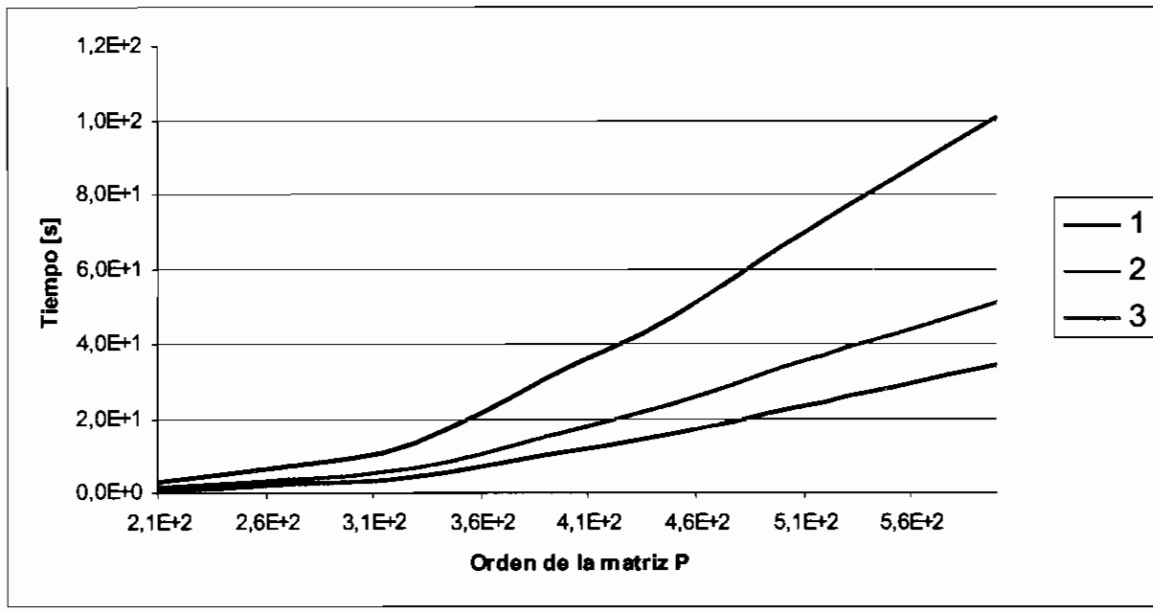


Figura 5-55. (Parte II) Comparación del tiempo requerido con 1, 2 y 3 procesadores para ejecutar la aplicación para resolver CMTD usando *Rocks*

De la Figura 5-55 se puede mencionar que el tiempo de ejecución de la aplicación que permite evaluar CMTDs, se reduce aproximadamente al 50% si se utilizan dos procesadores y en un valor cercano al 60% al utilizar tres procesadores si se compara con el tiempo de ejecución de la aplicación en un solo procesador.

Para Cadenas de Markov a Tiempo Continuo, utilizando *Rocks*, se obtuvo la distribución en el tiempo 0,25 y la distribución de régimen.

En la Tabla 5-14 se presentan los resultados obtenidos al calcular la distribución en el paso 25 y en la Figura 5-56 se realiza una comparación de dichos resultados con 1, 2 y 3 procesadores.

En la Tabla 5-15 se presentan los resultados obtenidos al calcular la distribución de régimen y en la Figura 5-57 se realiza una comparación de dichos resultados con 1, 2 y 3 procesadores.

En la Tabla 5-16 se presentan los tiempos de ejecución que tomó la resolución de la CMTC y en la Figura 5-58 se realiza una comparación de dichos resultados.

Al igual que en la resolución de CMTD, descrita en los párrafos anteriores, también se presentan dos figuras por cada tabla, para presentar de mejor manera los resultados obtenidos.

Orden de la matriz Q(t)	Procesos 1	2	3
3	2,34E-4	2,74E-4	3,12E-4
30	9,89E-2	5,01E-2	3,40E-2
60	6,98E-1	3,58E-1	2,96E-1
90	2,18E+0	1,10E+0	7,34E-1
120	4,97E+0	2,52E+0	1,67E+0
150	9,59E+0	4,83E+0	3,22E+0
210	3,67E+1	1,83E+1	1,26E+1
270	7,96E+1	3,95E+1	2,64E+1
330	1,54E+2	7,69E+1	5,52E+1
390	3,47E+2	1,77E+2	1,18E+2
450	5,13E+2	2,65E+2	1,76E+2
510	7,82E+2	3,93E+2	2,61E+2
600	1,14E+3	5,28E+2	3,79E+2

Tabla 5-14. Tiempo en segundos requerido para obtener la distribución en el tiempo 0,25 con 1, 2 y 3 procesadores usando *Rocks*

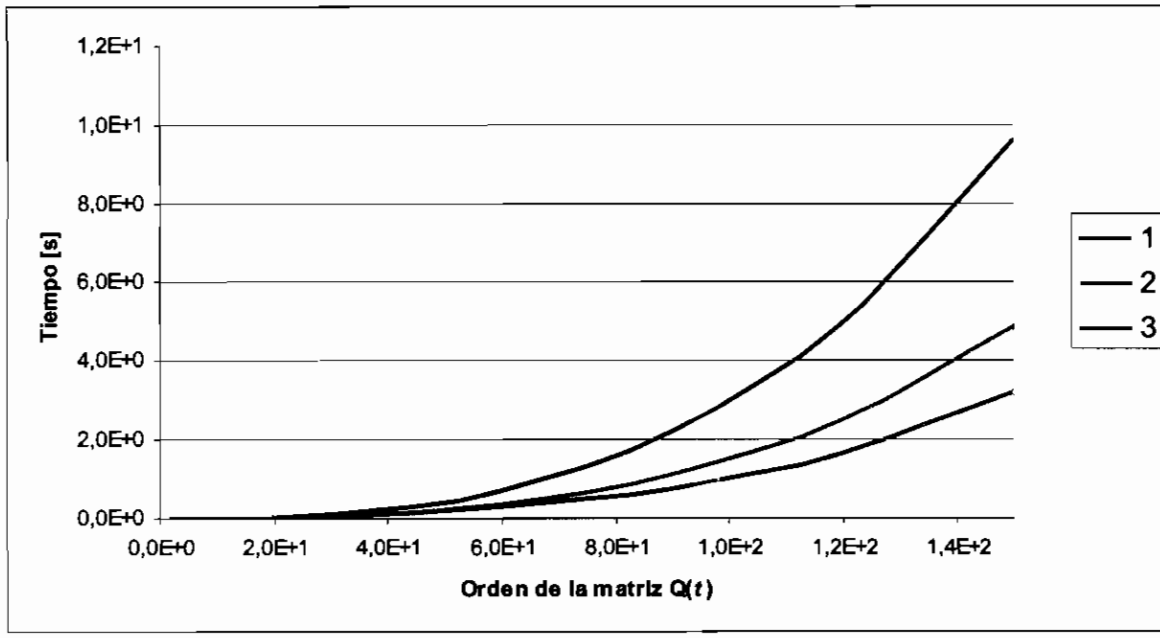


Figura 5-56. (Parte I) Comparación del tiempo requerido con 1, 2 y 3 procesadores para obtener la distribución en el tiempo 0,25 usando *Rocks*

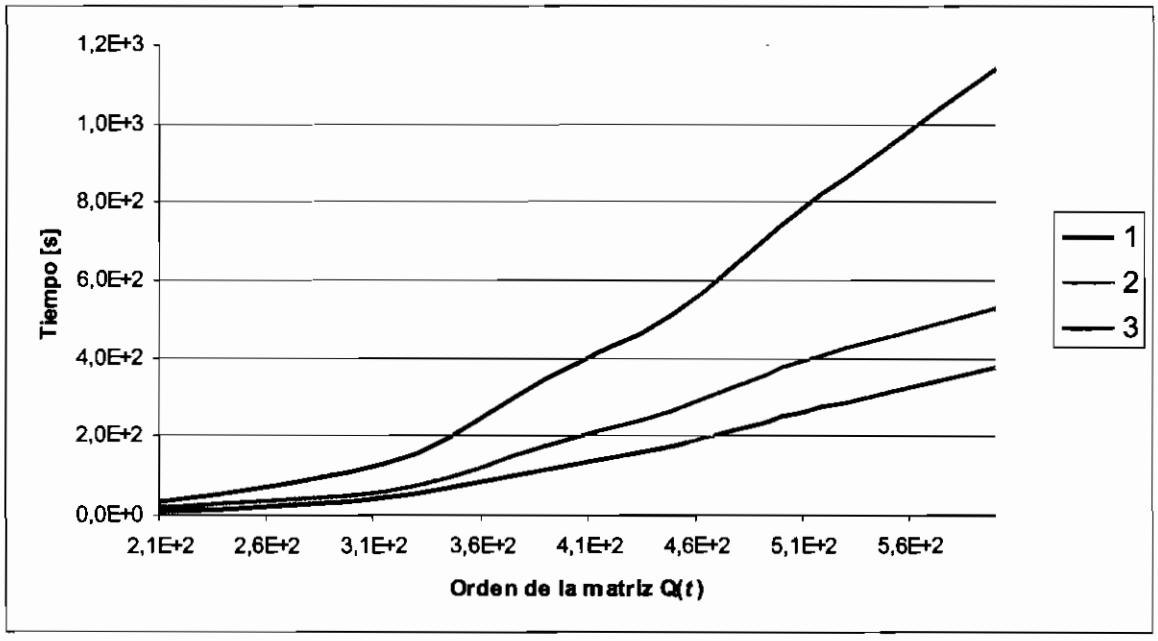


Figura 5-56. (Parte II) Comparación del tiempo requerido con 1, 2 y 3 procesadores para obtener la distribución en el tiempo 0,25 usando *Rocks*

El tiempo necesario para obtener la distribución en el tiempo 0,25, utilizando un sólo procesador aproximadamente es dos veces el tiempo requerido si se emplean dos procesadores y casi tres veces si se emplean tres procesadores.

		Procesos		
		1	2	3
Orden de la matriz Q(t)	3	7,00E-6	3,98E-2	7,07E-4
	30	3,50E-4	4,11E-2	4,20E-2
	60	2,46E-3	3,52E-3	4,92E-3
	90	7,94E-3	7,74E-3	8,66E-3
	120	1,84E-2	1,47E-2	1,41E-2
	150	3,55E-2	2,51E-2	2,28E-2
	210	9,59E-2	6,19E-2	5,27E-2
	270	2,08E-1	1,22E-1	9,70E-2
	330	3,89E-1	2,14E-1	1,62E-1
	390	6,81E-1	3,57E-1	2,75E-1
	450	1,04E+0	5,52E-1	3,94E-1
	510	1,53E+0	8,22E-1	5,54E-1
	600	2,44E+0	1,37E+0	9,16E-1

Tabla 5-15. Tiempo en segundos requerido para obtener la distribución de régimen (CMTC) con 1, 2 y 3 procesadores usando *Rocks*

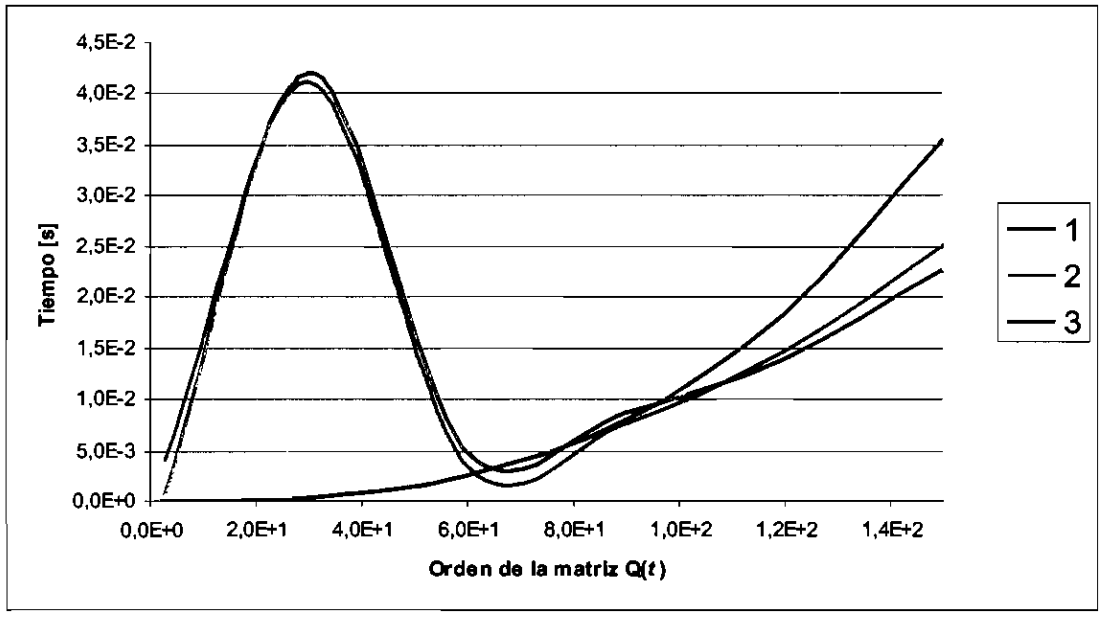


Figura 5-57. (Parte I) Comparación del tiempo requerido con 1, 2 y 3 procesadores para obtener la distribución de régimen (CMTC) usando *Rocks*

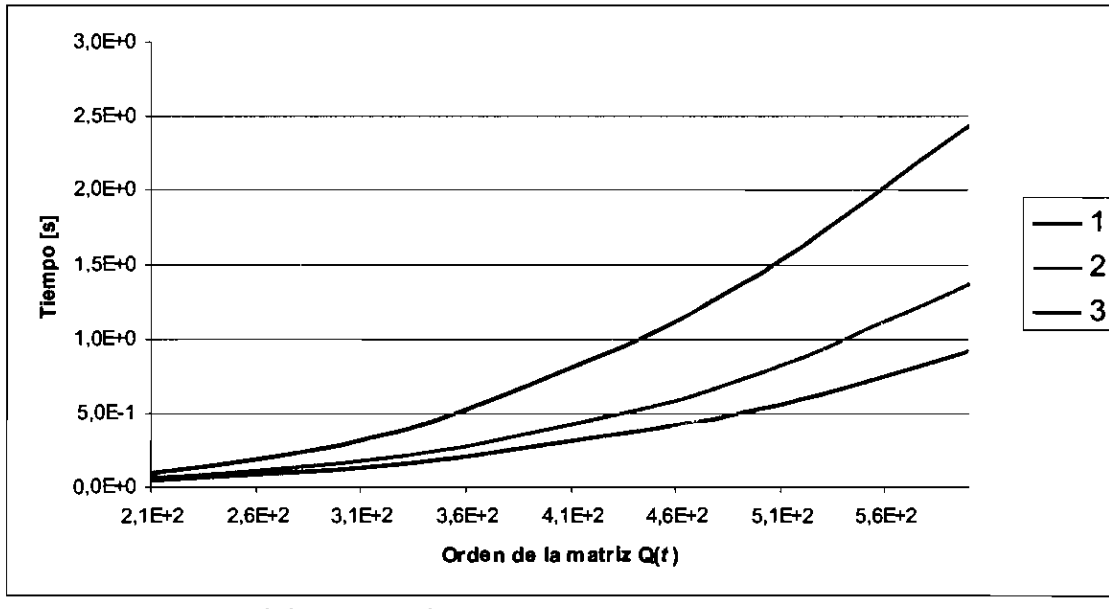


Figura 5-57. (Parte II) Comparación del tiempo requerido con 1, 2 y 3 procesadores para obtener la distribución de régimen (CMTC) usando *Rocks*

De igual manera que en CMTD, las partes I y II de la Figura 5-57 presentan resultados diferentes. La parte I muestra que no existen mejoras en el rendimiento si se utilizan matrices de tamaño bajo (3 a 150), debido al procesamiento requerido para su solución y al tiempo empleado en comunicaciones entre

procesos; mientras que la parte II muestra mejoras en el rendimiento al utilizar matrices de tamaño superior (210 a 600).

Orden de la matriz Q(t)	Procesos 1	2	3
3	0,00E+0	0,00E+0	0,00E+0
30	1,10E-1	6,00E-2	4,00E-2
60	7,40E-1	3,90E-1	2,80E-1
90	2,29E+0	1,19E+0	8,20E-1
120	5,15E+0	2,65E+0	1,84E+0
150	9,88E+0	5,07E+0	3,46E+0
210	3,73E+1	1,89E+1	1,28E+1
270	8,06E+1	4,67E+1	2,73E+1
330	1,56E+2	7,83E+1	5,65E+1
390	3,49E+2	1,75E+2	1,18E+2
450	5,16E+2	2,62E+2	1,76E+2
510	7,85E+2	3,97E+2	2,65E+2
600	1,14E+3	5,87E+2	3,82E+2

Tabla 5-16. Tiempo en segundos para ejecutar la aplicación que permite resolver CMTC con 1, 2 y 3 procesadores usando *Rocks*

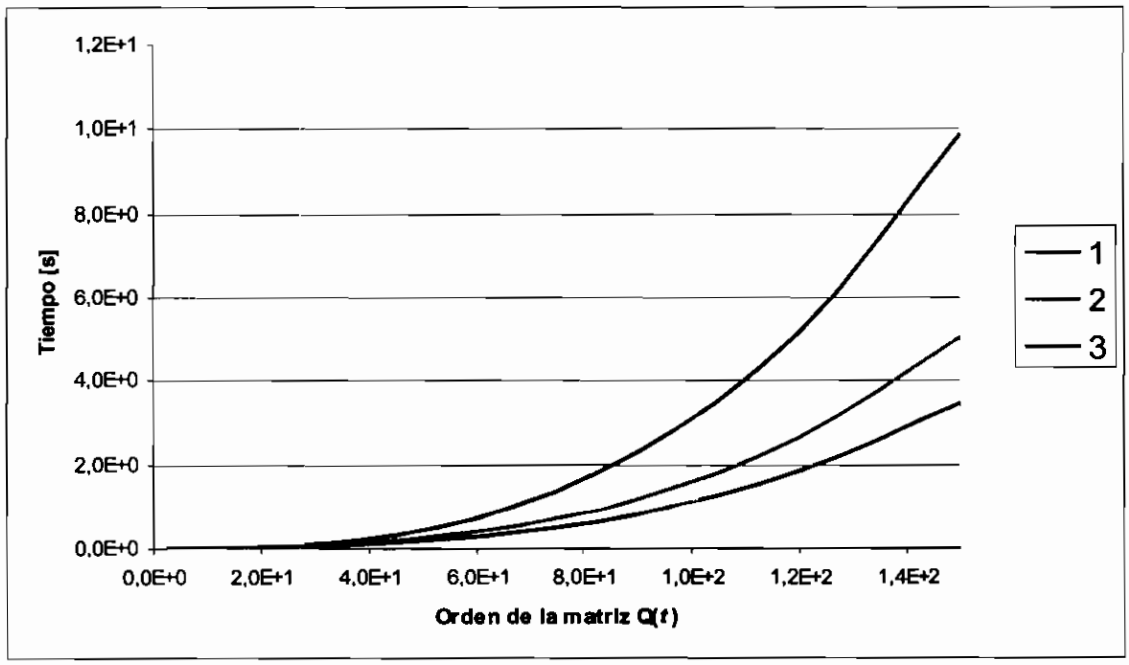


Figura 5-58. (Parte I) Comparación del tiempo requerido con 1, 2 y 3 procesadores para ejecutar la aplicación para resolver CMTC usando *Rocks*

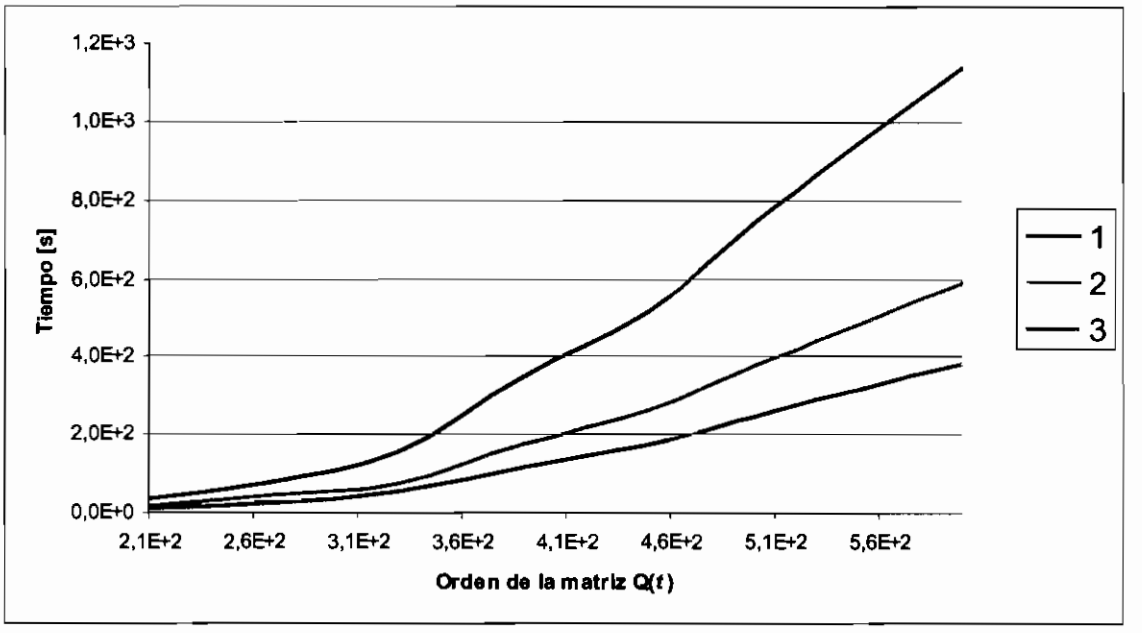


Figura 5-58. (Parte II) Comparación del tiempo requerido con 1, 2 y 3 procesadores para ejecutar la aplicación para resolver CMTC usando *Rocks*

De forma similar que en CMTD, el rendimiento mejora en un valor cercano al 50% con dos procesadores y casi un 60% al utilizar tres procesadores en lugar de uno.

5.4.3. ANÁLISIS DE RESULTADOS

Se puede ver que en promedio, usando 2 procesadores, el tiempo que toma obtener la distribución en el paso 25 de Cadenas de Markov a Tiempo Discreto, tanto en OSCAR como en *Rocks,* es un valor cercano a la mitad del tiempo que toma el hacerlo con un sólo procesador; y usando 3 procesadores, el tiempo que toma se reduce a un valor cercano al tercio del valor que toma sobre uno sólo.

En promedio algo similar a lo descrito en la obtención de la distribución en el paso 25, ocurre para la distribución en el tiempo 0,25 de Cadenas de Markov a Tiempo Continuo.

Se puede apreciar también, que en el caso de la distribución de régimen, tanto en CMTD como en CMTC, para OSCAR y *Rocks*, se tienen valores que no muestran una mejora al compararlos con respecto a los valores obtenidos en un procesador, para matrices de tamaño pequeño (de orden 3 a 150), debido a que

la cantidad de operaciones es baja comparada con el tiempo requerido para el envío de datos entre procesos. Pero se puede apreciar una mejora en las matrices de mayor tamaño (orden superior a los 150), usando 2 procesadores el tiempo que se tarda en obtener la solución del sistema de ecuaciones se reduce en un 50% aproximadamente, y utilizando 3 procesadores el tiempo se reduce en promedio en un 60%.

Finalmente, se puede mencionar que el tiempo de ejecución total se comporta de forma similar a la descrita para la distribución en el paso 25 o en el tiempo 0,25. Esto se debe principalmente a que el tiempo que toma el obtener la distribución de régimen es bajo comparado con el tiempo que toma el obtener la distribución en el paso o en el tiempo y a la cantidad de operaciones involucradas en el cálculo de ambas. Se puede apreciar también, que el tiempo en leer la matriz **P** o **Q**, y almacenar los resultados, es decir las operaciones no paralelizadas, consumen poco tiempo comparado con las operaciones realizadas en paralelo.

5.4.4. EXTENSIÓN DEL ANÁLISIS DE RESULTADOS MEDIANTE EL USO DE MPE – *MULTI-PROCESSING ENVIRONMENT*

MPE provee un interfaz para recopilar información de aplicaciones desarrolladas con MPI mediante el uso de librerías. MPE se utiliza como una opción durante la compilación de un programa escrito con MPI y permite presentar información sobre las llamadas realizadas a la librería MPI.

MPE provee características adicionales para creación de archivos `log`, una librería para gráficos, herramientas para visualización entre otras.

Los archivos fuente se compilan utilizando la opción **–mpilog,** la cual permite integrar MPE a la aplicación, para luego de ejecutar la aplicación con el comando **mpirun** generar un archivo `log`. Los archivos `log` se leen mediante herramientas

creadas con un lenguaje de *scripts* llamado Tcl/Tk tales como: Upshot[20] y Nupshot, u otras escritas en Java tales como: Jumpshot-2, Jumpshot-3 y Jumpshot-4[21]. Estas herramientas permiten desplegar de forma gráfica Diagramas de **Gantt** de la información de las llamadas realizadas a MPI por la aplicación. Un Diagrama de **Gantt** es una línea de tiempo, una por cada proceso. La línea de tiempo correspondiente a un proceso muestra su estado. Al enlazar el código fuente con la librería de MPE, se obtiene información del tiempo consumido por el programa en cada función de MPI.

En OSCAR, MPICH no incluye las herramientas de visualización, debido a que no son parte de la implementación de MPI que OSCAR instala. *Rocks* instala la herramienta Jumpshot-4; por este motivo se utilizó *Rocks* para visualizar los archivos generados al ejecutar la aplicación sobre *Rocks* y OSCAR. Sin embargo, si se requiere se puede instalar Jumpshot-4 sobre OSCAR.

El comando utilizado para compilar la aplicación que integra a MPE y que resuelve Cadenas de Markov a Tiempo Discreto es:

```
$ mpiCC -mpilog -o tesistd main.cpp Matrix.cpp Matrix_Handler.cpp MarkovTD.cpp
```

El comando utilizado para compilar la aplicación que integra a MPE y que resuelve Cadenas de Markov a Tiempo Continuo es:

```
$ mpiCC -mpilog -o tesistc main.cpp Matrix.cpp Matrix_Handler.cpp MarkovTC.cpp
```

En la Figura 5-59 se muestra la leyenda que indica el tipo de llamada a MPI la cual se identifica por un color específico. En las siguientes figuras se utiliza la simbología para *message*, BARRIER, BCAST, RECV y SEND; y no se utilizan

[20] Información acerca de Upshot, Nupshot, Jumpshot-2 y Jumpshot-3 se puede encontrar en el libro "*User's Guide for MPE: Extensions for MPI Programs*" escrito por Anthony Chan y William Gropp, editado por el Laboratorio Nacional Argonne de Estados Unidos.

[21] Información acerca de Jumpshot-4 se puede encontrar en el libro "*User's Guide for Jumpshot-4*" escrito por por Anthony Chan y William Gropp, editado por el Laboratorio Nacional Argonne de Estados Unidos.

Preview_Arrow y *Preview_State*. Si se desea información sobre *Preview_Arrow* y *Preview_State* se recomienda revisar el sitio Web:

http://www-unix.mcs.anl.gov/perfvis/software/viewers/jumpshot-4/usersguide.html

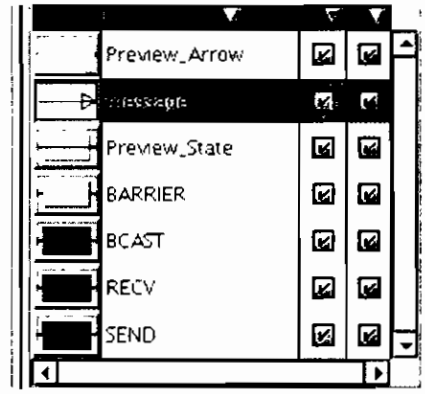


Figura 5-59. Leyenda de Jumpshot-4

BARRIER muestra el tiempo que toma una llamada a **MPI_Barrier**, BCAST muestra el tiempo que toma una llamada a **MPI_Bcast**, RECV presenta el tiempo que tarda realizar una llamada a **MPI_Recv** y SEND indica el tiempo de la llamada **MPI_Send**.

El tiempo que toma cada una de las llamadas a MPI, incluye el tiempo en establecer una conexión segura para transmitir el mensaje, el tiempo de encripción del mensaje, el tiempo de transmisión del mensaje, y el tiempo de desencripción del mensaje.

En estas figuras generadas usando MPE, se puede apreciar que se generan barras solamente para las llamadas a MPI, las otras partes del programa se indican con una línea horizontal.

La Figura 5-59 sirve de leyenda para las figuras que se presentan a continuación.

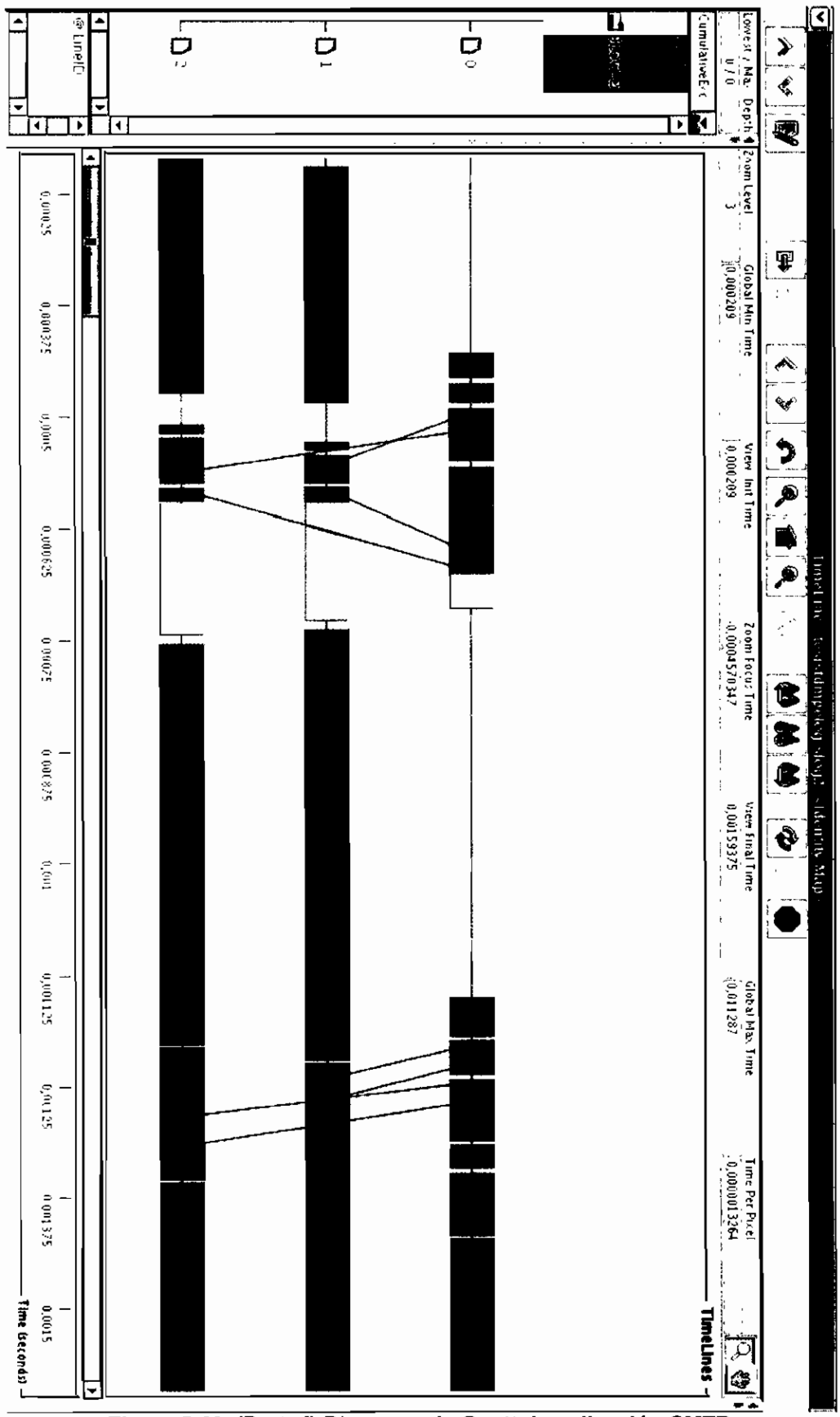


Figura 5-60. (Parte I) Diagrama de Gantt de aplicación CMTD

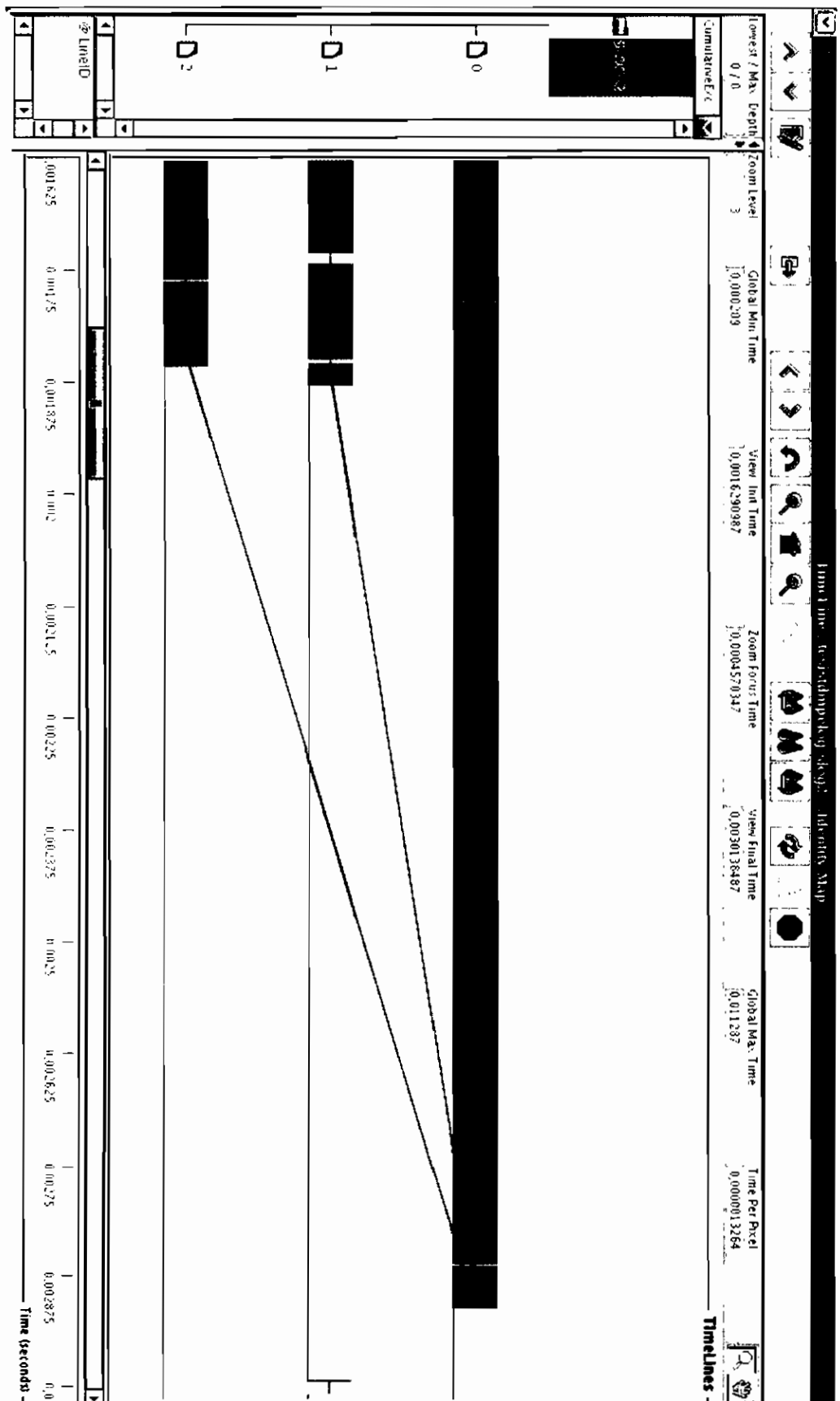


Figura 5-60. (Parte II) Diagrama de Gantt de aplicación CMTD

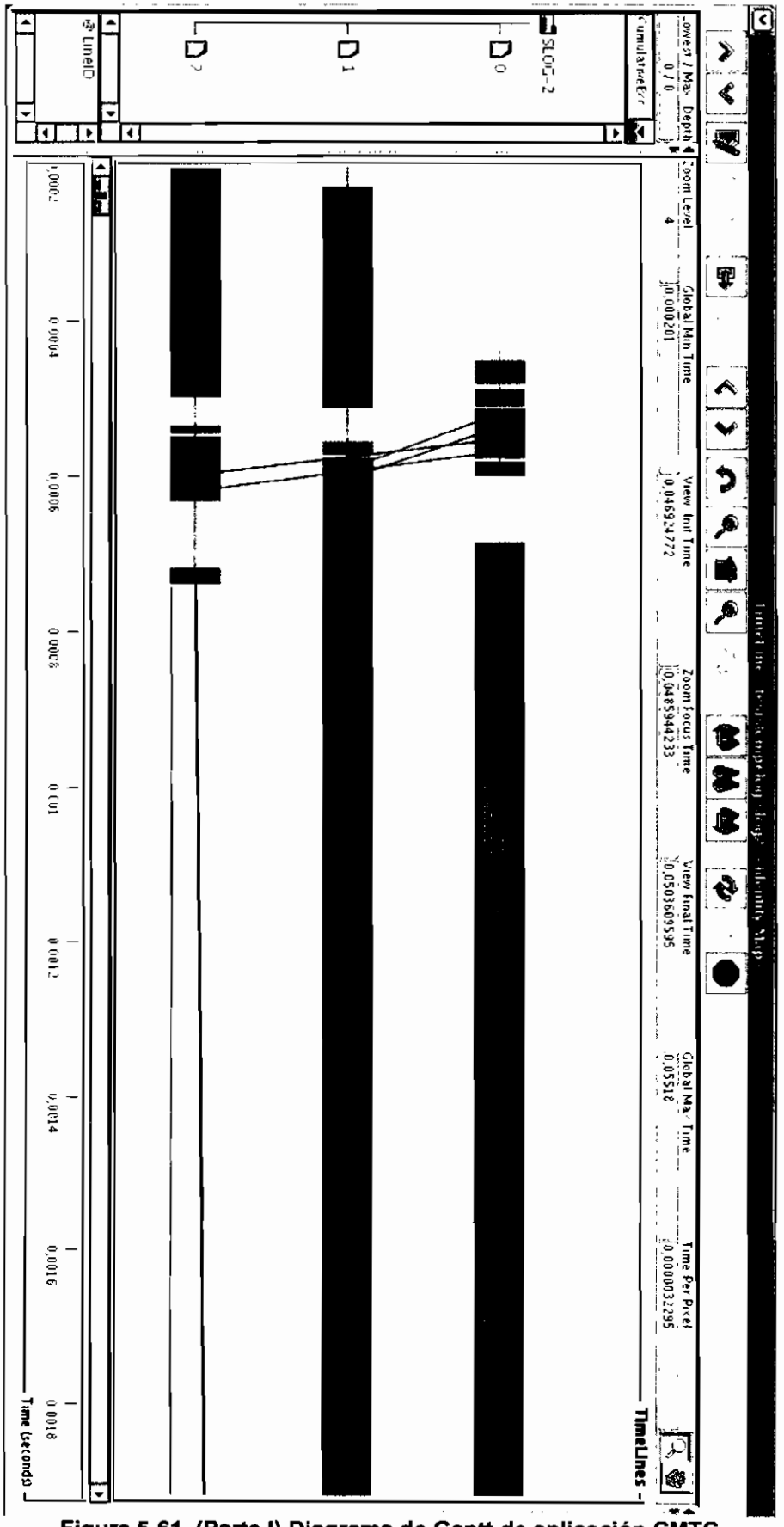


Figura 5-61. (Parte I) Diagrama de Gantt de aplicación CMTC

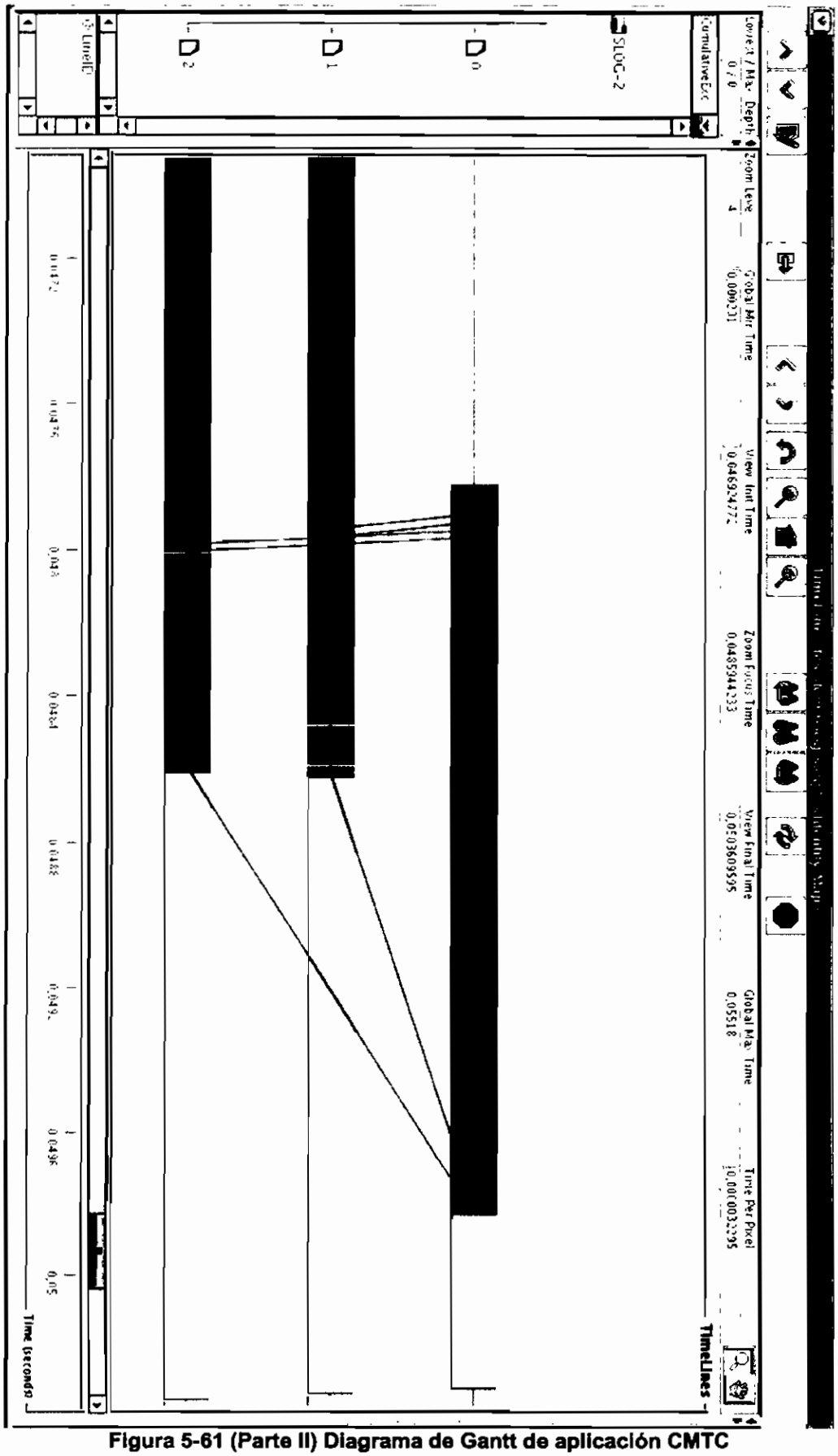


Figura 5-61 (Parte II) Diagrama de Gantt de aplicación CMTC

En la Figura 5-60 se puede apreciar que los procesos 1 y 2 consumen una gran cantidad de tiempo en la primera llamada a **MPI_Bcast**, debido a que deben esperar a que el proceso maestro lea la matriz **P** desde el archivo de texto, en esta llamada se envía la dimensión de la matriz y el paso en el que se evalúa la cadena. Luego se realiza una llamada a **MPI_Send** en el proceso 0 para enviar la porción correspondiente a cada proceso, y el resto de procesos realizan la llamada a **MPI_Recv** para recibir su porción de matriz. Además, se envía toda la matriz mediante **MPI_Bcast** para obtener la distribución al paso; se puede apreciar que esta llamada es de una duración corta. Luego el proceso 0 realiza una llamada a **MPI_Recv** para recibir los datos desde los otros procesos, se puede apreciar que la primera llamada a **MPI_Recv** tiene mayor duración debido a que el proceso 0 se encuentra realizando sus operaciones correspondientes. Finalmente se realiza una llamada a **MPI_Barrier**. Los procesos 1 y 2 consumen más tiempo en la llamada a **MPI_Barrier** debido a que el proceso 0 tiene que recibir los resultados antes de realizar esta llamada. Luego el proceso 0 imprime los resultados y los imprime en un archivo, mientras los otros procesos esperan por la finalización de **MPI_Bcast**. El proceso 0 envía las porciones correspondientes de las matrices que permiten resolver el sistema de ecuaciones lineales primero al proceso 1 y luego al proceso 2. Se puede apreciar que el proceso 1 no debe esperar mucho para obtener su porción de las matrices, sin embargo el proceso 2 debe esperar a que el proceso envíe los datos al proceso 1 para empezar a recibir su porción. Luego se realizan una serie de llamadas a **MPI_Bcast** para enviar el pívot y realizar las operaciones correspondientes. La duración a la llamada **MPI_Bcast** difiere entre los procesos de acuerdo a la cantidad de trabajo que está realizando.

En la Figura 5-60 parte II, se observan otras llamadas a **MPI_Bcast**. Luego el proceso 0 realiza la llamada a **MPI_Recv** para recibir los resultados emitidos por los procesos 1 y 2 los cuales realizan una llamada a **MPI_Send**. El proceso 0 recibe primero los resultados del proceso 1 y luego de finalizada esta operación recibe los resultados del proceso 2. El proceso 0, 1 y 2 realizan una llamada a **MPI_Barrier**, sin embargo el proceso 1 y 2 esperan a que el proceso 0 finalize las

operaciones de escritura de los resultados en un archivo antes de finalizar la llamada a **MPI_Barrier**.

Un proceso similar ocurre en la Figura 5-61 donde se pueden apreciar llamadas a **MPI_Bcast**, **MPI_Send**, **MPI_Recv** y **MPI_Barrier** y situaciones similares descritas en los párrafos anteriores.

De la Figura 5-60 y de la Figura 5-61 se puede concluir, que el tiempo que toma una llamada a **MPI_Send** incluye los tiempos de encripción, transmisión y desencripción del mensaje, y cuya duración es de un valor relativamente pequeño; de igual forma que con una llamada a **MPI_Send**, una llamada a **MPI_Recv** incluye los tiempos de encripción, transmisión y desencripción del mensaje y su duración también es relativamente pequeña; que los procesos 1 y 2 consumen una gran cantidad de tiempo en algunas de las llamadas a **MPI_Bcast**, debido a que deben esperar a que el proceso 0 realice operaciones seriales, como lectura de la matriz **P** o $\mathbf{Q}(t)$ desde el archivo de texto, impresión en consola y almacenamiento de resultados en archivos; además las llamadas a **MPI_Barrier**, tienen una duración mayor en los procesos 1 y 2 debido a que deben esperar por el proceso 0.

CAPÍTULO 6. CONCLUSIONES Y RECOMENDACIONES

- La instalación de un *cluster* involucra la configuración de una gran cantidad de servicios, el problema que se tendría al realizarlo de forma manual sería la gran cantidad de tiempo que se debería invertir para configurar cada uno de los equipos, además, se debe invertir tiempo adicional en la búsqueda y resolución de problemas que se generan por una mala manipulación de los archivos de configuración. Por este motivo, la ventaja que se obtiene al utilizar herramientas, como OSCAR y NPACI *Rocks*, que permiten instalar y configurar un *cluster* es evidente; el tiempo requerido para su instalación se reduce drásticamente y no se requiere de tiempo adicional en tratar de buscar y solucionar problemas que pueden generarse por un error en la configuración.

- Adicionalmente, estas herramientas de instalación automática, no sólo reducen el trabajo y el tiempo que un administrador invierte al momento de instalar y configurar el sistema *cluster*, sino que también proveen métodos eficientes en caso de errores posteriores. Las dos herramientas toman muy poco tiempo en reinstalar los nodos cliente, y no se pierde la información de los usuarios debido a que ésta se encuentra almacenada en el nodo servidor. Sin embargo, la desventaja es clara, el servidor se convierte en un único punto de falla. Por este motivo, se deben plantear y buscar alternativas que permitan reducir los problemas que se podría tener frente a la falla del nodo servidor; por ejemplo, se podrían usar RAIDs para evitar fallas en los discos duros, o se podría utilizar alternativas que permitan tener otros servidores redundantes.

- Cabe recalcar que se pueden instalar *clusters* en tiempos relativamente cortos y con poco esfuerzo utilizando las herramientas de instalación automática; sin embargo, el utilizar software libre conlleva ciertas particularidades respecto a su instalación sobre algunos componentes de hardware, es posible que no se disponga del soporte adecuado para dichos componentes, motivo por el cual se debe disponer de conocimientos adicionales para poder enfrentarse a estos problemas y del tiempo necesario para proveer dicho soporte. Adicionalmente, la instalación conlleva realizar pruebas para comprobar el correcto funcionamiento del componente de hardware.

- Como parte del trabajo desarrollado se realizó el diseño, la configuración y la instalación de un sistema *cluster* de PCs utilizando el sistema operativo Linux, *toolkits* para la instalación automática de *clusters* de distribución libre: OSCAR y NPACI *Rocks*. Para el desarrollo de aplicaciones se utilizó la implementación del estándar MPI Versión 1 y 2 conocida como MPICH; y se emplearon las herramientas para la administración, balanceo de carga y monitoreo de los nodos del *cluster* C3, Ganglia.

- La arquitectura actual del *cluster* construido presenta un punto de falla crítico, debido a que sólo se dispone de un equipo de interconectividad, lo que significa que si por algún motivo este equipo llegara a fallar el *cluster* quedaría inutilizable, por lo que se recomienda buscar alternativas que permitan proveer redundancia a nivel de comunicaciones, por ejemplo utilizando tecnologías como *channel bonding* que permiten proveer balanceo de carga mediante la distribución de los datos en múltiples interfaces. Por otro lado, debido a las modificaciones realizadas en los archivos de configuración para conseguir instalar los dos sistemas operativos, la falla de un nodo no puede solucionarse simplemente con la reinstalación del mismo, se debe seguir el mismo proceso descrito en la instalación para no perder los dos sistemas en el momento de la reinstalación.

- Para este proyecto se presentó la información concerniente con OSCAR y NPACI *Rocks*, así como información de algunas aplicaciones que estas herramientas instalan de forma automática como Ganglia, PBS, HPFS, NFS, C3, Maui, entre otras. Además, fue necesario buscar información relacionada con el sistema operativo Linux para poder resolver muchos de los problemas que se presentaron durante la instalación del *cluster*, se revisaron aspectos como compilación del *kernel*, gestor de arranque, administración de demonios, compilación e instalación de programas, entre otros.

- Durante la fase de instalación se encontraron algunos problemas: el sistema operativo Linux Red Hat 9.0 y el paquete **System Installer Suite** del *toolkit* OSCAR Versión 3.0 y 4.X no tienen el soporte para algunos dispositivos físicos como discos duros SATA y tarjetas Gigabit Ethernet; para solucionarlos fue necesario el compilar un nuevo *kernel* con soporte para discos duros SATA y para las tarjetas Gigabit; sin embargo, se debió utilizar otra distribución de Linux conocida como Fedora Core 3.0. Además, el paquete de MPICH que se incluye en la Versión 3.2 de NPACI Rocks no tiene soporte para el lenguaje C++, por lo que se utilizó la Versión 4.0 de NPACI *Rocks* y el sistema operativo Linux CentOS.

- El sistema operativo Linux posee las características y funcionalidades más prominentes de los sistemas operativos que se encuentran disponibles en el mercado. Éste es de mejores características respecto a la velocidad de acceso a los recursos físicos del computador, y mediante el establecimiento de servicios reduce al mínimo el seguimiento de las tareas de configuración. Linux es uno de los sistemas operativos que además de ofrecer las características mencionadas, brinda el beneficio adicional de ser de distribución libre, reduciendo los costos totales inmersos en la instalación de los nodos de un sistema *cluster*.

- La aplicación desarrollada fue concebida con la intención de demostrar las bondades del *cluster*. La aplicación fue desarrollada utilizando el lenguaje C++ y la librería de paso de mensajes MPI; permite resolver Cadenas de Markov que involucran resolución de sistemas de ecuaciones, operaciones en matrices (transpuesta, identidad) y operaciones entre matrices (suma, multiplicación, potencia) haciendo uso de la capacidad computacional que el *cluster* ofrece.

- Como parte de este proyecto, fue muy importante el revisar y comprender el funcionamiento de MPI para desarrollar la aplicación que permita resolver las Cadenas de Markov. MPI permite que las operaciones involucradas para poder enviar y recibir la información desde y hacia los nodos se realicen de forma sencilla y rápida.

- En este trabajo se presentaron tópicos muy útiles acerca de MPI relacionados en el desarrollo de la aplicación seleccionada; sin embargo, se recomienda revisar más a fondo la bibliografía sobre MPI, debido a que MPI presenta aspectos particulares que requieren mayor atención como realizar tareas I/O de forma paralela o hacer uso de topologías para mejorar el desempeño de las aplicaciones desarrolladas.

- Para poder desarrollar la aplicación se revisaron los métodos numéricos asociados a la resolución de sistemas de ecuaciones lineales utilizando el método de Gauss-Jordan, operaciones entre matrices y Series de Taylor.

- Para el desarrollo de la aplicación se utilizaron los sistemas operativos Windows XP Profesional con Service Pack 2 y Windows Server 2003 con Service Pack 1. Se instaló MPICH Versión 1 y 2, y se integraron con el IDE Visual Studio .NET 2003, con el objetivo de agilitar el proceso de escritura del código debido al amplio conocimiento adquirido del entorno de desarrollo durante los años de pregrado. A su vez el

desarrollo de la aplicación sobre un sistema Windows permitió verificar la portabilidad del código fuente, debido a que luego fue compilado, enlazado y ejecutado en el *cluster*.

- El presente trabajo demostró las ventajas del *cluster* en la resolución de las Cadenas de Markov, la aplicación desarrollada consiguió reducir el tiempo que toma el resolver Cadenas de Markov utilizando un computador en un valor muy cercano a la mitad usando dos computadores del *cluster* o en un valor muy cercano a un tercio usando los tres computadores del *cluster*.

- Se recomienda utilizar alternativas de almacenamiento al sistema de archivos de red NFS mediante implementaciones gratuitas como PVFS, Coda, Lustre o implementaciones que requieren inversión como GFS de IBM y así mejorar los tiempos de acceso a los datos.

- Durante la etapa final del desarrollo del Proyecto de Titulación, las implementaciones de OpenPBS fueron reemplazadas en OSCAR y en *Rocks* por alternativas de mejores prestaciones como Torque para OSCAR y SGE para *Rocks*. Se recomienda revisar la información sobre estos planificadores de tareas debido a las mejoras que ofrecen frente a OpenPBS.

- Los *clusters* se presentan como una solución económica viable para obtener una gran capacidad computacional. En junio de 2005, el 60,8% de los 500 computadores más veloces del mundo eran *clusters*. En nuestro país existe una gran cantidad de equipos que se dan de baja o se dejan de usar por que se consideran obsoletos; sin embargo, podrían usarse para construir un *cluster* que presente características de procesamiento realmente buenas, y el costo involucrado en la construcción sería relativamente bajo.

- En el presente trabajo de Titulación se mencionó que se pueden utilizar computadores de diferentes arquitecturas para la implementación de *clusters*; sin embargo se recomienda utilizar computadores que posean las mismas características para mejorar el rendimiento y evitar posibles cuellos de botella al tener computadores de diferentes características cuyos recursos no sean accesibles de manera eficaz. En cuanto a la selección del hardware y de la arquitectura a utilizarse, se debe tomar en consideración aspectos como costo, número de CPUs por nodo, rendimiento (enteros y punto flotante), cantidad de RAM soportada, direccionamiento de memoria (32 - 64 bits), ancho de banda del CPU y de entrada/salida; también se debe considerar el rendimiento que se espera obtener, la disponibilidad de controladores para el sistema operativo escogido, se debe tomar en consideración el espacio físico necesario para ubicar el *cluster* y otros factores involucrados como consumo de energía eléctrica, aire acondicionado, seguridad, entre otros.

- El presente trabajo sólo inicia el camino en las investigaciones de paralelismo y espera ser la base para futuros trabajos que traten de buscar soluciones más avanzadas para producir *clusters* de mejores características, haciendo uso de mejores herramientas. Este trabajo pretende dar las pautas necesarias para construir un *cluster* básico, pero explora muchas soluciones que podrían usarse para producir mejores *clusters;* por ejemplo, con redes de mejores prestaciones como las presentadas brevemente en el Capítulo 1; o usando otras herramientas que permiten construir *clusters* de forma automática como las mostradas de forma breve en el Capítulo 2; o haciendo uso de otras utilidades para desarrollar aplicaciones paralelas como PVM u OpenMP, presentadas en el Capítulo 3.

BIBLIOGRAFÍA

1 BUYYA R. *High Performance Cluster Computing: Architectures and Systems, Volume I*; Prentice Hall, Upper Saddle River, New Jersey, 1999.

2 BUYYA R. *High Performance Cluster Computing: Architectures and Systems, Volume II*; Prentice Hall, Upper Saddle River, New Jersey, 1999.

3 PFISTER, Gregory. *In Search of Clusters: The coming battle in lowly parallel computing*; Prentice Hall, Upper Saddle River, New Jersey, 1995.

4 LUCKE, Robert. *Building Clustered Linux Systems*; Prentice Hall, Upper Saddle River, New Jersey, 2005.

5 SLOAN, Joseph. *High Performance Linux Clusters with OSCAR, Rocks, OpenMosix & MPI*; O'Reilly, Cambridge, 2005.

6 STERLING, Thomas. *Beowulf Cluster Computing with Linux*; The MIT Press, Massachusetts, 2002.

7 STERLING, Thomas. *Beowulf Cluster Computing with Windows*; The MIT Press, Massachusetts, 2002.

8 ABBAS, Ahmar. *Grid Computing*: *A Practical Guide to Technology and Applications*; Charles River Media, Inc. Massachusetts, 2004.

9 PACHECO, Peter. *Parallel Programming with MPI*; Morgan Kauffman, San Francisco, California, 1997.

10 QUINN, Michael. *Parallel Programming in C with MPI and OpenMP*; McGraw Hill Higher Education, New York, 2003.

11 BLANCHETTE, Jasmin; SUMMERFIELD, Mark. *C++ GUI Programming with Qt3*; Prentice Hall, Upper Saddle River, New Jersey, 2004.

12 KREYSZIG, Erwin; *Matemáticas Avanzadas para Ingeniería Volumen II*; Limusa Wiley, México, 2000.

13 HIGGINBOTTOM, Gary; *Performance Evaluation of Communication Networks;* Artech House, Inc., Londres, Inglaterra, 1998.

14 RED HAT, Inc.; *Red Hat Linux Customization Guide*, Red Hat, Inc., Estados Unidos, 2003.

15 IBM Certified Course Material, *Linux System Administration I: Implementation*, IBM, Estados Unidos, 2004.

16 http://www.beowulf.org
Sitio Web con información general acerca de *clusters Beowulf*.

17 http://www.tldp.org
Sitio Web con documentación acerca del sistema operativo Linux.

18 http://rocks.npaci.edu
Sitio Web de *Rocks*.

19 http://oscar.openclustergroup.org
Sitio Web de OSCAR

20 http://www.top500.org
Sitio Web con la lista de los 500 supercomputadores más veloces del mundo.

21 http://www.cms.wisc.edu/~cvg/course/491/modules/Markov/Markov/node2.html
Sitio Web con información acerca de las Cadenas de Markov.

ANEXO A

INSTALACIÓN DE OSCAR

REQUERIMIENTOS DEL NODO SERVIDOR

El nodo central debe poseer un procesador 586 o superior. Requiere de por lo menos 4 GB libres en disco duro: 2 GB para el directorio / y 2 GB para el directorio `/var`. Si el servidor va a conectarse a una red pública, se requieren dos tarjetas de red.

En el nodo central se debe preinstalar una versión de Linux compatible con OSCAR con soporte **X-Windows** y proceder a configurar la información de la red.

Para configurar las tarjetas se puede utilizar alguna de las herramientas propias de la distribución como **netcfg**, **redhat-config-network**, **system-config-network**, **ifconfig** o un editor de texto. En la configuración de red se deben satisfacer los siguientes requerimientos:

- HOSTNAME: Para que la instalación de OSCAR sea exitosa, se debe cambiar el nombre que por defecto la mayoría de distribuciones de Linux asignan a los equipos (`localhost` o `localhost.localdomain`). Se puede cambiar el nombre del equipo con el comando **hostname**, o modificando el archivo `/etc/sysconfig/network`. El nombre puede ser cualquiera pero no debe contener el guión bajo ("_").
- Interfaz de red pública: Debe ser configurada de acuerdo a la red.
- Interfaz de red privada: Esta interfaz estará en contacto con el *cluster*. Para configurarlo se debe tomar en cuenta: utilizar una dirección IP privada, utilizar una máscara de red apropiada y asegurar que la interfaz esté activa cuando inicia el sistema.

REQUERIMIENTOS DE LOS NODOS CLIENTE

Los computadores deben tener alto grado de similitud en su hardware. Deben poseer un procesador 586 o superior. Requieren de por lo menos 4 GB libres en disco duro: 2 GB en el directorio / y 2 GB en el directorio `/var`. No es necesario el disponer de teclado y monitores. Se requieren de disquetes o de habilitar PXE en el BIOS.

DESCARGA Y DESEMPAQUETAMIENTO DE OSCAR EN EL SERVIDOR

Se procede a descargar una copia del servidor OSCAR disponible en el URL:

http://oscar.sourceforge.net/

El paquete de instalación debe ubicarse en `/root` (la carpeta home del usuario root).

```
# cd /root
```

Se procede a desempaquetar el paquete descargado mediante el comando:

```
# tar xvzf <nombre_del_archivo_a_desempaquetar>
```

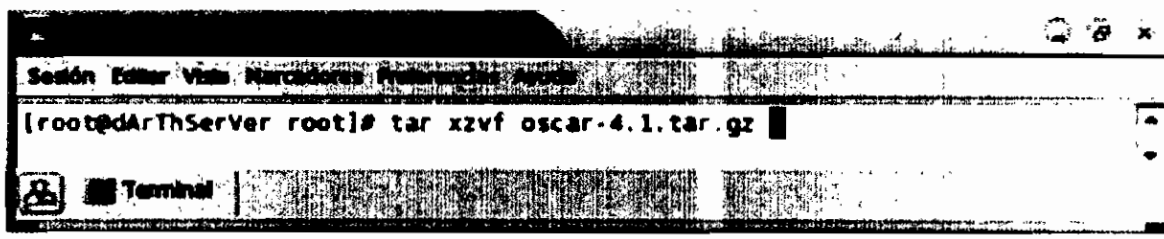


Figura A - 1. Desempaquetado del paquete OSCAR

La opción `x` del comando **tar** permite extraer los archivo contenidos, la opción `v` es para imprimir una lista con los nombres de los archivos que están siendo extraídos, la opción `z` es para descomprimir los archivos mediante **gzip**, y la opción `f` es para especificar el nombre del archivo sobre el cual se van a realizar las operaciones antes mencionadas.

En la Tabla A - 1 se muestran los directorios y archivos que se desempaqueta al ejecutar el comando **tar** y una descripción del contenido de cada uno. En los siguientes ejemplos y figuras se muestra la ejecución de comandos y resultados sobre OSCAR Versión 4-1.

Elemento	Descripción
~/oscar-4.1/	Directorio base de OSCAR.
~/oscar-4.1/COPYING	Licencia Pública General GNU.
~/oscar-4.1/dist	Scripts de la distribución para configuración e instalación.
~/oscar-4.1/doc	Directorio con documentación de OSCAR.
~/oscar-4.1/images	Imágenes utilizadas por el GUI.
~/oscar-4.1/install_cluster	*Script* de instalación principal.
~/oscar-4.1/lib	Rutinas de las librerías auxiliares.
~/oscar-4.1/packages	Archivos de instalación y RPMs para los paquetes OSCAR.
~/oscar-4.1/README	Documento de texto README.
~/oscar-4.1/scripts	Contienen los *scripts* que hacen la mayoría del trabajo de instalación.
~/oscar-4.1/share	Archivos de ayuda auxiliares.
~/oscar-4.1/testing	Contiene scripts para realizar pruebas.
~/oscar-4.1/VERSION	Archivo que contiene el número de la versión de OSCAR.

Tabla A - 1. Directorios y Archivos de OSCAR

CONFIGURACIÓN E INSTALACIÓN DEL SERVIDOR

Se procede a ejecutar el *script* de configuración que se encuentra en el directorio principal de OSCAR. En la Figura A - 2 se muestran un ejemplo de la ejecución de este *script*.

```
# cd /root/oscar-<version instalada>
# ./configure
```

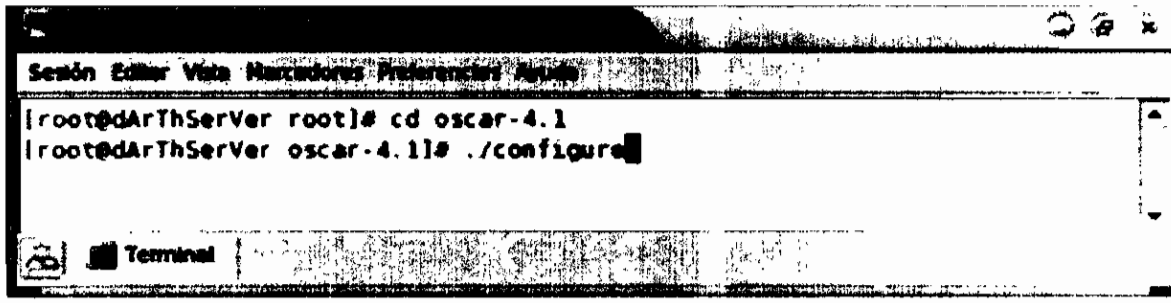


Figura A - 2. Ejecución del *script* de configuración

El *script* `configure`, configura los archivos fuente de OSCAR para su compilación de acuerdo al sistema. La salida del *script* `configure` se muestra en la Figura A - 3.

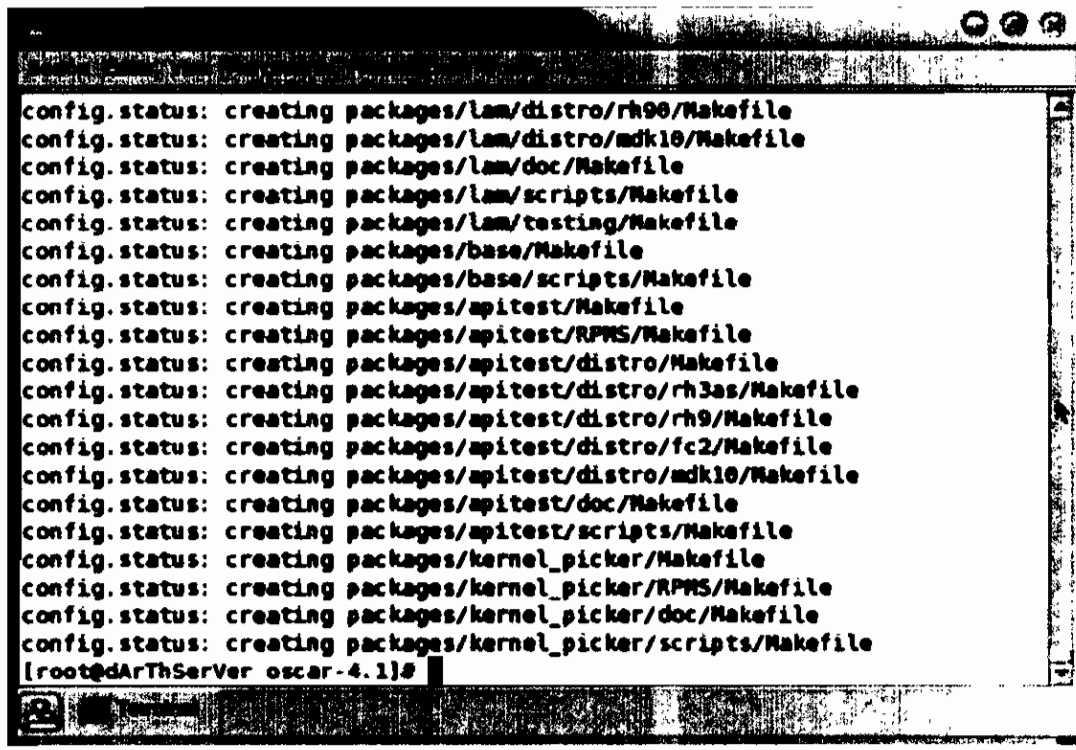


Figura A - 3. Salida del *script* de configuración

Una vez que el script de configuración complete exitosamente, se requiere compilar OSCAR, para lo cual se ejecuta el comando:

```
# make install
```

La salida del comando **make** se muestra en la Figura A - 4.

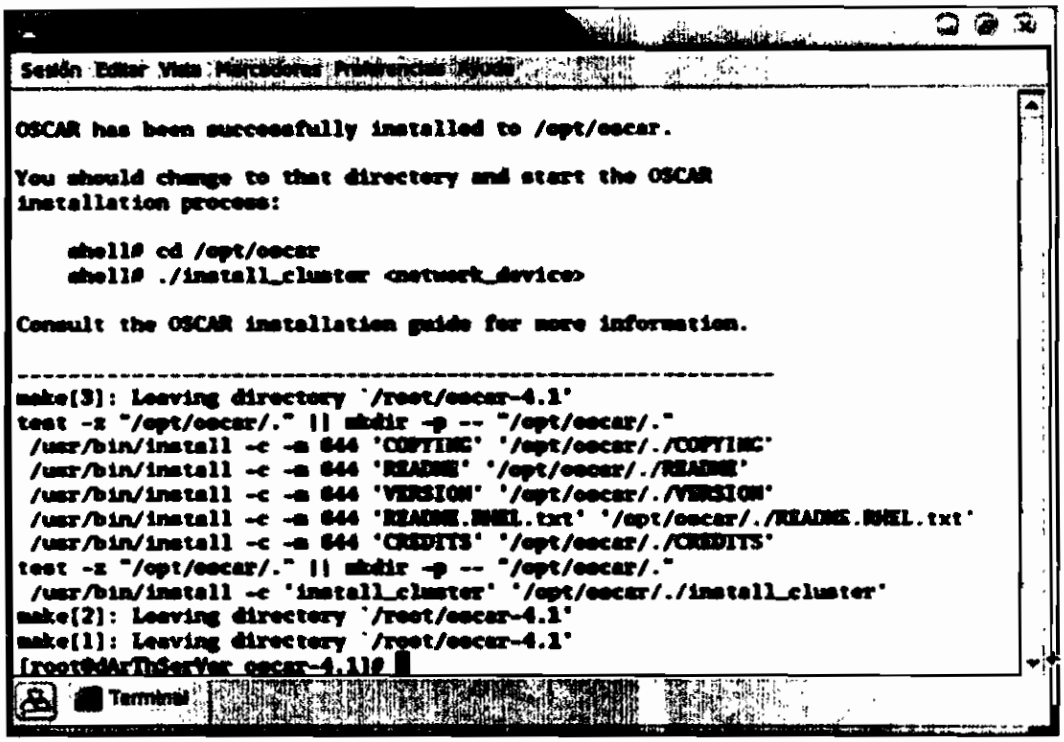


Figura A - 4. Ejecución de `make install`

Se procede a copiar los RPMs de la distribución escogida al directorio `/tftpboot/rpm`. Para cada CD, se debe localizar el directorio que contiene los RPMs. Por ejemplo en Red Hat 9.x, los RPMs están localizados en el directorio `RedHat/RPMS`. En la Figura A - 5 se muestra la copia del contenido de los CDs de Mandrake al directorio `/tftpboot/rpm`.

```
# mount /mnt/cdrom
# cp /mnt/cdrom/RedHat/RPMS/*.rpm /tftpboot/rpm
```

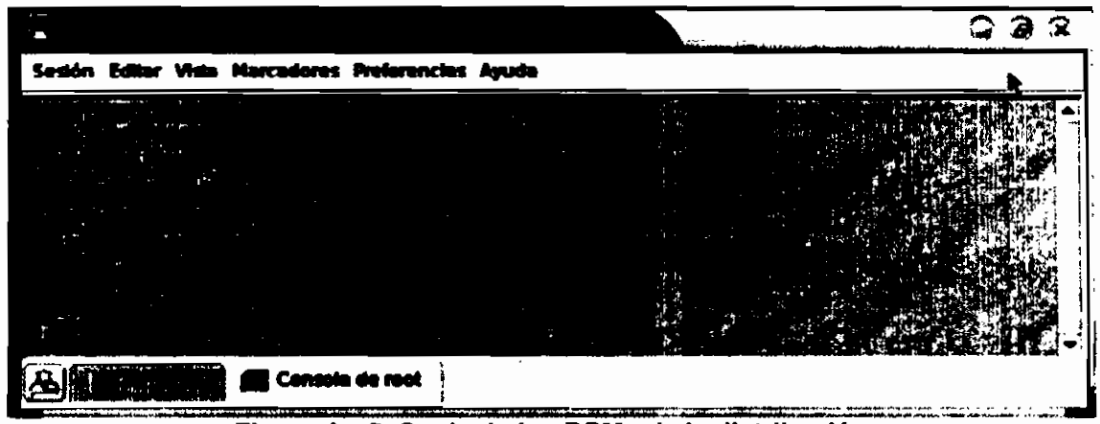


Figura A - 5. Copia de los RPMs de la distribución

ASISTENTE DE OSCAR (*Wizard*)

Se procede a ejecutar el *script* de instalación del *cluster*: `install_cluster` especificando la interfaz de red de la red privada. Al ejecutar este *script*, una gran cantidad de información se mostrará en la consola donde se invocó al *script*. Esto refleja la salida de las operaciones que los comandos de instalación están realizando. Está salida también se almacena en el archivo `oscarinstall.log`. La salida del *script* de instalación de OSCAR se muestra en la Figura A - 6.

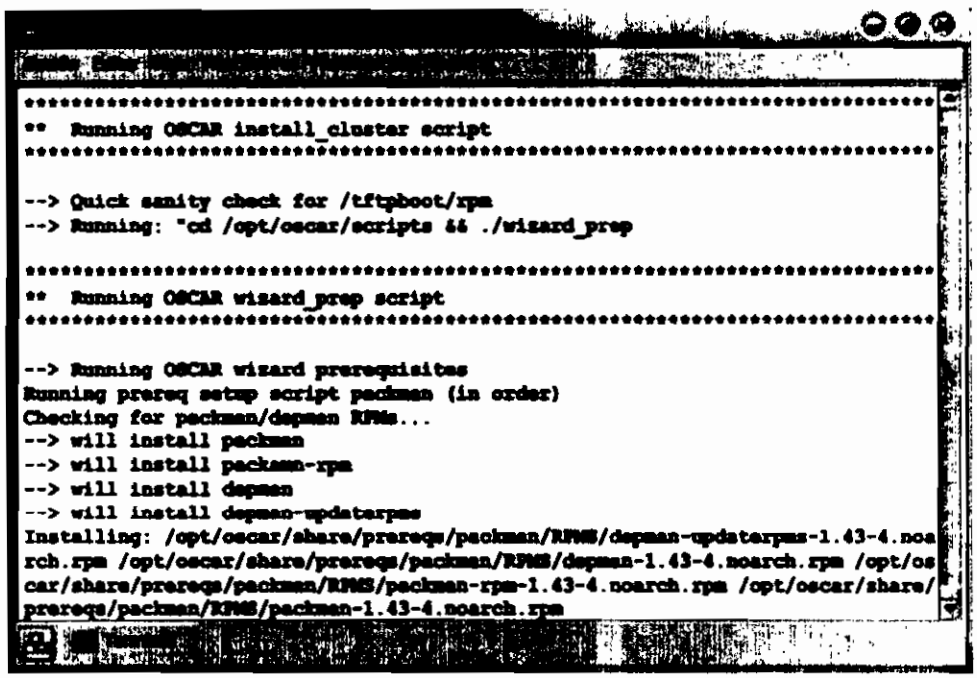


Figura A - 6. Ejecución del *script* de instalación del *cluster*

Básicamente, el *script* de instalación realiza las siguientes operaciones:

1. Instala los paquetes de prerrequisitos en el servidor.
2. Copia los RPMs de OSCAR al directorio `/tftpboot/rpm`.
3. Instala todos los paquetes de OSCAR en el servidor.
4. Actualiza el archivo `/etc/hosts` con los alias que utiliza OSCAR.
5. Actualiza el archivo `/etc/exports`.
6. Se agregan el *path* de OSCAR al archivo `/etc/profile`.
7. Se actualiza los *scripts* de inicio del sistema (`/etc/rc.d/init.d`).
8. Se reinicializan los servicios afectados.

Para proceder con la instalación se ejecuta el comando:

```
# ./install_cluster <interfaz_de_red_privada>
```

Donde `<interfaz_de_red_privada>` es el nombre de la interfaz de red del *cluster* (por ejemplo eth0).

Una vez que el *script* de instalación del *cluster* termina su ejecución, el *Wizard* de instalación de OSCAR aparecerá automáticamente. La Figura A - 7 muestra la interfaz gráfica del *Wizard* de OSCAR.

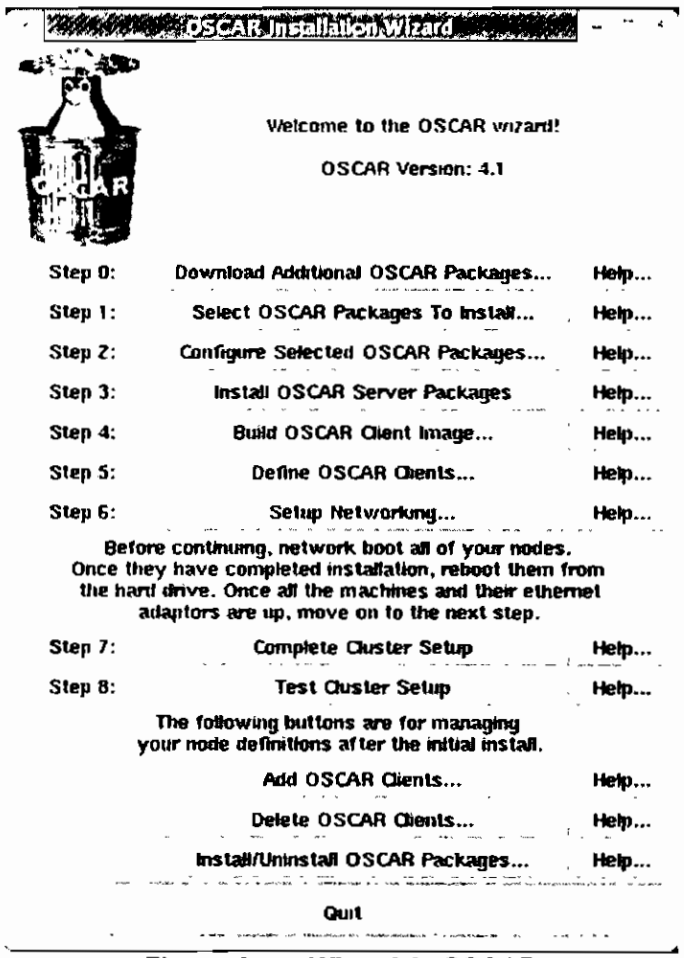


Figura A - 7. *Wizard* de OSCAR

El Wizard es una guía en el resto de la instalación del *cluster*. El Wizard consta de una serie de pasos que deben seguirse para completar la instalación. El Wizard también permite agregar o eliminar nodos, e instalar o desinstalar paquetes de OSCAR. El Wizard dispone de un botón **<Help>** el cual muestra una ventana con información del propósito de cada paso.

Paso 0 (*Step 0*)

El primer paso del *Wizard* permite descargar paquetes adicionales utilizando OPD. Este paso es opcional y se invoca al presionar el botón **<Download Additional OSCAR Packages>**. El *Wizard* utiliza un GUI conocido como **OPDer**, para evitar el acceder a la herramienta OPD basada en línea de comandos. En la Figura A - 8 se muestra la interfaz gráfica de **OPDer**.

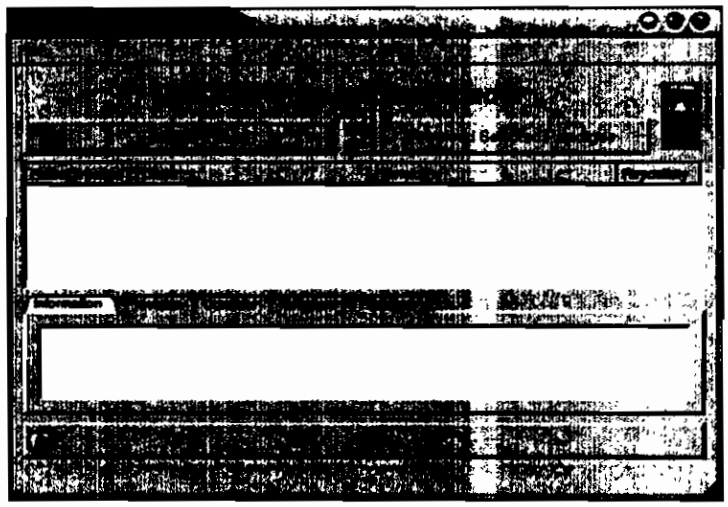


Figura A - 8. Interfaz de OPDer

OPDer permite seleccionar los paquetes desde la lista de OSCAR o permite agregar el URL del servidor desde el cual se desea descargar los paquetes. En la Figura A - 9 se muestra la ventana que permite ingresar el URL de un servidor que dispone de paquetes.

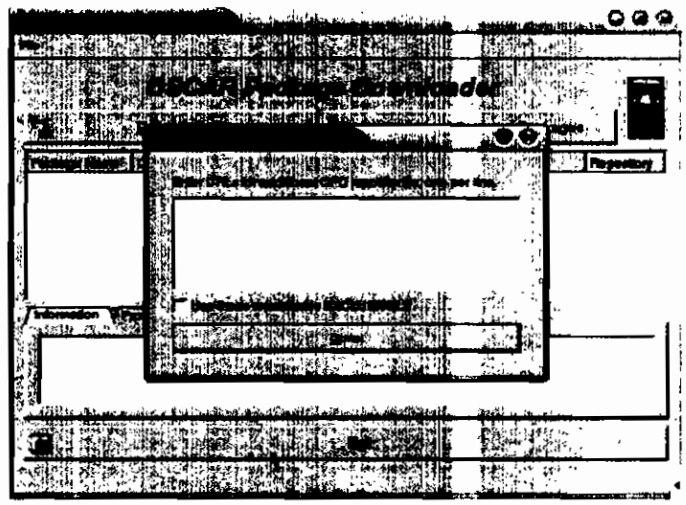


Figura A - 9. Ventana para ingreso de URLs de contenedores OPD adicionales

Paso 1 (*Step 1*)

Este paso del *Wizard* permite escoger los paquetes que serán instalados. Es opcional y se invoca al presionar el botón **<Select OSCAR Packages To Install>**; por defecto, todos los paquetes incluidos en OSCAR son seleccionados e instalados. Sin embargo, si se descargo algún paquete adicional con OPD/**OPDer** se debe seleccionarlo utilizando este paso. En la Figura A - 10 se muestra la interfaz gráfica del Paso 1.

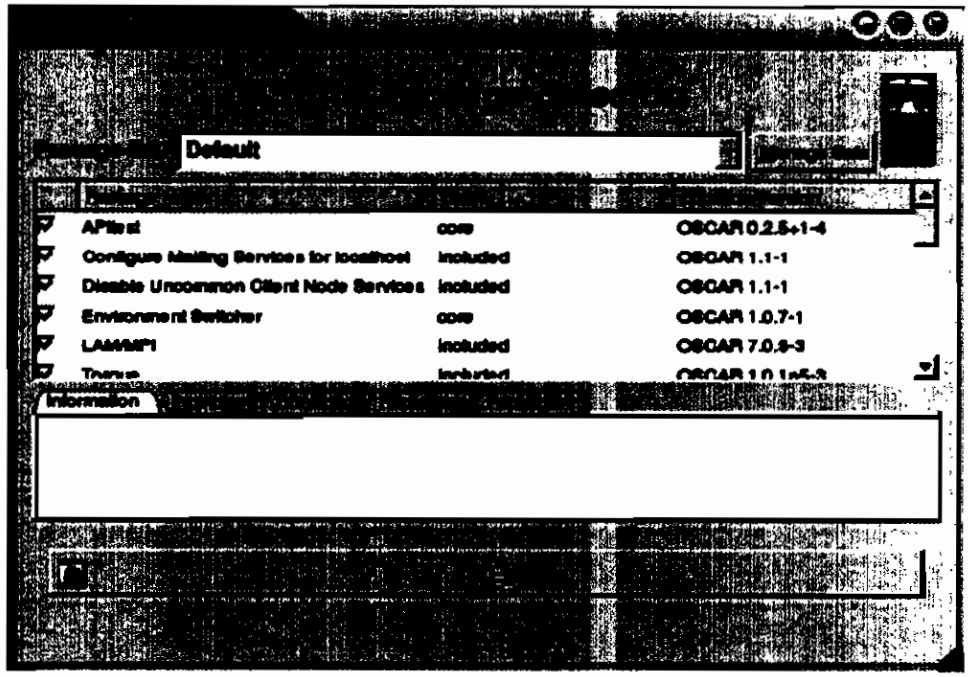


Figura A - 10. Ventana para seleccionar los paquetes OSCAR

Paso 2 (*Step 2*)

El paso 2 del *Wizard* permite configurar algunos paquetes, es opcional y se invoca al presionar el botón **<Configure Selected OSCAR Packages>**. Este paso permite configurar los paquetes: **Environment Switcher**, **Ganglia**, **kernel_picker**, **ntpconfig**, **Torque**. En la Figura A - 11 se muestra la interfaz gráfica del Paso 2.

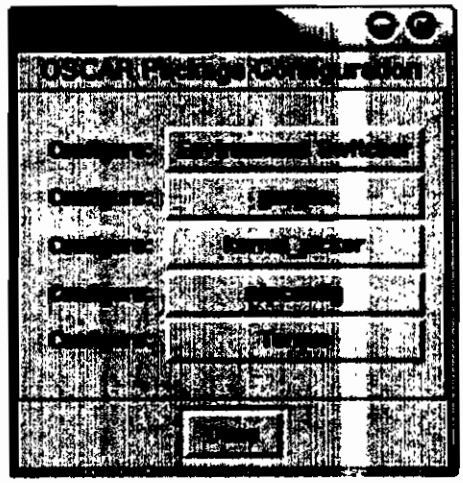


Figura A - 11. Ventana para configuración de paquetes

Para el paquete **Environment Switcher** se puede escoger la configuración por defecto para la etiqueta MPI: ninguna, LAM o MPICH. En la Figura A - 12 se muestra la ventana que permite configurar la etiqueta MPI mediante **Environment Switcher**.

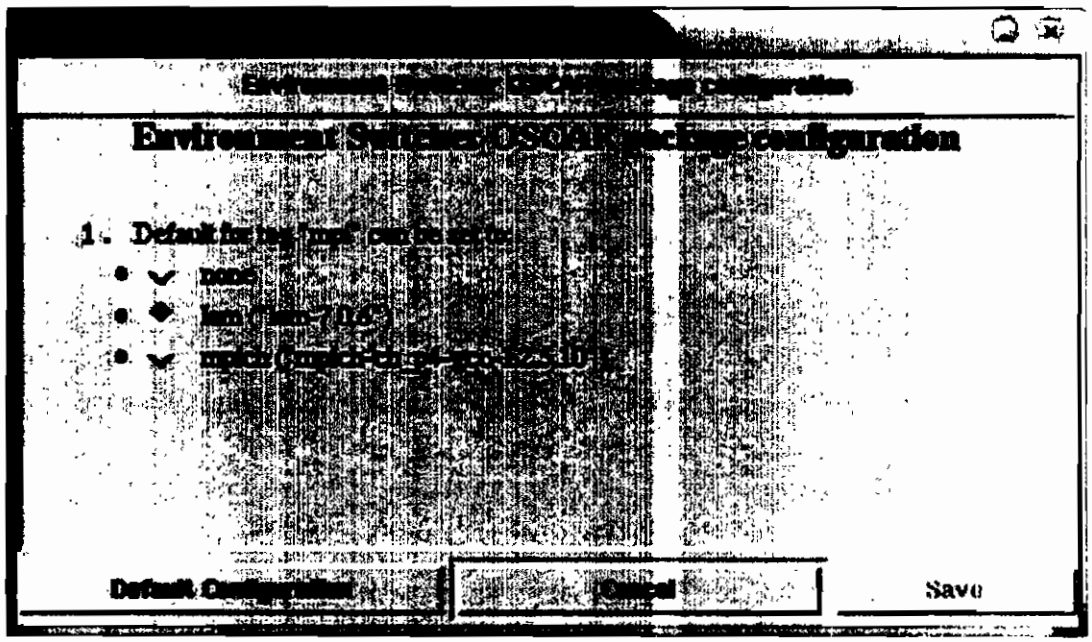


Figura A - 12. Ventana para configuración de Environment Switcher

Para el paquete **Ganglia** se puede definir el nombre del *cluster*, el nombre del *grid*, el propietario del *cluster*, entre otros. En la Figura A - 13 se muestra la ventana que permite configurar **Ganglia**.

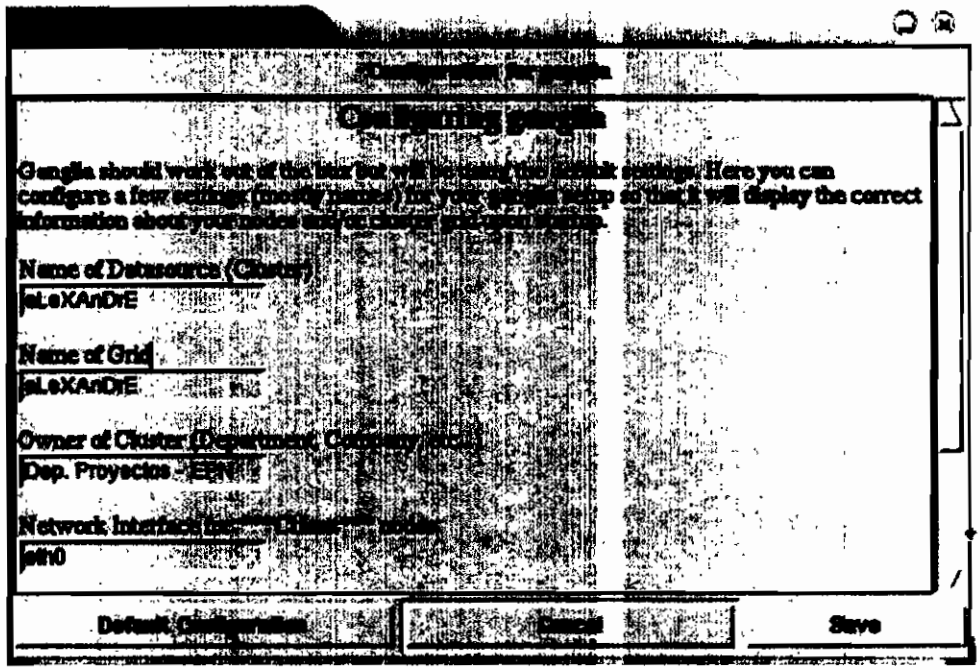


Figura A - 13. Ventana para configuración de Ganglia

Para el paquete **kernel_picker** se puede escoger el kernel que se instalará en la imagen de los nodos cliente. La ventana para configuración de **kernel_picker** se muestra en la Figura A - 14.

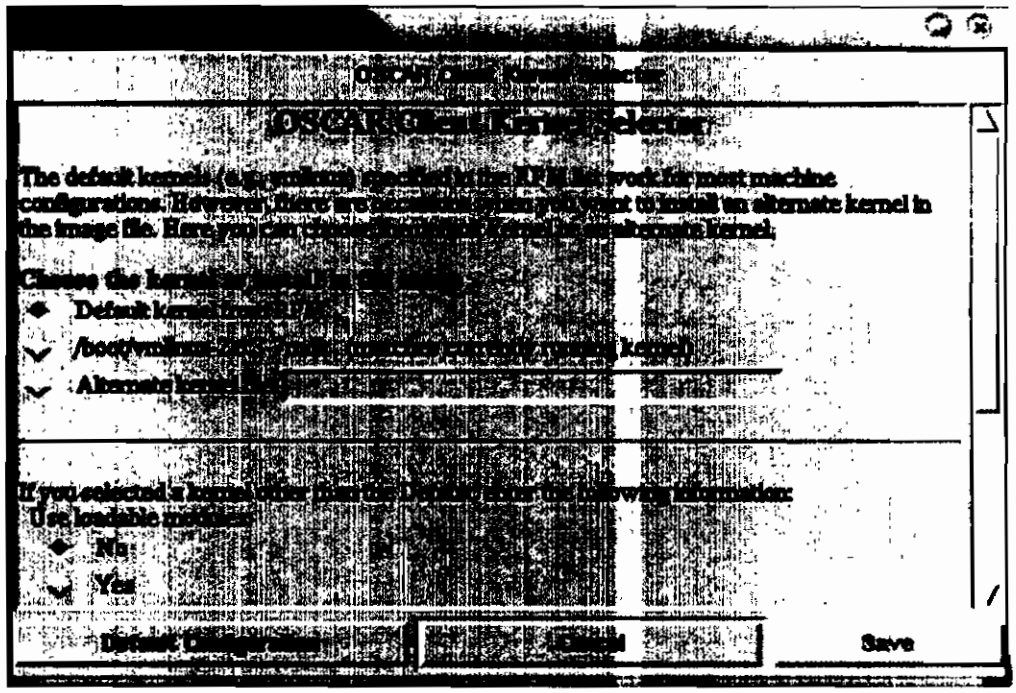


Figura A - 14. Ventana para selección del kernel

Para el paquete **ntpconfig** se puede definir el nombre del servidor NTP. La ventana para configuración de NTP se muestra en la Figura A - 15.

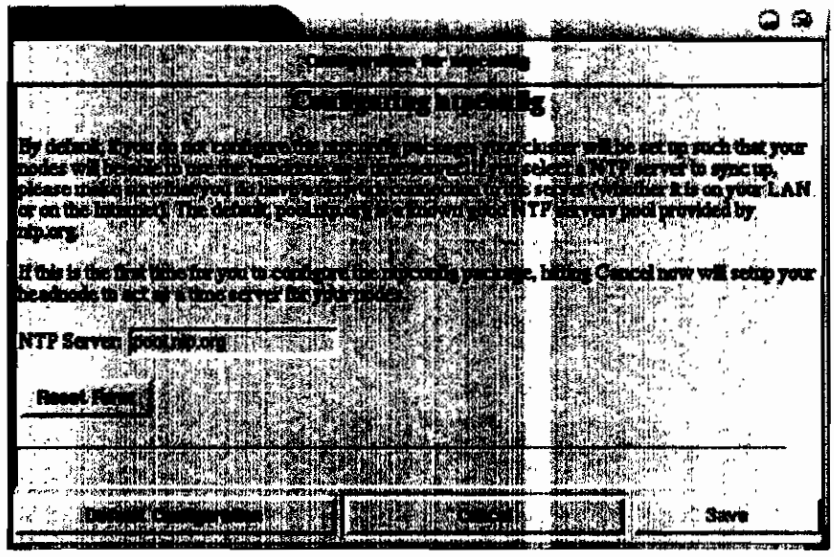


Figura A - 15. Ventana para configuración de ntpconfig

Paso 3 (*Step 3*)

El paso 3 del *Wizard* instala los paquetes de OSCAR en el servidor. Este paso se invoca al presionar el botón **<Install OSCAR Server Packages>** En este paso se instala y configura a varios RPMs. La ejecución de este paso puede tomar varios minutos, los mensajes de salida se despliegan en otra ventana. Los mensajes de salida se muestran en la Figura A - 16.

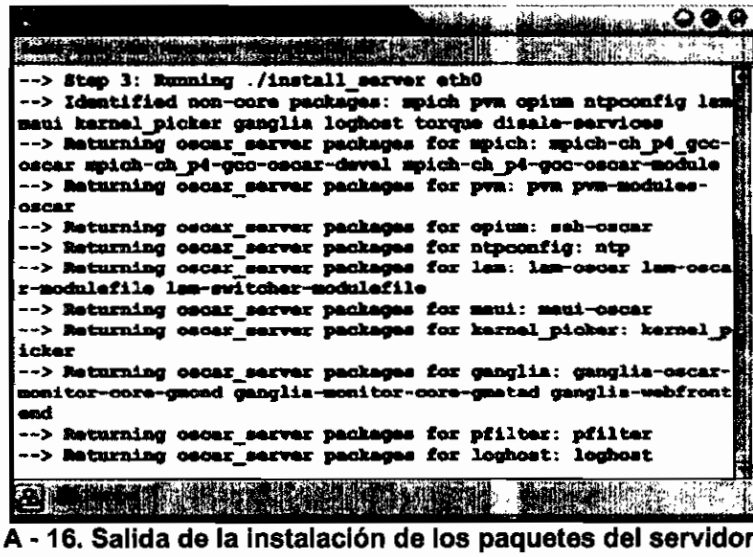


Figura A - 16. Salida de la instalación de los paquetes del servidor OSCAR

Si la instalación ha sido exitosa, se presenta un mensaje como el mostrado en la Figura A - 17.

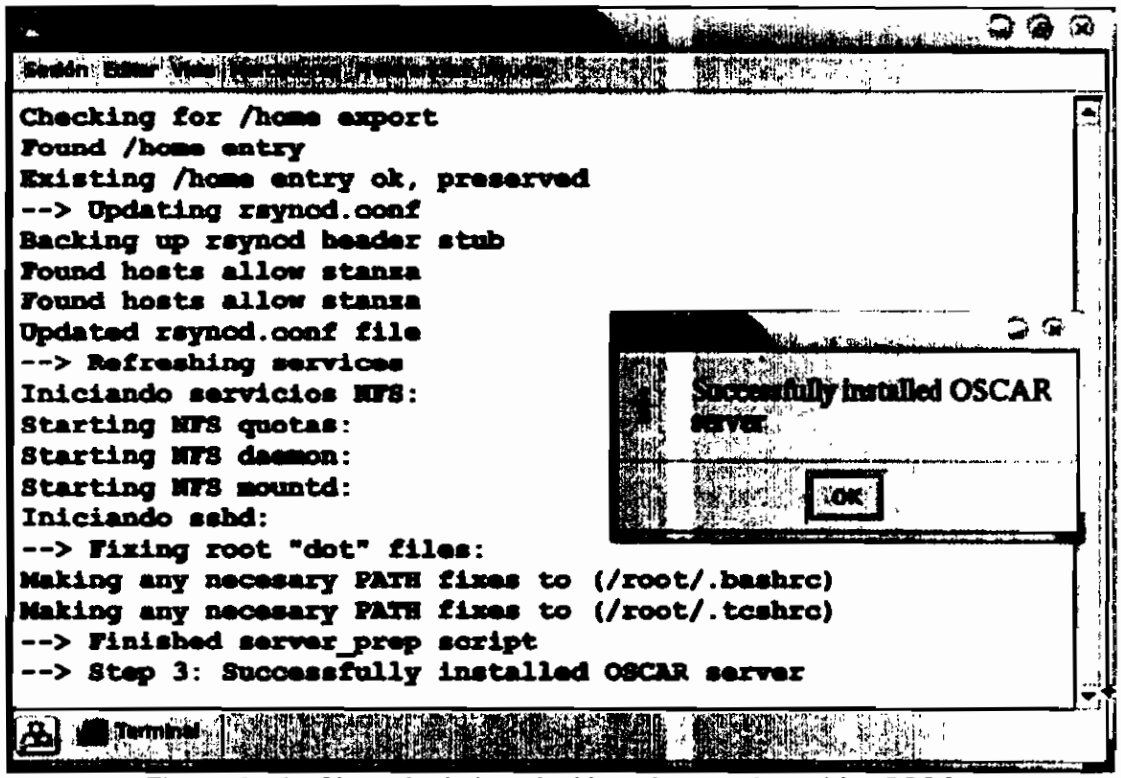


Figura A - 17. Mensaje de instalación exitosa del servidor OSCAR

Paso 4 (*Step 4*)

Antes de realizar el paso 4, se debe permitir que el usuario *root* pueda utilizar **ssh** para conectarse con los nodos cliente. Para esto se debe editar el archivo `/etc/ssh/sshd_config` y buscar la opción `PermitRootLogin` y colocarla en `yes`. Además, se deben revisar que todo el tráfico dentro de la red privada esté permitido, para lo cual se deben revisar los archivos `/etc/host.allow` y `/etc/host.deny`. Si estos requerimientos no son cumplidos, el paso 4 fallará.

El paso 4 se invoca al presionar el botón **<Build OSCAR Client Image>**. Este paso dispone de una ventana que permite seleccionar la configuración de la imagen en la que se basan los clientes. En la Figura A - 18 se muestra la ventana que permite construir la imagen SIS.

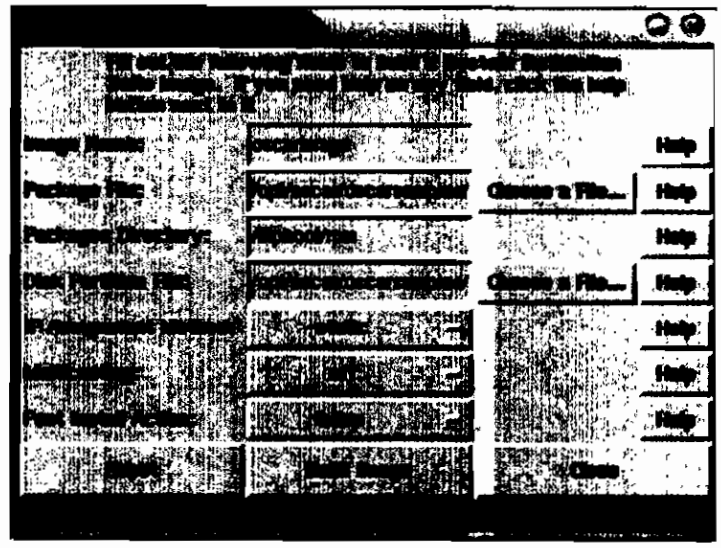


Figura A - 18. Ventana para construcción de la imagen SIS

Se puede personalizar la imagen que se desea generar. Entre los aspectos que se pueden personalizar están:

- **Image Name**: Permite especificar el nombre de la imagen que se creará.

- **Package file**: Contiene una lista con los nombres de los RPMs que van a ser instalados en los clientes.

- **Disk Partition File**: Contiene el archivo de configuración de las particiones del disco duro de los clientes. Utiliza el siguiente formato:
  `<partition> <size in megabytes> <type> <mount point> <options>`

- **IP Assignement Method**: Permite escoger el método de asignación de las direcciones IP: usando DHCP o de forma estática.

- **Post Install Action**: Permite definir la acción que se tomará una vez finalizada la instalación: un pitido para indicar la finalización o reiniciar el sistema.

Al presionar **<Choose a file...>** en los ítems **Package File** y **Disk Partition File** se despliega una ventana que permite seleccionar el archivo que tenga lo pedido.

En la Figura A - 19 se muestra la ventana que permite seleccionar los archivos con la lista de RPMs o con la información de cómo particionar el disco duro.

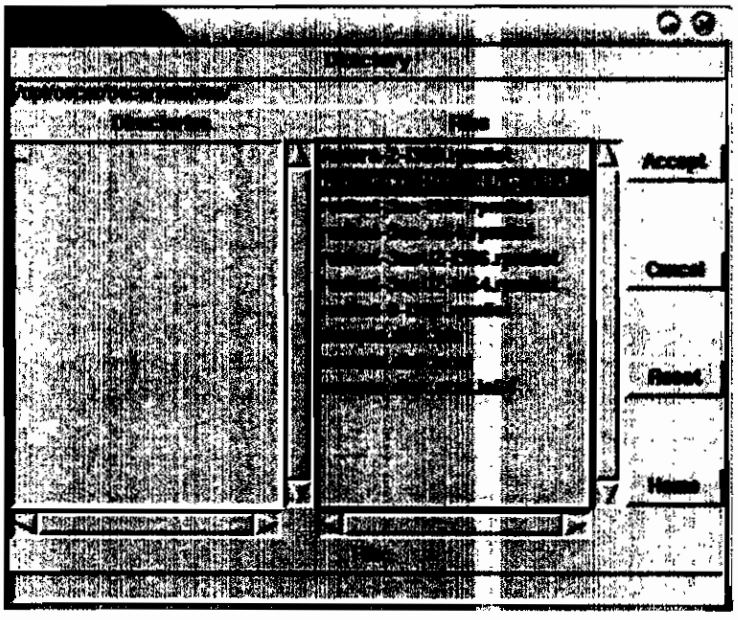


Figura A - 19. Cuadro de diálogo que permite seleccionar un archivo de configuración

Si la creación de la imagen SIS ha sido exitosa, se presenta una pantalla como la mostrada en la Figura A - 20.

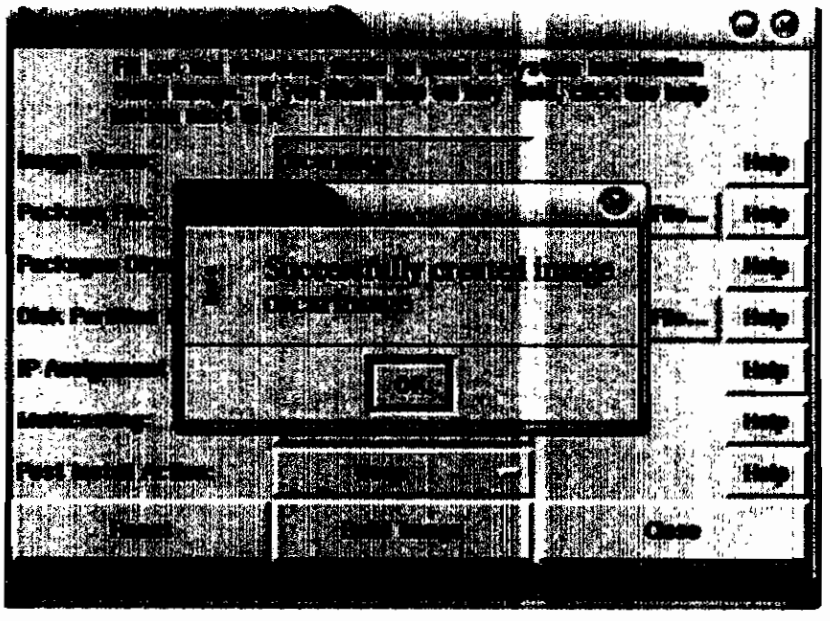


Figura A - 20. Creación exitosa de la imagen SIS

Paso 5 (*Step 5*)

El paso 5 permite definir información acerca de los clientes OSCAR. Se invoca al presionar el botón <**Define OSCAR Clients**>. La Figura A - 21 muestra la interfaz gráfica que permite definir a los clientes.

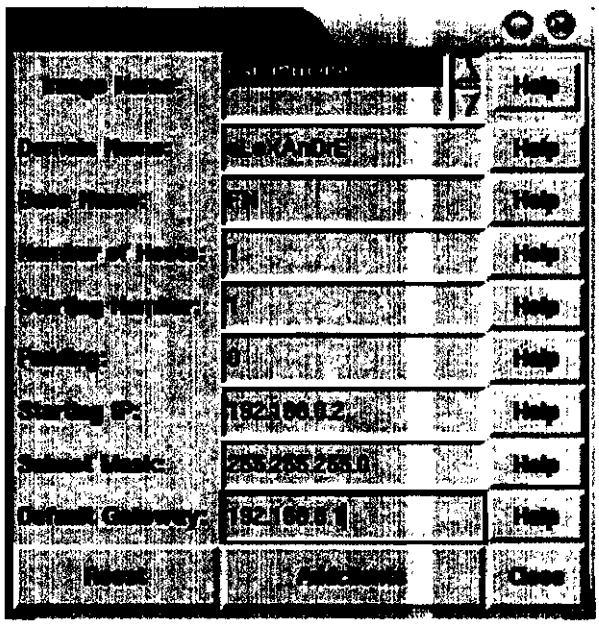


Figura A - 21. Definición de los clientes OSCAR

El dialogo permite definir información de los clientes como:

1. **Image Name**: Permite escoger entre las imágenes creadas.

2. **Domain Name**: Permite especificar el nombre del dominio. Si el servidor dispone de un nombre de dominio se muestra este nombre, caso contrario por defecto mostrará `oscardomain`. Este campo debe tener un valor, no puede estar en blanco.

3. **Base name**: Permite especificar el prefijo de los nombres de los nodos cliente. Este campo no puede contener el carácter "_".

4. **Number of Hosts**: Permite especificar la cantidad de clientes que se van a generar. Este campo debe ser mayor a 0.

5. **Starting Number**: Especifica el índice que se va a agregar al `Base Name`. En cada cliente, este valor será incrementado de forma automática.

6. **Padding**: Permite especificar un número de dígitos que servirán de relleno entre el `Base Name` y el `Starting Number`.

7. **Starting IP**: Permite especificar la primera dirección IP del primer cliente. Este valor se incrementará para cada cliente subsiguiente.

8. **Subnet Mask**. Especifica la máscara de subred de los clientes.

9. **Default Gateway**: Define la ruta de acceso por defecto.

Al terminar de completar esta información, se procede a presionar el botón **<Addclients>**.

En la Figura A - 22 se muestra el mensaje obtenido al definir exitosamente los clientes para la imagen SIS.

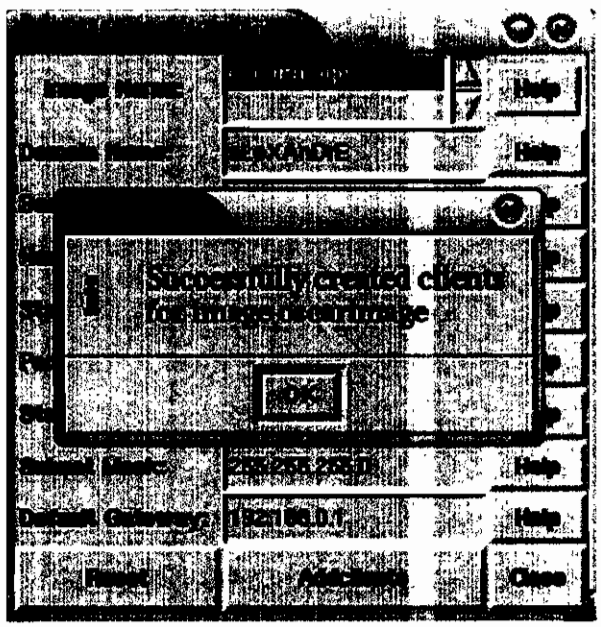


Figura A - 22. Creación de clientes para la imagen SIS

Paso 6 (*Step 6*)

El paso 6 permite configurar la red. Una vez presionado el botón **<Setup Networking>** se muestra un dialogo de configuración inicial. En la Figura A - 23 se muestra la ventana de configuración de red.

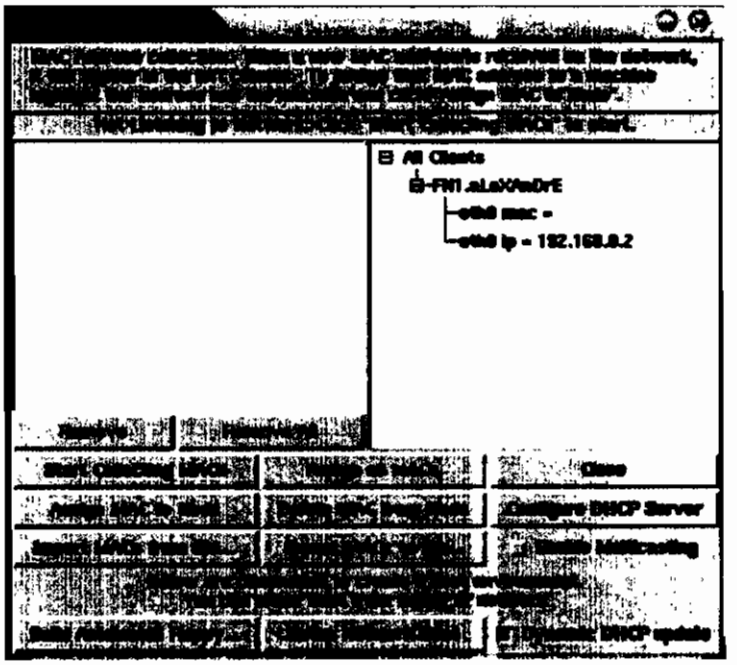


Figura A - 23. Configuración de la red

Este dialogo permite configurar el método de inicio de los clientes. Con el botón **<Build Autoinstall Floppy...>** se puede construir un disquete de inicio para los nodos que no soporten PXE. En la Figura A - 24 se muestra el diálogo que se muestra al construir disquetes de auto instalación. La Figura A - 25 muestra las salidas obtenidas al construir un disquete de auto instalación.

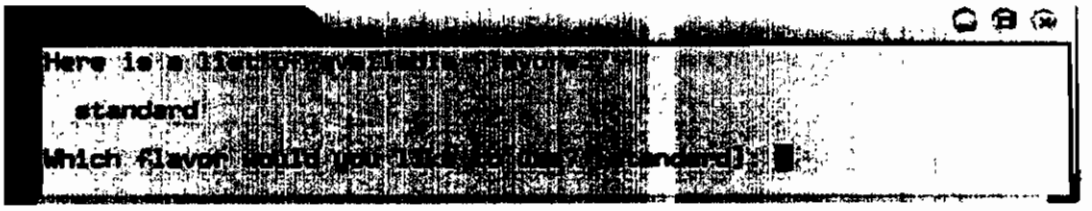


Figura A - 24. Construcción del disquete de instalación

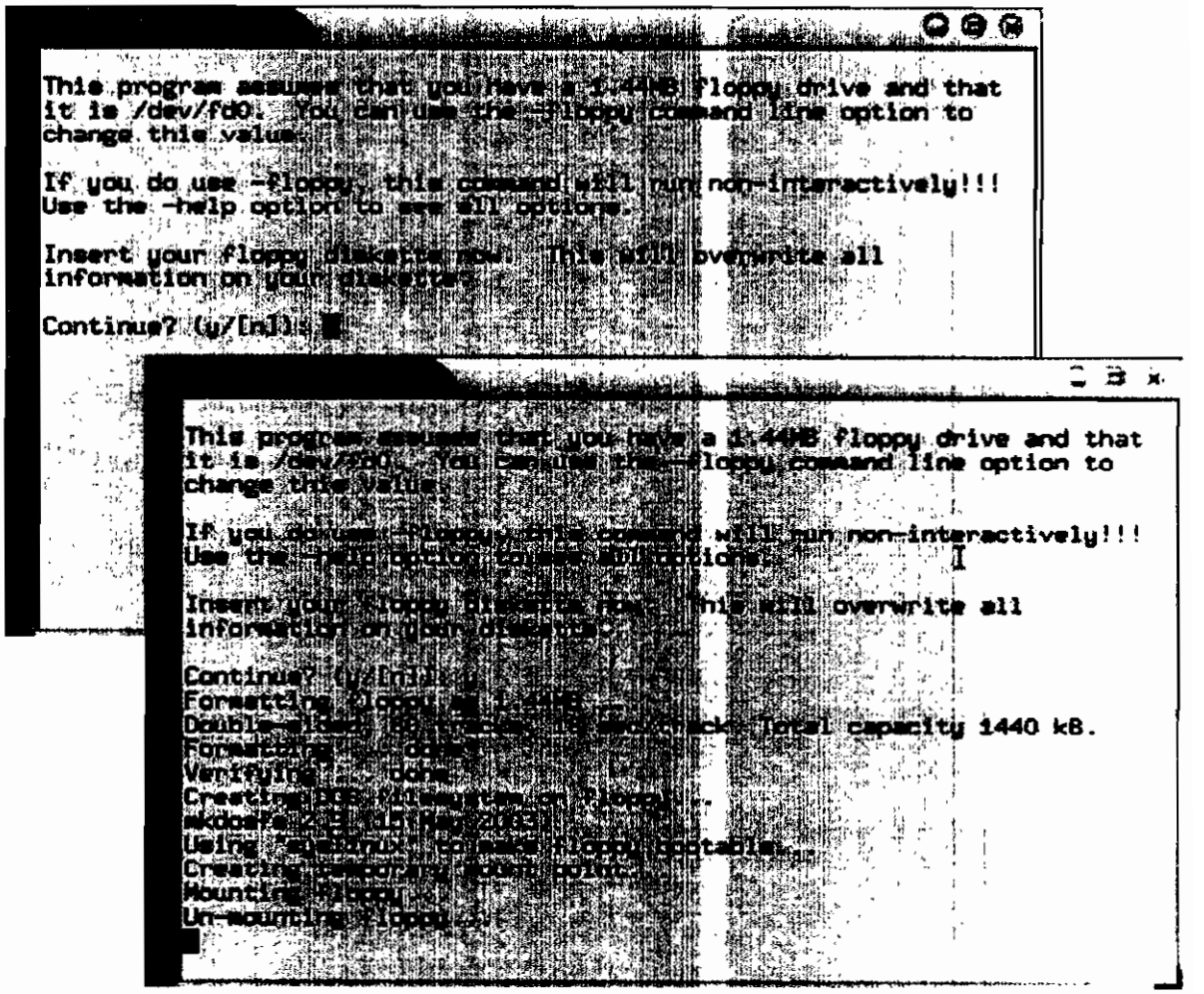


Figura A - 25. Salidas obtenidas al construir el disquete de instalación

Con el botón **<Setup Network Boot>** se configura al servidor para responder a los pedidos de inicio de los clientes mediante PXE.

Se procede a iniciar a los clientes, ya sea mediante PXE o usando el disquete de auto instalación y se procede a presionar el botón de recolección de direcciones MAC **<Collect MAC Address>**. En la Figura A - 26 se muestra el arranque de un cliente. El cliente carga la versión del *kernel* que estaba en el disquete o aquella que fue transferida mediante PXE.

```
SYSLINUX 1.76 Mandrake Linux  Copyright (C) 1994 2002 H. Peter Anvin.

                    Welcome to SystemImager

  This is the SystemImager autoinstall system v3.3.2.  If you do not want
  this system to be autoinstalled, power off now!

  The autoinstall system will partition your hard drives, create filesystems
  on those partitions, and create your swap space.  It will then contact your
  imageserver and install the designated image on this machine.

  If you find this software useful, please consider making contributing code
  or documentation to the project.  Thanks!

  You can find documentation and support at http://systemimager.org

  Have fun!

boot _
```

Figura A - 26. Arranque de un cliente

El cliente envía un mensaje DHCPDISCOVER a la dirección IP de *broadcast*. El servidor captura la dirección MAC del cliente del mensaje DHCPDISCOVER. Las salidas del cliente se muestran en la Figura A - 27.

```
variableize_kernel_append_parameters
read_kernel_append_parameters
read_local_cfg
Checking for floppy diskette.
YOU MAY SEE SOME "wrong magic" ERRORS HERE, AND THAT IS NORMAL.
cramfs: wrong magic
Found floppy diskette.
No /local.cfg on floppy diskette.
read_kernel_append_parameters
start_network
IP Address not set with pre-boot settings.
sleep 0:  This is to give your switch (if you're using one) time to
          recognize your ethernet card before we try the network.
          Tip: You can use <ctrl>+<c> to pass the time (pun intended)

dhclient
Internet Software Consortium DHCP Client 2.0rc13
Copyright 1995, 1996, 1997, 1998, 1999 The Internet Software Consortium.
All rights reserved.

Please contribute if you find this software useful.
For info, please visit http://www.isc.org/dhcp-contrib.html

eth0: Using EEPROM-set media 100baseTx-FDX.
Listening on LPF/eth0/00:03:ff:e8:57:12
Sending on   LPF/eth0/00:03:ff:e8:57:12
Listening on LPF/lo/{nu||}
Sending on   LPF/lo/{nu||}
Sending on   Socket/fallback/fallback-net
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 3
DHCPDISCOVER on lo to 255.255.255.255 port 67 interval 5
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 8
DHCPDISCOVER on lo to 255.255.255.255 port 67 interval 6
DHCPDISCOVER on lo to 255.255.255.255 port 67 interval 6
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 10
DHCPDISCOVER on lo to 255.255.255.255 port 67 interval 10
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 16
DHCPDISCOVER on lo to 255.255.255.255 port 67 interval 7
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 13
DHCPDISCOVER on lo to 255.255.255.255 port 67 interval 15
```

Figura A - 27. Envió de la dirección MAC del cliente mediante un mensaje DHCPDISCOVER

Una vez que se dispone de todas las direcciones MAC de los clientes se puede hacer clic en **<Assign MAC to Node>** para asignar la dirección seleccionada a un nodo específico; o se puede presionar **<Assign all MACs>** para asignar todas las direcciones IP definidas en el paso 5 a todas las direcciones MAC recogidas. Al terminar de recolectar las direcciones MAC se debe hacer clic en **<Stop Collecting MACs>**. En la Figura A - 28 se muestra la captura de direcciones MAC de los nodos cliente. En la Figura A - 29 se muestra la asignación de las direcciones MAC a las imágenes de cada cliente.

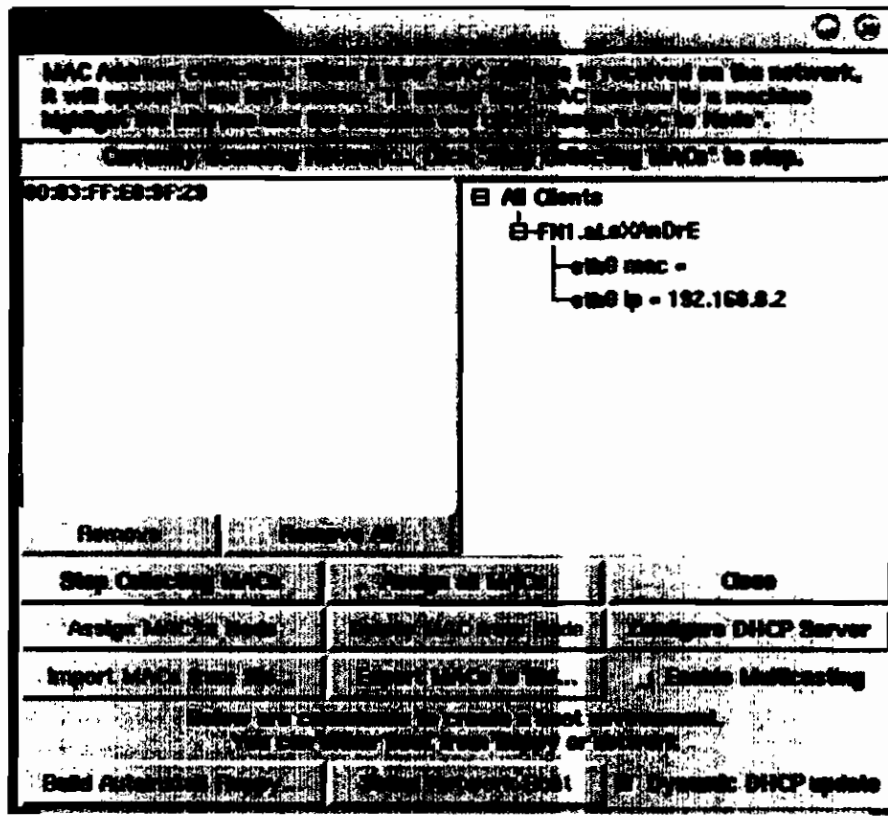


Figura A - 28. Obtención de las direcciones MAC de los clientes

Para remover direcciones MAC extrañas, se dispone del botón **<Remove>**, si se desea borrar todas las direcciones MAC se dispone de **<Remove all>**.

En cualquier momento, se puede hacer clic en **<Export MACs to file...>**, para guardar las direcciones en un archivo. Si se conoce las direcciones MAC de los nodos clientes se puede importar la lista desde un archivo haciendo clic en **<Import MACs from file...>**

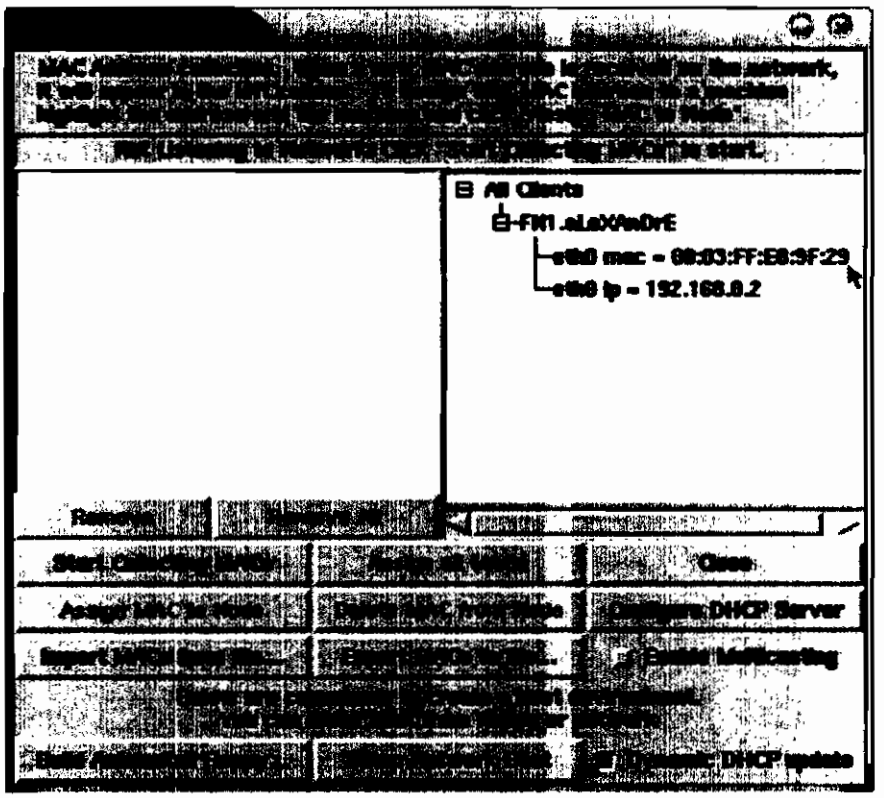


Figura A - 29. Asignación de las direcciones MAC a las imágenes de los clientes

INSTALACIÓN DE LOS NODOS CLIENTE

Durante esta fase, los nodos cliente deben iniciar desde la red a través de PXE o del disquete. Una vez que inicien, automáticamente serán instalados y configurados. Después de varios minutos, los clientes terminarán la instalación. Una vez terminado el proceso, los clientes se detendrán, reiniciarán o emitirán un pitido constante.

El nodo inicia por segunda vez para descargar la imagen, primero procede a cargar los controladores requeridos para funcionar. La salida del inicio del cliente y la carga de controladores se muestra en la Figura A - 30.

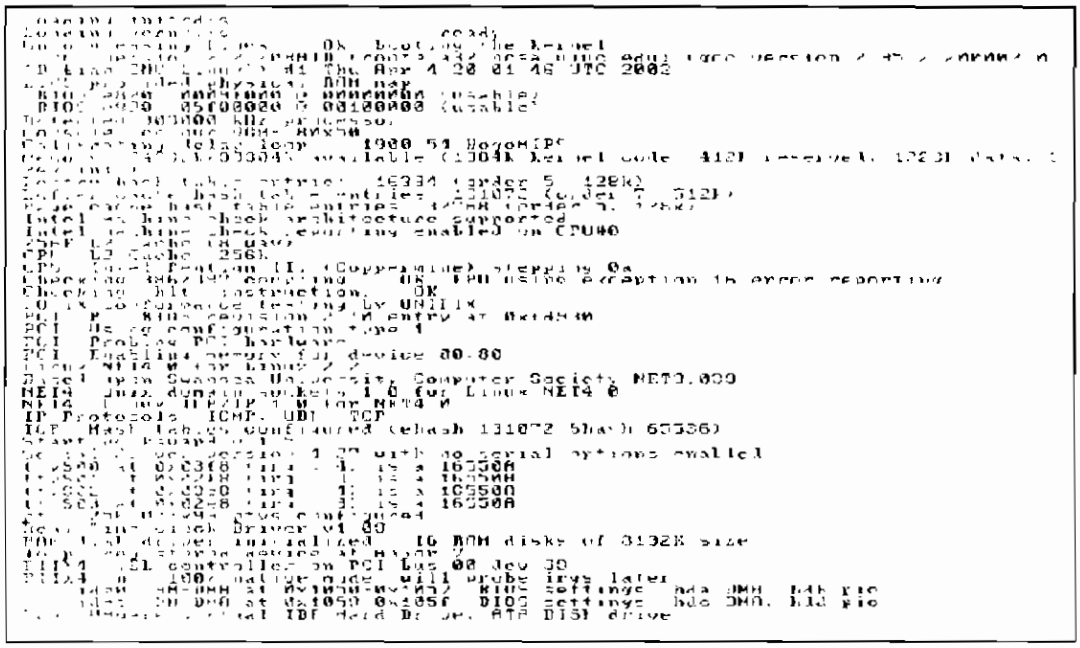


Figura A - 30. Reinicio del nodo cliente y carga de controladores necesarios

De forma automática se procede a particionar el disco duro de acuerdo a la configuración especificada en el paso 5, formatear el disco duro y configurar la tabla de particiones. La salida se muestra en la Figura A - 31.

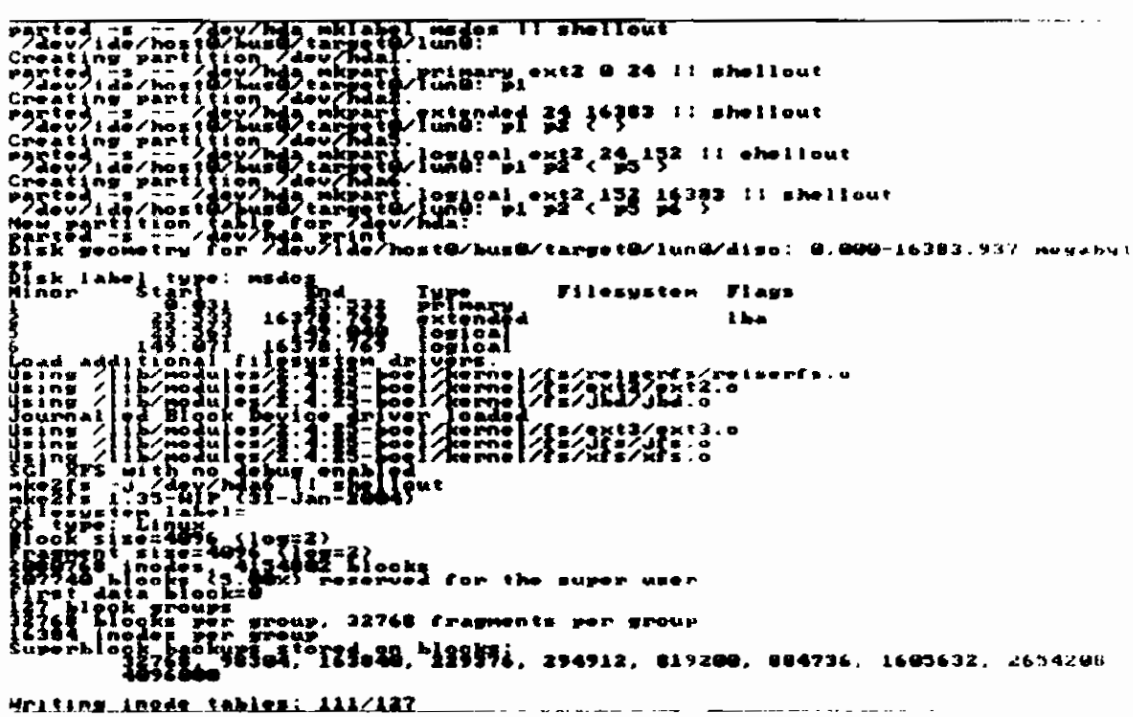


Figura A - 31. Particionamiento del disco duro del nodo cliente

Una vez que se ha formateado el disco duro de los clientes, se procede a descargar e instalar la imagen, como se muestra en la Figura A - 32.

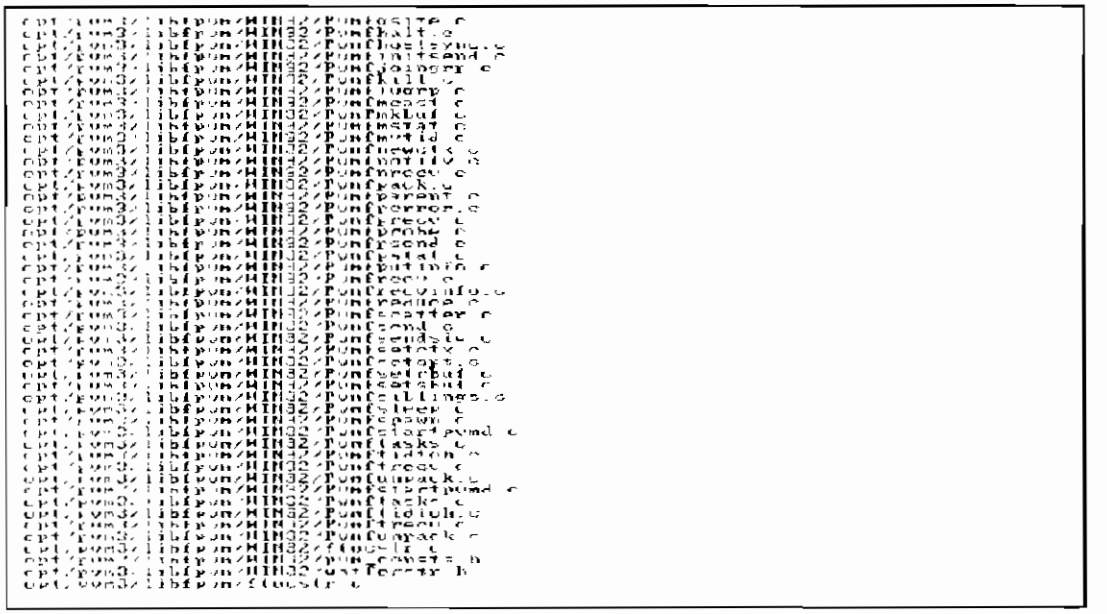


Figura A - 32. Descarga de la imagen SIS

Una vez que se termine la instalación el cliente puede reiniciar o estar pitando constantemente para indicar que ha terminado. En la Figura A - 33 se muestra la salida de un cliente que se encuentra esperando para ser reiniciado de forma automática. Cabe recalcar que si el cliente se configura para reiniciar de forma automática, y el proceso de transferencia de imagen no ha terminado aún (para otros clientes), OSCAR de forma automática volverá a reinstalar este cliente.

Figura A - 33. Finalización de la instalación del nodo cliente

Luego de que en todos los clientes se haya instalado el sistema, se procede a cerrar la ventana **<Setup Networking>** presionando el botón **<Close>** y se procede a reiniciar los nodos. En la Figura A - 34 se muestra la pantalla del gestor de carga de Mandrake Linux 10.0 que tienen todos los nodos.

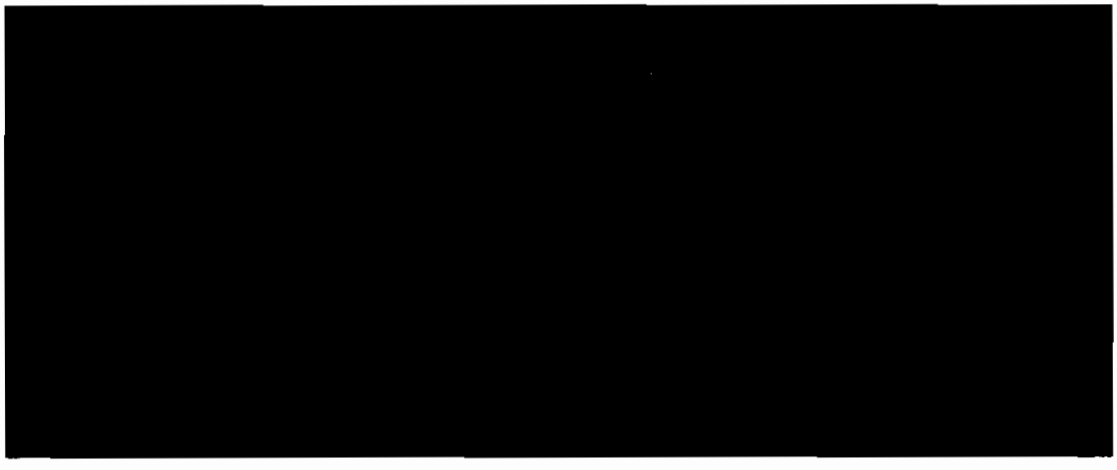


Figura A - 34. Gestor de carga de los nodos cliente

Paso 7 (*Step 7*)

Una vez que todos los nodos cliente hayan iniciado su sistema se procede a presionar el botón **<Complete Cluster Setup>**. El paso 7 ejecuta los *scripts* de configuración final para cada paquete OSCAR y realizará varias funciones de

limpieza y re inicialización. Un dialogo informará el éxito o fracaso de este paso. Luego se debe presionar **<Close>**. En la Figura A - 35 se muestra las salidas que genera la ejecución del Paso 7. En la Figura A - 36 se muestra un diálogo informando el éxito al completar la instalación del *cluster*.

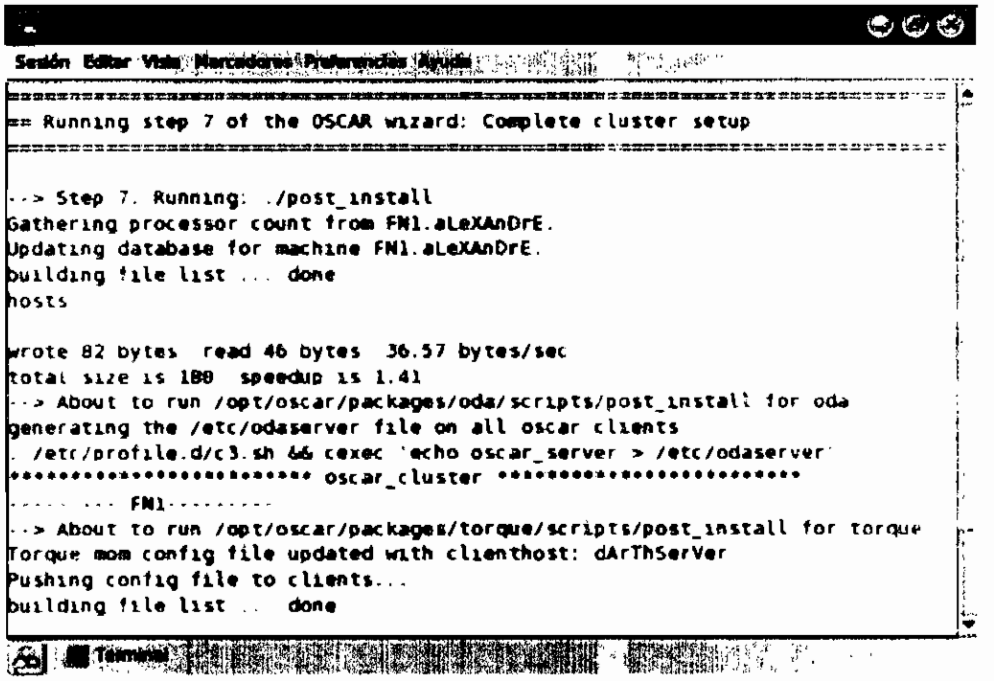


Figura A - 35. Ventana con información sobre la ejecución del paso 7

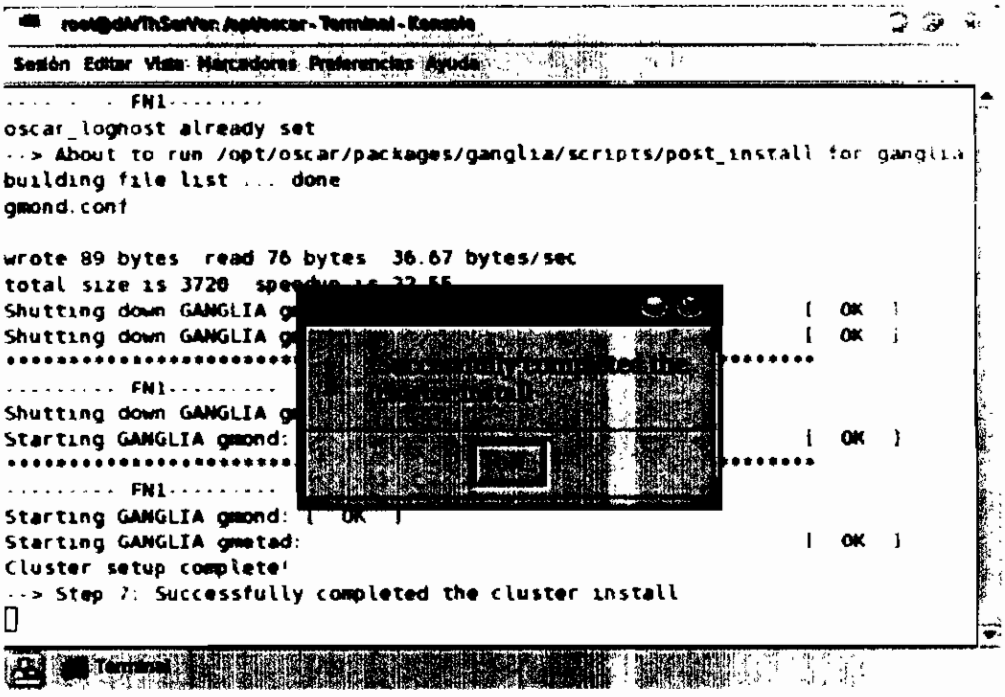


Figura A - 36. Finalización exitosa de la configuración del *cluster*

Paso 8 (*Step 8*)

Se puede realizar un conjunto de pruebas para comprobar que los componentes principales (OpenSSH, PBS, MPI, PVM, etc.) del *cluster* estén correctamente funcionando. Para realizar la prueba se presiona el botón **<Test Cluster Setup>**. El paso 8 abrirá una nueva ventana donde se ejecutarán las pruebas, se realizará la prueba de un programa en particular y se informará su éxito o fracaso. En la Figura A - 37 se muestra la salida de la ejecución del paso 8.

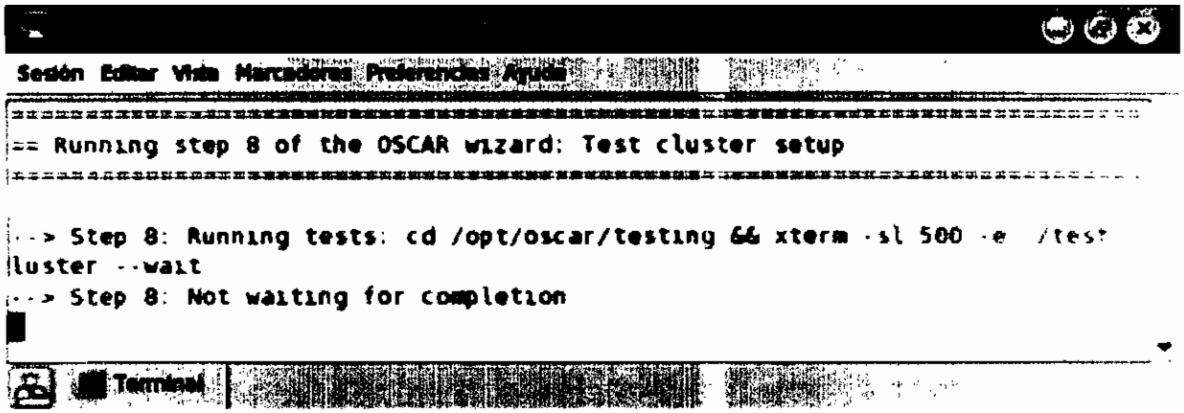


Figura A - 37. Ventana con información sobre la ejecución del paso 8

La nueva ventana con información sobre las pruebas se muestra en la Figura A - 38. Si alguna de las pruebas falla hay un problema con la instalación. Luego se debe presionar **<Close>** para salir del *Wizard*.

Figura A - 38. Pruebas del *cluster*

ANEXO B

INSTALACIÓN DE *Rocks*

SERVIDOR

1. Se inserta el CD **Rocks Base** en el computador *frontend* y se lo resetea.

2. Al reiniciar la máquina, aparecerá una pantalla de bienvenida, como la mostrada en la Figura B - 1, en la cual se debe digitar `frontend`.

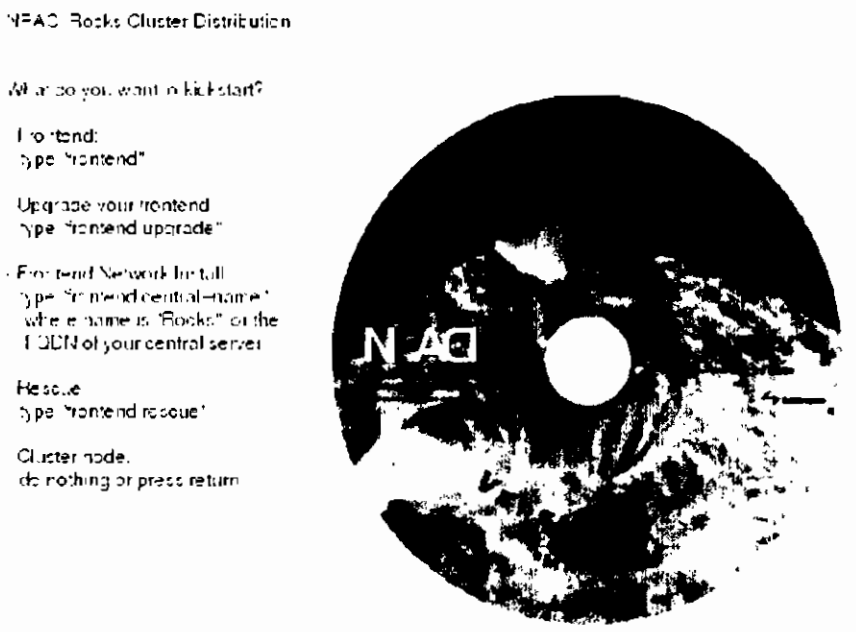


Figura B - 1. Pantalla de bienvenida de *Rocks*

Luego aparecerá una pantalla en la que se indica si se tiene algún otro *Roll*. Está pantalla se muestra en la Figura B - 2.

Figura B - 2. Pantalla para ingresar otros *Rolls*

4. Después de que el CD/DVD sea expulsado, se debe colocar el CD **HPC Roll** en la unidad. Cuando el **Roll HPC** es descubierto, *Rocks* informa de este descubrimiento. Luego *Rocks* vuelve a preguntar por otro *Roll*.

```
Found Roll 'hpc'
```

5. Cuando se han agregado todos los *Roll*, se pide volver a colocar el CD **Rocks Base**.

6. Luego se pide ingresar información sobre el *cluster*: nombre completo del *frontend*, nombre del *cluster*, organización propietaria, localización, estado, país, contacto, URL. Esta información es usada por Ganglia para identificar de manera única al *cluster*. Es importante el especificar el nombre completo del *frontend*; este nombre es escrito en varios archivos tanto en el *frontend* como en los nodos de cómputo; si se cambia el nombre del *frontend*, servicios como: SGE, Globus, NFS, AutoFS y Apache no funcionarán. En la Figura B - 3 se muestra la pantalla que permite ingresar información del *cluster*.

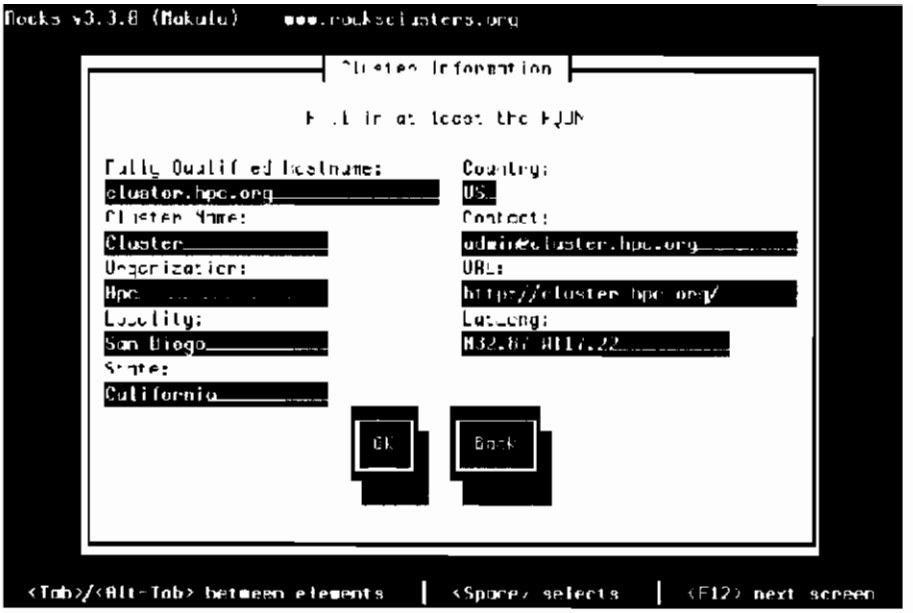


Figura B - 3. Pantalla para ingresar información del *cluster*

7. Luego aparece la pantalla de particionamiento, en la cual se puede seleccionar un particionamiento manual o automático. En la Figura B - 4 se muestra esta pantalla. En la Tabla B - 1 se muestra un ejemplo del particionamiento automático del disco duro.

Nombre de la Partición	Tamaño
/	6 GB
swap	1 GB
/export	El resto del disco

Tabla B - 1. Información para particionamiento automático del disco duro

Si se desea otro tipo de particionamiento se debe escoger `Disk Druid`. Pero no se debe olvidar el crear la partición `/export` por que la instalación fallará. Dependiendo de la versión de *Rocks* que se esté usando, el nombre de la partición `/export` puede cambiar.

Figura B - 4. Pantalla para particionar el disco

8. Luego se muestra la pantalla para configuración del adaptador de la red interna. Esta pantalla se muestra en la Figura B - 5.

Figura B - 5. Pantalla para configuración de la red interna

9. Luego se muestra la pantalla para configurar el adaptador de la red externa. En la Figura B - 6 se muestra esta pantalla.

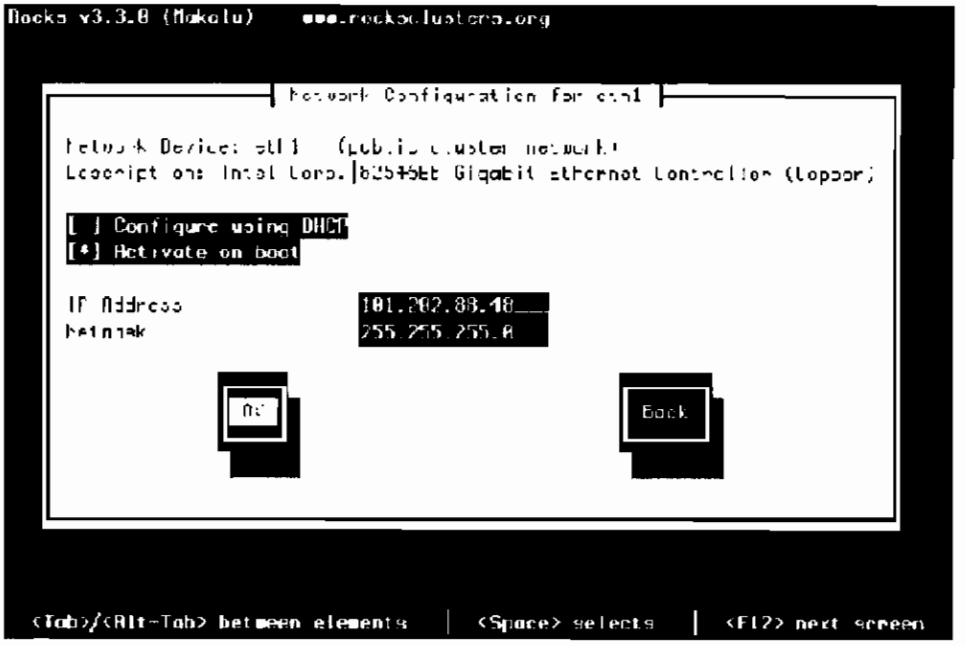


Figura B - 6. Pantalla para configuración de la red externa

10. Luego se puede configurar el *gateway* y el DNS de la red externa. Esta pantalla se muestra en la Figura B - 7.

Figura B - 7. Pantalla para configuración parámetros adicionales de la red

11. Luego se debe configurar la zona de tiempo y el servidor NTP. Esta pantalla se muestra en la Figura B - 8.

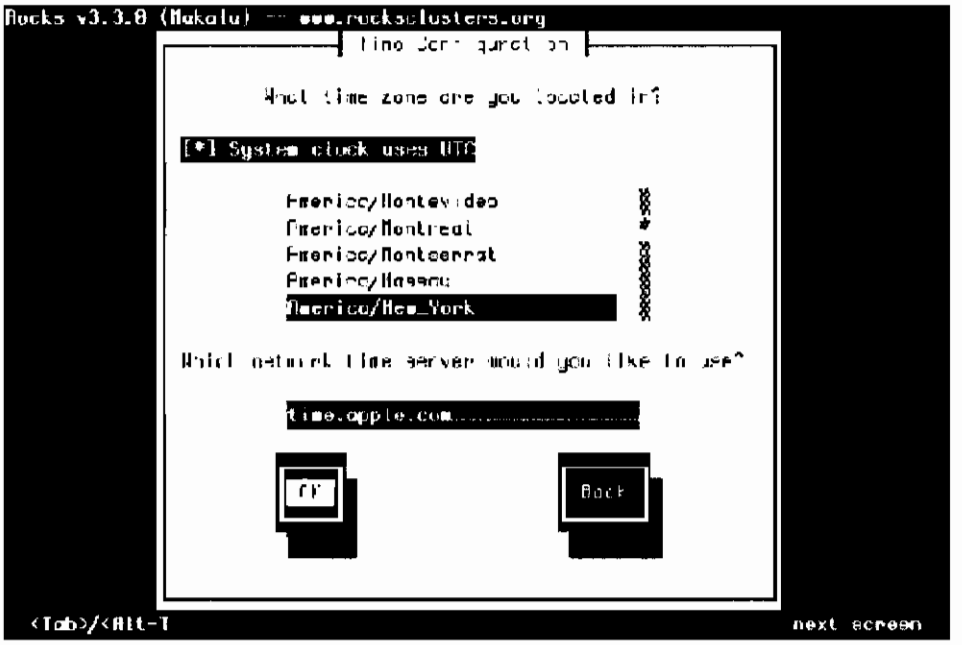


Figura B - 8. Pantalla para configuración de la zona de tiempo y del servidor de tiempo

12. Luego se pide ingresar el *password* del *root*. En la Figura B - 9 se muestra la pantalla que permite ingresar el *password*.

Figura B - 9. Pantalla para configuración del *password* del *root*

13. Después de estos pasos, se procederá a formatear el sistema de archivos. Luego se instalarán los paquetes básicos. En la Figura B - 10 se muestra la pantalla que indica la instalación de los paquetes.

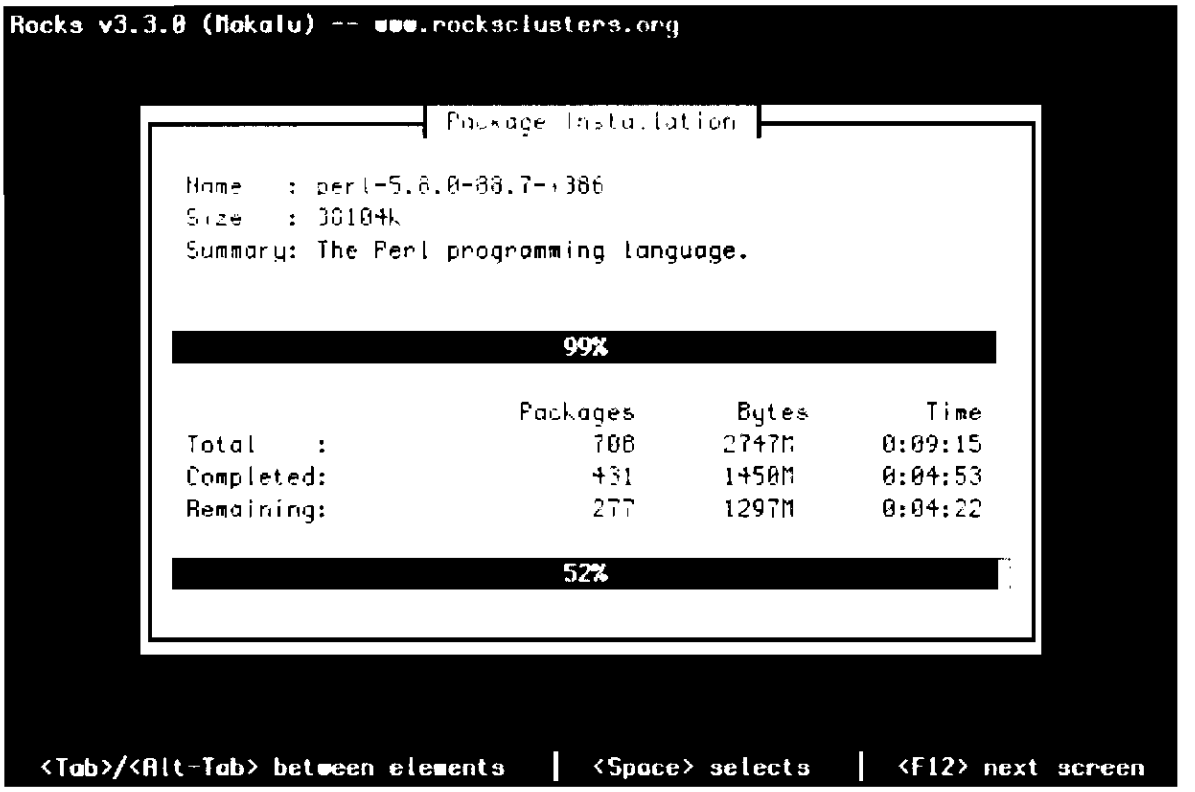


Figura B - 10. Instalación de paquetes

14. Luego el instalador pedirá los CDs o DVDs que contengan los Roll adicionales.

15. Luego de instalar todos los paquetes, el computador reiniciará.

INSTALACIÓN DE LOS NODOS DE CÓMPUTO

1. En el *frontend* se inicia sesión como *root*.

2. Se ejecuta el programa que capture pedidos DHCP y coloque esta información dentro de la base de datos **MySQL** de *Rocks*. En la Figura B - 11 se muestra la pantalla de **insert-ethers**.

   ```
   # insert-ethers
   ```

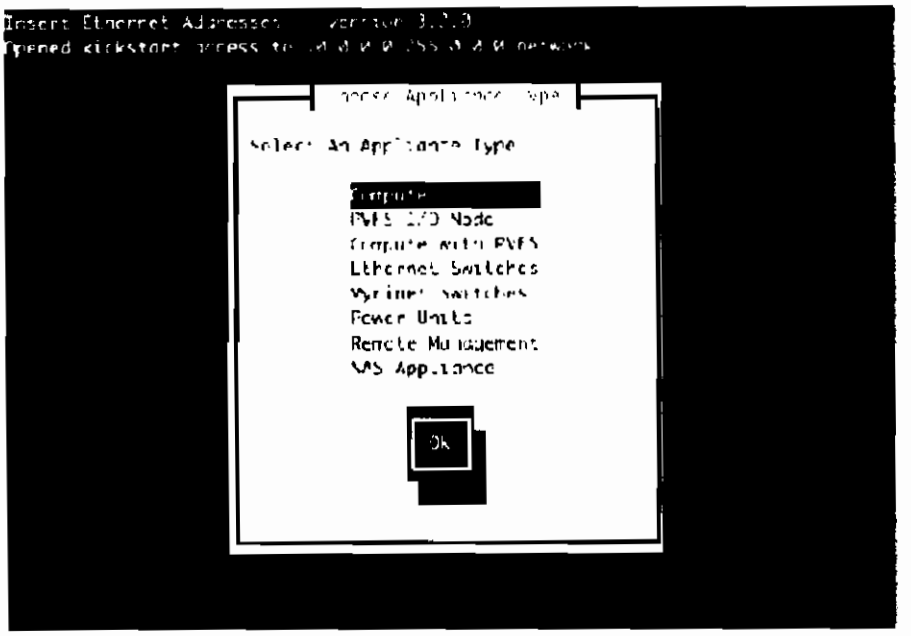


Figura B - 11. Pantalla de insert-ethers

3. Se selecciona la opción *Compute* y se presiona OK. Luego aparecerá una pantalla, como la mostrada en la Figura B - 12, que indica que se está esperando por los nodos de cómputo.

Figura B - 12. Pantalla de indicación de espera por nodos

4. Luego se toma el CD (el mismo con el que se instaló el *frontend*) y se lo coloca en el primer nodo de cómputo. Si no se dispone de una unidad de CD, se puede utilizar PXE (arranque desde la red) o se puede utilizar un disquete.

5. Se enciende el primer nodo de cómputo.

6. Cuando el *frontend* recibe el pedido DHCP desde el nodo de cómputo, se observa una pantalla, como la presentada en la Figura B - 13, que indica que **insert-ethers** recibió una pedido de DHCP, se inserta la información de la base de datos y se actualiza todos los archivos de configuración (`/etc/hosts`, `/etc/dhcpd.conf`, archivos de PBS, Maui y NIS).

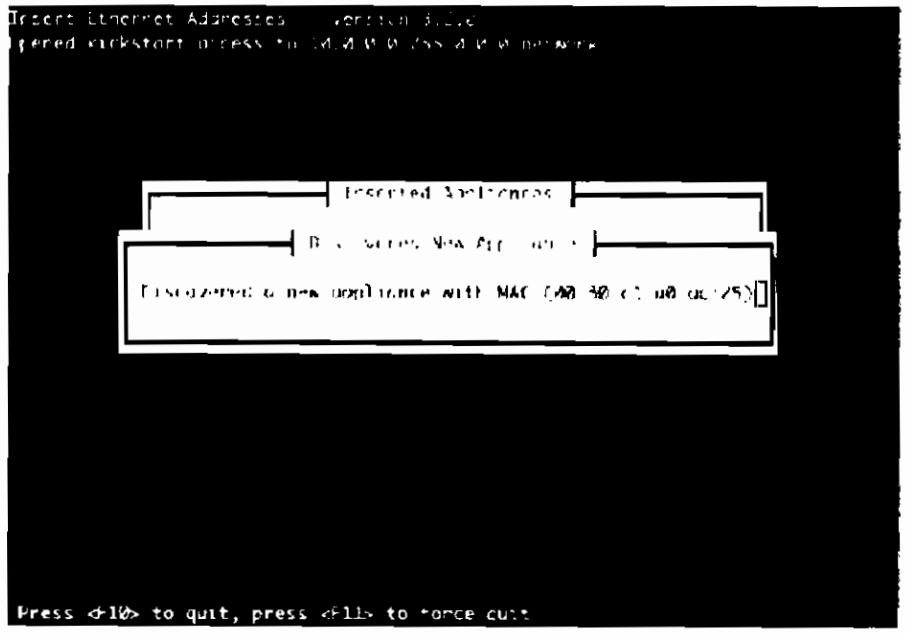


Figura B - 13. Pantalla que informa el descubrimiento de un cliente

En la Figura B - 14 se puede observar que **insert-ethers** descubrió un nodo de cómputo, Se puede apreciar "( )", lo que indica que el nodo aún no ha realizado el pedido del archivo *kickstart*.

Figura B - 14. Pantalla que indica que el cliente aún no obtiene el archivo *kickstart*

En la Figura B - 15 se puede apreciar "(*)", lo que indica que el nodo de cómputo ha recibido el archivo *kickstart*. Los archivos kickstart se envían a través de https. Si existe algún error durante la transmisión del archivo *kickstart*, se mostrará un código de error de HTTP.

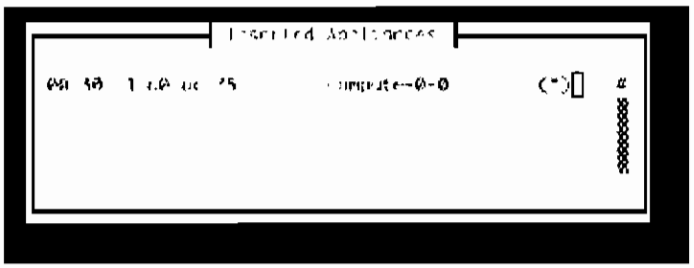


Figura B - 15. Pantalla que indica que el cliente obtuvo el archivo *kickstart*

. . En este punto se puede monitorear la instalación usando **ssh**. El nombre del nodo se puede extraer desde la salida en pantalla de **insert-ethers**. El nombre del nodo es compute-x-z; donde x es el número del gabinete y z es el número del nodo. Por ejemplo compute-0-0 significa que es el nodo 0 del gabinete 0.

```
# ssh compute-0-0 -p 2200
```

8. Cuando se completa la instalación, el CD es expulsado. Se debe tomar el CD y colocarlo en el nodo siguiente.

9. Se puede modificar el número del gabinete, saliendo de **insert-ethers** presionando la tecla F10 y luego ejecutando el comando:

```
# insert-ethers -cabinet=1
```

ANEXO C

COMPILACIÓN DE APLICACIONES A PARTIR DE LAS FUENTES

En muchas ocasiones, no es posible obtener un programa en formato RPM, por lo que es necesario compilarlo desde los archivos de código fuente.

DESCARGA

Lo primero que se debe hacer es localizar y descargar las fuentes que se desean compilar. Es posible encontrar las fuentes contenidas dentro de un archivo con un nombre con la terminación: `.tar.gz`, `.tar.Z`, `.tar.bz2`, o `.tgz`. A estos archivos que contienen archivos fuentes usualmente se los conoce como fuentes binarias (*binary source*) o *tarballs*. También sería recomendable el descargar la información del proceso de instalación.

Es posible que el programa que se está tratando de compilar dependa de la existencia de otros programas que pueden o no estar instalados en el sistema. Si el programa depende de otros programas, se deben instalar estos primero.

DESEMPAQUETAMIENTO

La fase de desempaquetamiento es relativamente fácil.

Si el nombre del archivo termina con `.tar.gz`, `.tar.Z` o `.tgz`, se realiza el desempaquetamiento mediante el comando **tar**, con las opciones `xzvf`; por ejemplo:

```
$ tar xzvf nombre_paquete.tar.gz
```

La opción `x` del comando **tar** permite extraer los archivo contenidos, la opción `z` es para descomprimir los archivos mediante **gzip**, la opción `v` es para imprimir

una lista con los nombres de los archivos que están siendo extraídos, y la opción `f` es para especificar el nombre del archivo sobre el cual se van a realizar las operaciones antes mencionadas.

Si el nombre del archivo termina con `.tar.bz2`, se realiza el desempaquetamiento mediante el comando **tar**, con las opciones `xjvf`; por ejemplo:

```
$ tar xjvf nombre_paquete.tar.bz2
```

La opción `x` del comando **tar** permite extraer los archivo contenidos, la opción `j` es para descomprimir los archivos mediante **bzip2**, la opción `v` es para imprimir una lista con los nombres de los archivos que están siendo extraídos, y la opción `f` es para especificar el nombre del archivo sobre el cual se van a realizar las operaciones antes mencionadas.

Todos los archivos desempaquetados se descomprimen en un directorio que se genera dentro del directorio actual (donde se ejecutó el comando).

INSPECCIÓN DE LOS ARCHIVOS FUENTE

Después de desempaquetar los archivos fuente, se debe ingresar al directorio creado y localizar la documentación relacionada con la instalación. Por lo general, la información de la instalación se encuentra en los archivos `README` o `INSTALL` o puede existir información adicional respecto a un sistema operativo en particular en los archivos `README.plataform` o `INSTALL.plataform`, donde `plataform` es el nombre del sistema operativo, por ejemplo Linux.

CONFIGURACIÓN

Dentro del directorio, donde se descomprimieron los archivos, se encuentra un *script* denominado `configure`. Este *script* está diseñado para configurar las fuentes para su compilación de acuerdo al sistema. Al ejecutar el *script*

`configure`, éste determina el sistema y las características del sistema, creando un archivo denominado `Makefile`, el cual contiene las instrucciones necesarias para construir e instalar las fuentes en el sistema.

Si no se dispone del *script* `configure`, las fuentes deben poseer un archivo `Makefile` estándar, el cual debe estar diseñado para trabajar en una variedad de sistemas.

Para desplegar las opciones de configuración disponibles, para el programa que se desea compilar, se puede ejecutar:

```
$ ./configure --help
```

Para ejecutar el *script* `configure` se debe ejecutar:

```
$ ./configure <options>
```

La mayoría de *scripts* `configure` disponen de la opción `--prefix`, la cual permite especificar la ubicación que tendrá el programa una vez compilado. Por defecto, la mayoría de programas se instalan en el directorio `/usr/local`; por ejemplo, si se desea instalar el programa en el directorio `/opt`, se pude ejecutar:

```
$ ./configure --prefix=/opt
```

El *script* `configure` también permite especificar la ubicación de las páginas **man** mediante la opción `--mandir` y de la información del programa **info**, mediante la opción `--infodir`. Por defecto, la mayoría de programas instalan sus páginas **man** en el directorio `/usr/share/man` y sus páginas de información en el directorio `/usr/share/info`.

INVOCANDO A MAKE

Una vez que el *script* `configure` completa su operación, se procede a compilar los archivos fuente en un programa. Para compilar los archivos fuente se utiliza un

programa denominado **make**. El *script* `configure` genera un archivo denominado `Makefile` o `makefile`, el cual indica a **make** como construir las fuentes e instalar los binarios resultantes, páginas **man** y otros archivos de soporte.

El archivo `Makefile` contiene instrucciones, denominadas reglas, que especifican como construir ciertos objetivos (*targets*).

Para invocar a **make**, se debe ejecutar en el directorio en el que se ejecutó el *script* `configure`:

```
$ make
```

El programa **make**, busca e interpreta el archivo `makefile` o `Makefile` en el directorio actual. Por defecto, la invocación de **make** sin argumentos resulta en la construcción del objetivo por defecto (*default target*). Hay ocasiones en las que el programa que se está tratando de compilar no dispone de un objetivo por defecto, por lo que se debe ejecutar:

```
$ make all
```

La compilación del código fuente puede tardar, dependiendo del programa que se está compilando. Una vez terminada la ejecución de **make**, y presumiendo que no existieron errores, el programa compilado está listo para instalarse en el sistema.

INSTALACIÓN

A pesar de que el programa está compilado, aún no está listo para su uso. Todos los componentes del programa deben ser copiados desde el directorio actual a su correcta ubicación dentro del sistema de archivos. Por ejemplo, todos los binarios deben copiarse a `/usr/bin`, las páginas **man** a `/usr/local/man`.

Para la instalación del software, se requiere los permisos del administrador del sistema (*root*).

Para proceder a instalar los componentes del sistema, se debe ejecutar el comando:

```
# make install
```

Tradicionalmente, el objetivo `install` es utilizado para copiar todos los archivos creados a su ubicación correcta. Adicionalmente, el objetivo `install` asigna los propietarios y permisos apropiados a los archivos instalados.

Una vez que todos los componentes han sido instalados, el programa está listo para ejecutarse. Para iniciar el programa se puede ejecutar:

```
$ nombre_programa
```

ANEXO D

FUNDAMENTOS DE MPI

DECLARACIÓN DE LAS FUNCIONES MÁS UTILIZADAS

- **MPI_Get_count**

```
int MPI_Get_count(
                        MPI_Status* status /* in */,
                        MPI_Datatype datatype /* in */,
                        int* count /* out */
                    )
```

- **MPI_Recv**

```
int MPI_Recv(
                    void* message /* out */,
                    int count /* in */,
                    MPI Datatype datatype /* in */,
                    int source /* in */,
                    int tag /* in */,
                    MPI_Comm comm /* in */,
                    MPI_Status* status /* out */
                )
```

- **MPI_Send**

```
int MPI_ Send(
                        void* message /* in */,
                        int count /* in */,
                        MPI_Datatype datatype /* in */,
                        int dest /* in */,
                        int tag /* in */,
                        MPI_Comm comm /* in */
                    )
```

- **MPI_Buffer_attach**

```
int MPI_Buffer_attach(
                        void* buffer /* in */,
                        int size /* in */
                      )
```

- **MPI_Buffer_detach**

```
int MPI_Buffer_detach(
                        void* buffer address /* out */,
                        int size ptr /* out */
                      )
```

- **MPI_Irecv**

```
int MPI_Irecv(
                void* message /* out */,
                int count /* in */,
                MPI_Datatype datatype /* in */,
                int dest /* in */,
                int tag /* in */,
                MPI_Comm comm /* in */,
                MPI_Request* request /* out */
              )
```

- **MPI_Isend**

```
int MPI_Isend(
                void* message /* in */,
                int count /* in */,
                MPI_Datatype datatype /* in */,
                int dest /* in */,
                int tag /* in */,
                MPI_Comm comm /* in */,
                MPI_Request* request /* out */
              )
```

- **MPI_Test**

```
int MP Test(
                MPI_Request* request /* in/out */,
                int* flag /* out */,
                MPI_Status* status /* out */
            )
```

- **MPI_Wait**

```
int MPI_Wait(
                MPI_Request* request /* in/out */,
                MPI_Status* status /* out */
             )
```

- **MPI_Cancel**

```
int MPI_Cancel(
                     MPI_Request* request /* in */
               )
```

- **MPI_Pack**

```
int MPI_Pack(
                void* inbuf /* in */,
                int incount /* in */,
                MPI_Datatype datatype /* in */,
                void* pack_buf /* out */,
                int pack_buf_size /* in */,
                int* position /* in/out */,
                MPI_Comm comm /* in */
             )
```

- **MPI_Pack_size**

```
int MPI_Pack_size(
                    int incount /* in */,
                    MPI_Datatype datatype /* in */,
                    MPI_Comm comm /* in */,
                    int* size /* out */
                  )
```

- **MPI_Unpack**

```
int MPI_Unpack(
                    void* pack_buf /* in */,
                    int pack_buf_size /* in */,
                    int* position /* in/out */
                    void* outbuf /* out */,
                    int outcount /* in */,
                    MPI_Datatype datatype /* in */,
                    MPI_Comm comm /* in */
               )
```

- **MPI_Barrier**

```
int MPI_Barrier(
                    MPI_Comm comm /* in */
                )
```

- **MPI_Bcast**

```
int MPI_Bcast(
                    void* buffer /* in/out */,
                    int count /* in */,
                    MPI_Datatype datatype /* in */,
                    int root /* in */,
                    MPI_Comm comm /* in */
                )
```

- **MPI_Allgather**

```
int MPI_Allgather(
                    void* sendbuf /* in */,
                    int sendcount /* in */,
                    MPI_Datatype sendtype /* in */,
                    void* recvbuf /* out */,
                    int recvcount /* in */,
                    MPI_Datatype recvtype /* in */,
                    MPI_Comm comm /* in */
                  )
```

- **MPI_Alltoall**

```
int MPI_Alltoall(
                    void* sendbuf /* in */,
                    int sendcount /* in */,
                    MPI_Datatype sendtype /* in */,
                    void* recvbuf /* out */,
                    int recvcount /* in */,
                    MPI_Datatype recvtype /* in */,
                    MPI_Comm comm /* in */
                  )
```

- **MPI_Gather**

```
int MPI_Gather(
                    void* sendbuf /* in */,
                    int sendcount /* in */,
                    MPI_Datatype sendtype /* in */,
                    void* recvbuf /* out */,
                    int recvcounts /* in */,
                    MPI_Datatype recvtype /* in */,
                    MPI_Comm comm /* in */
                )
```

- **MPI_Scatter**

```
int MPI_Scatter(
                    void* sendbuf /* in */,
                    int sendcount /* in */,
                    MPI_Datatype sendtype /* in */,
                    void* recvbuf /* out */,
                    int recvcount /* in */,
                    MPI_Datatype recvtype /* in */,
                    int root /* in */,
                    MPI_Comm comm /* in */
                )
```

- **MPI_Reduce**

```
int MPI_Reduce(
                    void* operand /* in */,
                    void* result /* out */,
                    int count /* in */,
                    MPI_Op operator /* in */,
                    int root /* in */,
                    MPI_Comm comm /* in */
                )
```

- **MPI_Scan**

```
int MPI_Scan(
                void* operand /* in */,
                void* result /* out */,
                int count /* in */,
                MPI_Datatype datatype /* in */,
                MPI_Op operator /* in */,
                MPI_Comm comm /* in */
            )
```

- **MPI_Comm_rank**

```
int MPI_Comm_rank(
                    MPI_ Comm comm /* in */,
                    int* result /* out */
                  )
```

- **MPI_Comm_size**

```
int MPI_ Comm size(
                    MPI_ Comm comm /* in */,
                    int* size /* out */
                   )
```

- **MPI_Get_processor_name**

```
int MPI_Get_processor_name(
                              char* name /* out */,
                              int* resultlen /* out */
                           )
```

- **MPI_Wtick**

```
double MPI_Wtick(void)
```

- **MPI_Wtime**

```
double MPI_Wtime(void)
```

- **MPI_ Abort**

```
double MPI_Abort(
                    MPI_comm comm /* in */,
                    int error_code /* in */
                  )
```

- **MPI_Finalize**

```
int MPI_Finalize(void)
```

- **MPI_Init**

```
int MPI_Init(
                int* argc_ptr /* in/out */,
                char** argv_ptr[ ] /* in/out */
              )
```

CONSTANTES Y DEFINICIONES DE MPI MÁS UTILIZADAS

- **Convenciones de error**

```
MPI_SUCCESS             MPI_ERR_TOPOLOGY
MPI_ERR_BUFFER          MPI_ERR_DIMS
MPI_ERR_COUNT           MPI_ERR_ARG
MPI_ERR_TYPE            MPI_ERR_UNKNOWN
MPI_ERR_TAG             MPI_ERR_TRUNCATE
MPI_ERR_COMM            MPI_ERR_OTHER
MPI_ERR_RANK            MPI_ERR_INTERN
MPI_ERR_REQUEST         MPI_PENDING
MPI_ERR_ROOT            MPI_ERR_IN_STATUS
MPI_ERR_GROUP           MPI_ERR_LASTCODE
MPI_ERR_OP
```

- **Operaciones de reducción**

```
MPI_MAX
MPI_MIN
MPI_SUM
MPI_PROD
MPI_MAXLOC
MPI_MINLOC
MPI_BAND
MPI_BOR
MPI_BXOR
MPI_LAND
MPI_LOR
MPI_LXOR
```

- **Constantes**

```
MPI_BOTTOM
MPI_PROC_NULL
MPI_ANY_SOURCE
MPI_ANY_TAG
MPI_UNDEFINED
MPI_BSEND_OVERHEAD
MPI_KEYVAL_INVALID
```

ANEXO E

APLICACIÓN Y MANEJO DE LIBRERÍA MPI EN WINDOWS

Se utilizaron los lenguajes C y C++, para desarrollar la aplicación junto con MPICH Versión 1.2.6 y MPICH2 Versión 1.0.2, las cuales son las implementaciones de las librerías de la especificación MPI Versión 1 y Versión 2 respectivamente.

En la máquina de desarrollo se utilizó una partición de 20 GB de tamaño en disco, sobre la cual se instaló Windows XP Profesional con Service Pack 2. En dicho sistema operativo se instaló y configuró MPICH2 Versión 1.0.2, la cual incluye las llamadas y convenciones de la Versión 1 y 2 del estándar MPI, y tiene fácil integración con el entorno IDE de desarrollo Microsoft Visual Studio Enterprise For Architects 2003, y se utilizó Microsoft Visual C++ .NET, este último permitió la escritura del código secuencial y paralelo.

La aplicación se ejecutó sobre un sólo computador y la creación de procesos se logró mediante herramientas de MPICH2. MPICH2 para sistemas operativos Windows incluye herramientas que permiten la ejecución de una aplicación paralela en uno o más procesos mediante la utilización de línea de comandos de Windows y la utilización del comando **mpiexec** o una interfaz gráfica que permita seleccionar el ejecutable y el número de procesos, la cual se llama mediante la ejecución de la aplicación **wmpiexec**.

En la Figura E - 1 se muestra la herramienta gráfica para ejecución de tareas paralelas.

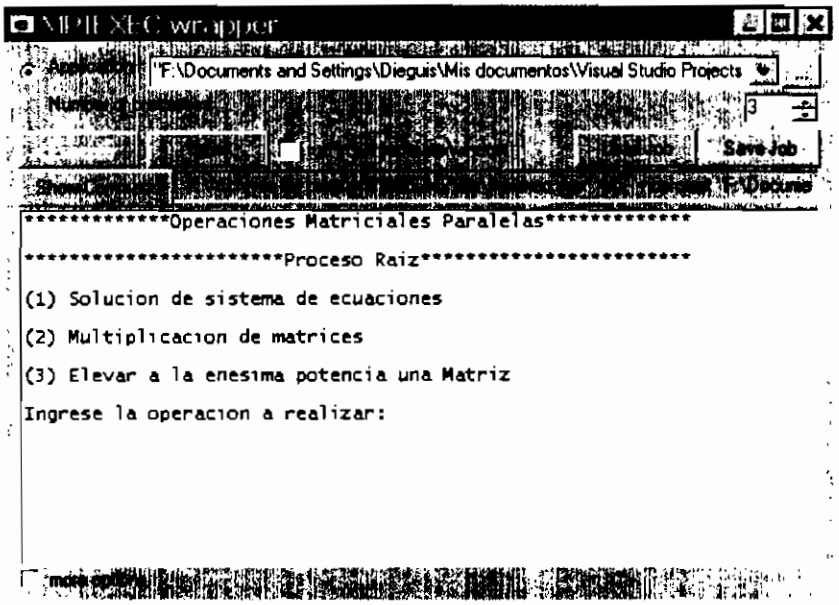


Figura E - 1. Herramienta de ejecución de MPICH2 - WMPIExec

Para incluir a MPICH2 en un proyecto de Visual C++ .NET se añadió las librerías de MPI **mpid.lib**, **mpich2.lib**, y **cxxd.lib**. En la Figura E - 2 se muestra el dialogo de propiedades del proyecto donde se añadieron las librerías.

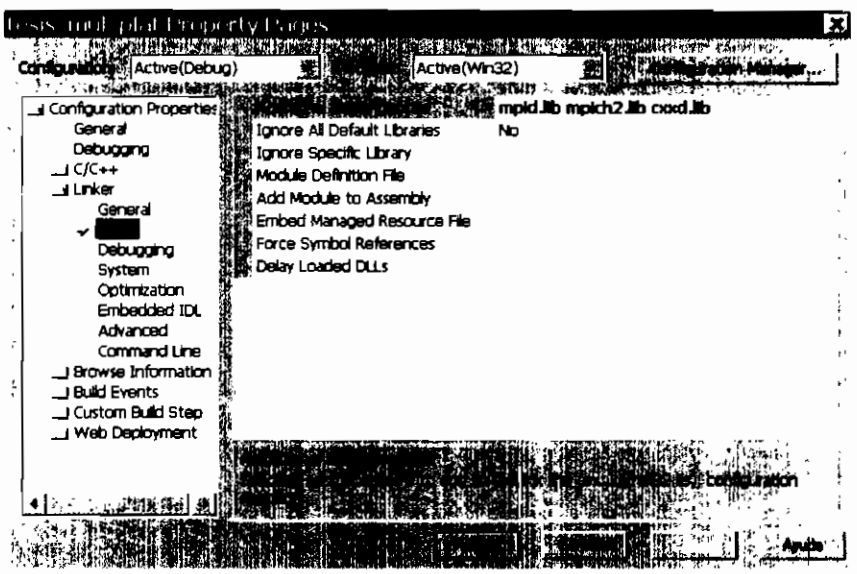


Figura E - 2. Librerías MPI en proyectos de Visual C++ .NET

ANEXO F

ARCHIVOS DE CONFIGURACIÓN DE OSCAR

ARCHIVO:
/var/lib/systemimager/image/oscarimage/etc/systemimager/autoinstallscript.conf

```
<!--
        This autoinstallscript.conf file was generated by SystemInstaller
        for use by SystemImager when creating the autoinstall script.
        This file generated at: 2005-8-25 13:41:34
        from: /opt/oscar/scripts//opt/oscar/oscarsamples/sample.disk.scsi
        Image directory: /var/lib/systemimager/images/oscarimage
-->
<config>
        <disk dev="/dev/sdb" label_type="msdos" unit_of_measurement="MB">
                <part num="1" size="24" p_type="primary" />
                <part num="2" size="*" p_type="extended" />
                <part num="5" size="128" p_type="logical" />
                <part num="6" size="*" p_type="logical" />
        </disk>
        <fsinfo line="100" real_dev="/dev/sdb6" mp="/" fs="ext3" options="defaults" dump="1" pass="2" />
        <fsinfo line="101" real_dev="/dev/sdb5" mp="swap" fs="swap" options="defaults" dump="0" pass="0" />
        <fsinfo line="102" real_dev="/dev/sdb1" mp="/boot" fs="ext3" options="defaults" dump="1" pass="2" />
        <fsinfo line="103" real_dev="/dev/fd0" mp="/mnt/floppy" fs="auto" options="noauto,owner" dump="0" pass="0" />
        <fsinfo line="104" real_dev="none" mp="/dev/pts" fs="devpts" options="defaults" dump="0" pass="0" />
        <fsinfo line="105" real_dev="none" mp="/proc" fs="proc" options="defaults" dump="0" pass="0" />
        <fsinfo line="106" real_dev="nfs_oscar:/home" mp="/home" fs="nfs" options="rw" dump="0" pass="0" />
        <boel devstyle="devfs" />
</config>
```

Nota 1: En color azul se encuentran las modificaciones realizadas en este archivo de configuración según se describe en la Sección 5.2.2.3.

Nota 2: En color rojo se encuentran las modificaciones realizadas en este archivo de configuración según lo descrito en la Sección 5.2.2.2.

ARCHIVO:
/var/lib/systemimager/image/oscarimage/etc/systemconfig/systemconfig.conf

```
# systemconfig.conf written by systeminstaller.
CONFIGBOOT = YES
CONFIGRD = YES

[BOOT]
      ROOTDEV = /dev/sdb6
      BOOTDEV = /dev/sdb
      DEFAULTBOOT = 2.6.9-1.667

[KERNEL0]
      PATH = /boot/vmlinuz-2.6.9-1.667smp
      LABEL = 2.6.9-1.667smp

[KERNEL1]
      PATH = /boot/vmlinuz-2.6.9-1.667
      LABEL = 2.6.9-1.667
```

Nota 1: En color azul se encuentran las modificaciones realizadas en este archivo de configuración según se describe en la Sección 5.2.2.3.

ARCHIVO:
/var/lib/systemimager/scripts/oscarimage.master

```
#!/bin/sh

#      "SystemImager"
#
#      Copyright (C) 1999-2004 Brian Elliott Finley
#
#      $Id: autoinstallscript.template,v 1.17 2004/04/17 14:47:53
#      brianfinley Exp $
#

# This master autoinstall script was created with SystemImager v3.3.2

# Load functions and other variables
. /etc/init.d/functions

get_arch

if [ -z $NO_LISTING ]; then
    VERBOSE_OPT="v"
else
    VERBOSE_OPT=""
fi

# Pull in variables left behind by the linuxrc script.
# This information is passed from the linuxrc script on the autoinstall
# media via /tmp/variables.txt.  Apparently the shell we use in BOEL is
# not intelligent enough to take a "set -a" parameter.
#

. /tmp/variables.txt || shellout

[ -z $IMAGENAME ] && IMAGENAME=oscarimage
[ -z $OVERRIDES ] && OVERRIDES="oscarimage"

### BEGIN Check to be sure this not run from a working machine ###
# Test for mounted SCSI or IDE disks
mount | grep [hs]d[a-z][1-9] > /dev/null 2>&1
[ $? -eq 0 ] &&  echo Sorry.  Must not run on a working machine... &&
shellout

# Test for mounted software RAID devices
mount | grep md[0-9] > /dev/null 2>&1
[ $? -eq 0 ] &&  echo Sorry.  Must not run on a working machine... &&
shellout

# Test for mounted hardware RAID disks
mount | grep c[0-9]+d[0-9]+p > /dev/null 2>&1
[ $? -eq 0 ] &&  echo Sorry.  Must not run on a working machine... &&
shellout
### END Check to be sure this not run from a working machine ###
```

```
##############################################################################
#
#   Stop RAID devices before partitioning begins
#
# Q1) Why did they get started in the first place?
# A1) So we can pull a local.cfg file off a root mounted software RAID
#     system. They may not be started on your system -- they would only
#     be started if you did the stuff in Q3 below.
#
# Q2) Why didn't my local.cfg on my root mounted software RAID work for
#     me with the standard kernel flavour?
# A2) The standard kernel flavour uses modules for the software RAID
#     drivers --therefore, software RAID is not available at the point in
#     the boot process where BOEL needs to read the local.cfg file.  They
#     are only pulled over when this script is run, which is, of course,
#     only runnable if it was pulled over the network using the settings
#     that you would have wanted it to get from the local.cfg file, which
#     it couldn't.  Right?
#
# Q3) Whatever.  So how do I make it work with a local.cfg file on my
#     root mounted software RAID?
# A3) Compile an autoinstall kernel with software RAID, and any other
#     drivers you might need built in (filesystem, SCSI drivers, etc.).
#

if [ -f /proc/mdstat ]; then
  RAID_DEVICES=` cat /proc/mdstat | grep ^md | sed 's/ .*$//g' `

  # raidstop will not run unless a raidtab file exists
  echo "" >> /etc/raidtab || shellout

  # turn dem pesky raid devices off!
  for RAID_DEVICE in ${RAID_DEVICES}
  do
    DEV="/dev/${RAID_DEVICE}"
    # we don't do a shellout here because, well I forgot why, but we don't.
    echo "raidstop ${DEV}" && raidstop ${DEV}
  done
fi

##############################################################################


##############################################################################
# BEGIN disk enumeration
# David N. Lombard, david.n.lombard@intel.com
#
# Note the kludgey way to get /dev/sd* and /dev/*/c*d* to sort properly...
#

echo DISKORDER=${DISKORDER=sd,cciss,ida,rd,hd}
[ -z $DISKORDER ] || {
  echo enumerate_disks
  order=`echo "$DISKORDER" | sed 's/ /,/g' | sed s/,,*/,/g | sed s/^,//`
  DISKS=0
  while : ; do
    [ -z $order ] && break
    type=`expr $order : '\([^,]*\),' \| $order`
    case $type in
    cciss | ida | rd )
      rest=
      [ $type = rd ] && rest="8 9 10 11 12 13 14 15"
```

```
      for c in 0 1 2 3 4 5 6 7 $rest ; do
        for dev in `ls /dev/$type/c${c}d[0-9] 2>/dev/null` ; do
          echo " $dev"
          eval DISK$DISKS=${dev}
          DISKS=`expr $DISKS + 1`
        done
        for dev in `ls /dev/$type/c${c}d[0-9][0-9] 2>/dev/null` ; do
          echo " $dev"
          eval DISK$DISKS=${dev}
          DISKS=`expr $DISKS + 1`
        done
      done
      ;;
    hd )
      for dev in `ls /dev/$type[a-z] 2>/dev/null` ; do
        echo " $dev"
        eval DISK$DISKS=$dev
        DISKS=`expr $DISKS + 1`
      done
      ;;
    sd )
      for dev in `ls /dev/$type[a-z] 2>/dev/null` ; do
        echo " $dev"
        eval DISK$DISKS=$dev
        DISKS=`expr $DISKS + 1`
      done
      for dev in `ls /dev/$type[a-z][a-z] 2>/dev/null` ; do
        echo " $dev"
        eval DISK$DISKS=$dev
        DISKS=`expr $DISKS + 1`
      done
      ;;
    * )
      echo "type='$type'"
      shellout
      ;;
    esac
    order=`expr $order : '[^,]*,\(.*\)'`
  done
  echo DISKS=$DISKS
  [ $DISKS -eq 0 ] && {
    beep
    beep
    echo ""
    echo "NO DISK DEVICE FILES WERE FOUND. THIS USUALLY MEANS THE KERNEL"
    echo "DID NOT RECOGNIZE ANY OF THE ATTACHED DISKS."
    echo ""
    echo "The kernel boot messages, which preceded this, may indicate why"
    echo ""
    echo "Reverting to disk configuration specified by image master script"
    DISKORDER=
    echo ""
  }
  echo
  beep
}
# END disk enumeration
##############################################################################
```

```
# Se comentan las siguientes líneas para permitir que se use el disco sdb
# en lugar del disco sda.
#if [ -z $DISKORDER ] ; then
  DISK0=/dev/sdb
#elif [ -z $DISK0 ] ; then
#  echo "Undefined: DISK0"
#  shellout
#fi

### BEGIN partition $DISK0 ###
echo "Partitioning $DISK0..."
echo "Old partition table for $DISK0:"
parted -s -- $DISK0 print

# Create disk label.  This ensures that all remnants of the old label,
# whatever type it was, are removed and that we're starting with a clean
# label.
echo "parted -s -- $DISK0 mklabel msdos || shellout"
parted -s -- $DISK0 mklabel msdos || shellout

# Get the size of the destination disk so that we can make the partitions
# fit properly.
DISK_SIZE=`parted -s $DISK0 print | grep 'Disk geometry for' | sed
's/^.*-//g' | sed 's/\..*$//' `

[ -z $DISK_SIZE ] && shellout
if [ "$ARCH" = "alpha" ]; then
    END_OF_LAST_PRIMARY=1
else
    END_OF_LAST_PRIMARY=0
fi

echo "Creating partition ${DISK0}1."
START_MB=$END_OF_LAST_PRIMARY
END_MB=$(echo "scale=3; ($START_MB + 24)" | bc)
echo "parted -s -- $DISK0 mkpart primary ext2 $START_MB $END_MB || shellout"
parted -s -- $DISK0 mkpart primary ext2 $START_MB $END_MB || shellout
END_OF_LAST_PRIMARY=$END_MB

echo "Creating partition ${DISK0}2."
START_MB=$END_OF_LAST_PRIMARY
END_MB=$(( $DISK_SIZE - 0 ))
echo "parted -s -- $DISK0 mkpart extended $START_MB $END_MB || shellout"
parted -s -- $DISK0 mkpart extended $START_MB $END_MB || shellout
END_OF_LAST_PRIMARY=$END_MB
END_OF_LAST_LOGICAL=$START_MB

echo "Creating partition ${DISK0}5."
START_MB=$END_OF_LAST_LOGICAL
END_MB=$(echo "scale=3; ($START_MB + 128)" | bc)
echo "parted -s -- $DISK0 mkpart logical ext2 $START_MB $END_MB || shellout"
parted -s -- $DISK0 mkpart logical ext2 $START_MB $END_MB || shellout
END_OF_LAST_LOGICAL=$END_MB

echo "Creating partition ${DISK0}6."
START_MB=$END_OF_LAST_LOGICAL
END_MB=$(( $DISK_SIZE - 0 ))
echo "parted -s -- $DISK0 mkpart logical ext2 $START_MB $END_MB || shellout"
parted -s -- $DISK0 mkpart logical ext2 $START_MB $END_MB || shellout
END_OF_LAST_LOGICAL=$END_MB
```

```
echo "New partition table for $DISK0:"
echo "parted -s -- $DISK0 print"
parted -s -- $DISK0 print
### END partition $DISK0 ###


echo "Load additional filesystem drivers."
# El kernel tiene compilados los controladores para los diferentes
# sistemas de archivos, por lo que modprobe falla si se lo ejecuta.
# modprobe reiserfs
# modprobe ext2
# modprobe ext3
# modprobe jfs
# modprobe xfs

### BEGIN swap and filesystem creation commands ###
echo "mke2fs -j ${DISK0}6 || shellout"
mke2fs -j ${DISK0}6 || shellout
echo "mkdir -p /a/ || shellout"
mkdir -p /a/ || shellout
echo "mount ${DISK0}6 /a/ -t ext3 -o defaults || shellout"
mount ${DISK0}6 /a/ -t ext3 -o defaults || shellout

echo "mkswap -v1 ${DISK0}5 || shellout"
# El kernel no dispone del comando mkswap, por lo que falla si se lo
# ejecuta.
# mkswap -v1 ${DISK0}5 || shellout
echo "swapon ${DISK0}5 || shellout"
# Debido a que no se formatea esta partición con soporte para swap, el
# tratar de montar la partición swap mediante el comando swapon fallará.
# swapon ${DISK0}5 || shellout

echo "mke2fs -j ${DISK0}1 || shellout"
mke2fs -j ${DISK0}1 || shellout
echo "mkdir -p /a/boot || shellout"
mkdir -p /a/boot || shellout
echo "mount ${DISK0}1 /a/boot -t ext3 -o defaults || shellout"
mount ${DISK0}1 /a/boot -t ext3 -o defaults || shellout

### END swap and filesystem creation commands ###


### BEGIN mount proc in image for tools like System Configurator ###

echo "mkdir -p /a/proc || shellout"
mkdir -p /a/proc || shellout
echo "mount proc /a/proc -t proc -o defaults || shellout"
mount proc /a/proc -t proc -o defaults || shellout

### END mount proc in image for tools like System Configurator ###
```

```
#############################################################################
#
#    Lay the image down on the freshly formatted disk(s)
#
if [ ! -z $FLAMETHROWER_DIRECTORY_PORTBASE ]; then
    # Use multicast
    MODULE_NAME="${IMAGENAME}"
    DIR=/a
    RETRY=7
    flamethrower_client
else
    # Use rsync
    if [ $NO_LISTING ]; then
        echo "Quietly installing image... "
        start_spinner
    fi
    if [ "${TMPFS_STAGING}" = "yes" ]; then

        # Deposit image into tmpfs
        DIR=/tmp/tmpfs_staging
        echo
        echo "TMPFS_STAGING=${TMPFS_STAGING} -- Staging in ${DIR}"
        mkdir -p ${DIR}

        echo "rsync -aHS${VERBOSE_OPT} --exclude=lost+found/
             --numeric-ids ${IMAGESERVER}::${IMAGENAME}/ ${DIR}/"
        rsync -aHS${VERBOSE_OPT} --exclude=lost+found/ --numeric-ids
              ${IMAGESERVER}::${IMAGENAME}/ ${DIR}/ || shellout

        # Move from staging in tmpfs to disk
        rsync -aHS${VERBOSE_OPT} --exclude=lost+found/ --numeric-ids
               ${DIR}/ /a/
    else
        echo "rsync -aHS${VERBOSE_OPT} --exclude=lost+found/
              --numeric-ids ${IMAGESERVER}::${IMAGENAME}/ /a/"
        rsync -aHS${VERBOSE_OPT} --exclude=lost+found/ --numeric-ids
              ${IMAGESERVER}::${IMAGENAME}/ /a/ || shellout
    fi
fi

beep

#
#############################################################################


if [ $NO_LISTING ]; then
    stop_spinner
fi

# Leave notice of which image is installed on the client
echo $IMAGENAME > /a/etc/systemimager/IMAGE_LAST_SYNCED_TO || shellout
```

```
### BEGIN generate new fstab file from autoinstallscript.conf ###
cat <<'EOF' > /a/etc/fstab
/dev/sdb6   /      ext3  defaults     1      2
/dev/sdb5   swap   swap  defaults     0      0
/dev/sdb1   /boot  ext3  defaults     1      2
/dev/fd0    /mnt/floppy auto  noauto,owner      0      0
none  /dev/pts    devpts      defaults     0      0
none  /proc proc  defaults     0      0
nfs_oscar:/home   /home nfs   rw     0      0
EOF
### END generate new fstab file from autoinstallscript.conf ###


################################################################################
#
#    Process override directories
#
for OVERRIDE in $OVERRIDES
do
    if [ ! -z $FLAMETHROWER_DIRECTORY_PORTBASE ]; then
        # Use multicast
        MODULE_NAME="override_${OVERRIDE}"
        DIR=/a
        RETRY=7
        flamethrower_client
    else
        # Use rsync
        echo "rsync -av --numeric-ids $IMAGESERVER::overrides/$OVERRIDE/ /a/"
        rsync -av --numeric-ids $IMAGESERVER::overrides/$OVERRIDE/ /a/
              || echo "Override directory $OVERRIDE doesn't seem to
              exist, but that may be OK."
    fi
done

beep
#
################################################################################

################################################################################
# BEGIN disk edits
# David N. Lombard, david.n.lombard@intel.com
#
[ -z $DISKORDER ] || {
  echo "Editing files for actual disk configuration..."
  echo " /dev/sdb -> $DISK0"
  for file in /etc/fstab /etc/raidtab etc/systemconfig/systemconfig.conf; do
    [ -f /a/$file ] || continue
    echo " $file"
    cp /a/$file /a/$file.image
    cat /a/$file.image |
    sed s:/dev/sdb:%DISK0%:g |
    sed s:%DISK0%:$DISK0:g |
    cat > /a/$file
  done
  echo
  beep
}
#
# END disk edits
################################################################################
```

```
#########################################################################
#
# Uncomment the line below to leave your hostname blank.
# Certain distributions use this as an indication to take on the
# hostname provided by a DHCP server.  The default is to have
# SystemConfigurator assign your clients the hostname that
# corresponds to the IP address the use during the install.
# (If you used to use the static_dhcp option, this is your man.)
#
#HOSTNAME=""


#########################################################################
#
# mount /dev /a/dev -o bind if needed
#
echo "mount /dev /a/dev -o bind || shellout"
mount /dev /a/dev -o bind || shellout
#
#########################################################################


#########################################################################
#
#   System Configurator
#
# Configure the client's hardware, network interface, and boot loader.
#
chroot /a/ systemconfigurator --
excludesto=/etc/systemimager/systemconfig.local.exclude --configsi --
stdin << EOL || shellout
[NETWORK]
HOSTNAME = $HOSTNAME
DOMAINNAME = $DOMAINNAME

[INTERFACE0]
DEVICE = eth0
TYPE = dhcp
EOL
#
#########################################################################


#########################################################################
#
#   Post Install Scripts
#
run_post_install_scripts
#
#########################################################################


#########################################################################
#
#   Unmount filesystems
#
echo "umount /a/proc || shellout"
umount /a/proc || shellout

echo "umount /a/dev || shellout"
```

```
umount /a/dev || shellout

echo "umount /a/boot || shellout"
umount /a/boot || shellout

echo "umount /a/ || shellout"
umount /a/ || shellout
#
#############################################################################

#############################################################################
#
#   Tell the image server we're done
#
rsync $IMAGESERVER::scripts/imaging_complete > /dev/null 2>&1
#
#############################################################################

# Take network interface down
ifconfig eth0 down || shellout

# Announce completion (even for non beep-incessantly --post-install
options)
beep 3
beep 3
beep 3
beep 3
beep 3
beep 3
beep 3

# reboot the autoinstall client
# shutdown -r now
```

Nota 1: En color azul se encuentran las modificaciones realizadas en este archivo de configuración según se describe en la Sección 5.2.2.3.

ANEXO G

COSTOS

	Ítem	Cant.	Costo (USD)	Costo Total (USD)
Hardware	Computadores[1]	3	$1.000	$ 3.000
	Switch	1	$ 140	$ 140
	Switch KVM	1	$ 100	$ 100
	Materiales de Cableado Estructurado[2]	1	$ 150	$ 150
	Total Hardware			**$ 3390**
Otros	Cableado Estructurado (Diseño y construcción)	10 puntos	$ 25	$ 250
	Diseño y Construcción del *cluster*	640 horas	$ 5	$3200
	Total Otros			**$ 3450**
	Total Construcción e Implementación del *cluster*			**$ 6840**

[1] Las características de los computadores utilizados se presenta en la Sección 5.2.1.1.

[2] Incluye canaletas y accesorios, cables cat 5e, cables AWG 10, tomas de fuerza, *rack* de 5 HU, *patch* panel, conectores RJ45, bandejas para *rack*.