UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Administración del cluster tipo Beowulf de la Facultad de Ingeniería

TESIS

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A N

Roy Balderas Jiménez

Jeanette Moreno Sánchez

Martín Placido Mojica

DIRECTORA DE TESIS

Lic. Laura Sandoval Montaño

Ciudad Universitaria, Cd. Mx., 2003

ADMINISTRACIÓN DEL CLUSTER TIPO BEOWULF DE LA FACULTAD DE INGENIERÍA

By

Roy Balderas Jiménez
Jeanette Moreno Sánchez
Martín Placido Mojica

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Mathematic

Approved by the
Examining Committee:

_____
Galileo, Thesis Adviser

_____
Carreón Granados, Member

_____
Copernicus, Member

_____
Aristotle, Member

Rensselaer Polytechnic Institute
Troy, New York

Noviembre 2003
(For Graduation May 1685)

ADMINISTRACIÓN DEL CLUSTER TIPO BEOWULF DE LA FACULTAD DE INGENIERÍA

By

Roy Balderas Jiménez
Jeanette Moreno Sánchez
Martín Placido Mojica

An Abstract of a Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Mathematic

The original of the complete thesis is on file
in the Rensselaer Polytechnic Institute Library

Examining Committee:

Galileo, Thesis Adviser
Carreón Granados, Member
Copernicus, Member
Aristotle, Member

Rensselaer Polytechnic Institute
Troy, New York

Noviembre 2003
(For Graduation May 1685)

Índice general

Índice de cuadros

Índice de figuras

Prefacio

Históricamente, la computación de alto desempeño ha sido un área muy especializada de la industria del cómputo, que ha servido a las necesidades computacionales de la ciencia, la ingeniería y la defensa. Sin embargo en años recientes se observa una importante transición que pasa de aplicaciones que necesitaban de cómputo exhaustivo a aplicaciones que necesitan también de comunicaciones exhaustivas.

El paralelismo es ciertamente un concepto bien conocido desde los primeros años de desarrollo de la computación y hoy en día comprende una amplia variedad de técnicas, usadas todas ellas en las áreas de las ciencias de la computación. Se ha venido presentando de diversas formas a lo largo de los años pero siempre con el objetivo de acelerar los tiempos de ejecución de los algoritmos.

Los pioneros en esta área desarrollaron los fundamentos del paralelismo en los primeros años de la década de los 70's, pero durante la mitad de los 80's tuvo lugar una expansión en esta disciplina, justo después de que apareció la primera máquina paralela de propósito general en el mercado.

El rápido desarrollo del procesamiento en paralelo se vio alentado por las exigencias crecientes de sus usuarios. En los 70's y 80's, el alto desempeño se basó en arquitecturas y tecnologías optimizadas para satisfacer los requerimientos específicos de aplicaciones que hacían uso de cómputo intensivo, pero desafortunadamente esta solución resultó costosa en comparación con el resto de las computadoras asequibles en esa época y fue principalmente por esto que a finales de esa década el interés por estas máquinas fue descendiendo.

Con la llegada de poderosos microprocesadores usados principalmente en estaciones de trabajo, hoy en día el mercado de las supercomputadoras dirige su atención a las redes y clusters de estaciones de trabajo los cuales proveen de grandes cantidades

de cómputo a un precio muy razonable.

Sobre los beneficios y las necesidades de computación paralela de alto desempeño podemos decir que las soluciones desarrolladas por el súper cómputo son de significativa importancia en diversas áreas como ingeniería, ciencia, economía, medio ambiente y sociedad, por mencionar algunas de las más importantes.

La disponibilidad de computadoras paralelas de alto desempeño ha abierto el camino para abordar problemas que resultaban hasta ahora demasiado complejos aún para las más poderosas computadoras secuenciales.

La computación en paralelo también ha estimulado la búsqueda y el uso de nuevos métodos matemáticos los cuales siguen siendo sujetos a investigación. El paralelismo también ha alentado a sus usuarios a desarrollar algoritmos apropiados para su ejecución distribuida o en paralelo. Y seguramente existen muchas otras áreas en donde la computación en paralelo puede proveer progreso adicional en un futuro cercano.

En la historia de la computación la capacidad de cómputo se ha incrementado principalmente por el perfeccionamiento en las tecnologías de circuitos electrónicos y de los conceptos del hardware. Aunque los límites en estas áreas aún parecen lejanos ya se puede percibir que fomentar mejoras en ellas requiere incrementar los esfuerzos científicos y financieros.

Otra forma de incrementar el poder de cómputo es ofrecido por el procesamiento paralelo y distribuido. La tecnología VLSI hace posible alcanzar alto desempeño por medio de procesadores acompañados de memoria local de bajo costo. Grandes computadoras paralelas pueden ser construidas conectando procesadores en un arreglo o en un sistema multiprocesador.

Construir sistemas operativos seguros y confiables para computadoras paralelas es

un enorme reto debido a las extensas exigencias derivadas del manejo de procesos y recursos[1].

Generalmente es necesaria la investigación para mejorar la eficiencia del sistema en conjunto, derivada de la eliminación de los cuellos de botella. Para que posteriormente el software y el hardware del sistema sean perfeccionados de manera coordinada.

Para los sistemas distribuidos el reciente progreso en las tecnologías de redes de comunicación, que involucra tanto a la velocidad como al ancho de banda ha sido gigantesco.

Los avances en las redes de interconexión están siendo conducidos por perfeccionamientos en las tecnologías de transmisión (procesamiento de señales y transmisión óptica y wireless).

El progreso en las tecnologías de cómputo y comunicaciones a lo largo de la más reciente década es la base para el rápido crecimiento de la computación distribuida; los apilamientos de computadoras interconectadas son sistemas considerados confiables, veloces, escalables y baratos.

Los enormes avances en las capacidades de cómputo, la integración del cómputo y las comunicaciones, además de la disminución del costo de estas tecnologías conducen a un incremento de los sistemas distribuidos en su número y tamaño. Esta evolución también ha tomado en cuenta la explosión de la Internet .

Actualmente podemos observar que el desarrollo de clusters de computadoras heterogéneas conduce hacia la convergencia de procesamiento paralelo y distribuido.

[1]Por ejemplo, un diseño de jerarquías de memoria con una latencia de acceso aceptable y anchos de banda grandes para la memoria, requiere de sofisticados mecanismos de asignación de recursos así como de software que soporte el uso automático de esas memorias.

Planteamiento del problema

Conforme el hardware (procesador y red de comunicación) converge más y más hacia los estándares de la industria obtenemos como resultado la disponibilidad de componentes baratos, y es entonces cuando la elección instalación y configuración de software, lenguajes de programación, herramientas y aplicaciones con la misma calidad (estandarizados, confiables y fáciles de usar) se convierten en el punto clave para el buen funcionamiento del sistema. La correcta administración del sistema deberá garantizar el óptimo desempeño del cluster tipo Beowulf de la Facultad de Ingeniería.

Objetivos

Después de instalar y configurar el cluster Beowulf de la Facultad de Ingeniería, debe garantizarse su funcionamiento de manera eficiente. Para cumplir con tal objetivo deben llevarse a cabo varias tareas escenciales de administración del sistema.

Estas tareas consistirán básicamente, en:

- Administrar cuentas y acceso de usuarios al sistema.
- Administración del hardware del sistema.
- Administración del software del sistema.
- Realizar rutinas de respaldo y recuperación de aquella información importante contenida en el sistema (tanto de usuarios como del sistema).
- Monitorear el sistema y resolver eventuales problemas (de red, hardware y/o software)
- Garantizar la seguridad del sistema.
- Soporte a usuarios acerca del uso de los servicios del sistema.

Resultados esperados

Asegurar el correcto funcionamiento y óptimo desempeño del cluster de la Facultad de Ingeniería a través de una correcta administración del sistema.

Generar una referencia detallada sobre la administración de cada uno de los subsistemas a través de los cuales se logra la implementación del cluster y sobre la correcta instalación y configuración de éstos para lograr el óptimo funcionamiento del sistema entero. Esta referencia deberá introducir al lector interesado en las técnicas generales de administración implícitas en la implementacion del cluster tipo Beowulf de la Facultad de Ingeniería y deberá ayudar a encontrar soluciones y resumir procedimientos para la solución de problemas comunes en este sistema.

La primer parte de este trabajo nos ofrece los conceptos necesarios para acceder sin dificultad a los temas presentados en capítulos posteriores. En él aproximaremos al lector a una primera definición de la computación distribuida y examinaremos cuales son sus principales características. La sección dedicada a conceptos de hardware nos recordara los elementos fundamentales que definen a un sistema de cómputo y nos introducirá a los principales problemas de la programación concurrente, posteriormente describiremos brevemente tres de las arquitecturas de computadoras usadas ampliamente hoy en día. Veremos como la integración de la computación distribuida con el procesamiento en paralelo ha beneficiado el desarrollo de sistemas de alto desempeño a bajo costo y ubicaremos a este tipo de sistemas dentro de la taxonomía o clasificación de arquitecturas de computadoras. Para finalizar revisaremos con mayor detalle los esquemas de clasificación para la computación basada en clusters e introduciremos a los clusters tipo Beowulf, examinando las diferencias y similitudes que estos tienen con la arquitectura NOW que es considerada su pariente más cercano.

El segundo capítulo - **Sistema Beowulf de la Facultad de Ingeniería** - introduce al lector en la construcción del cluster de la Facultad de Ingeniería. Describe de

forma general; los requerimientos mínimos, configuración de los diferentes archivos para su funcionamiento, configuración del kernel, configuración de los servidores NIS, TFTP, DHCP y NFS. Finalmente, describe lo que falta por hacer, es decir, una vez que se tiene el cluster funcionado correctamente, que se puede hacer para mejorar su desempeño.

El tercer capitulo - **Administración de recursos hardware del sistema** - primero se describen las características de los componentes para tener una visión general del hardware con el que cuenta el cluster, para de esta manera, poder administrarlo mejor y, se hace una revisión de los parámetros importantes para su optimización. Posteriormente se habla de un componente fundamental y muy importante, la red de conexión, para una administración óptima se necesita saber los diferentes tipos de tecnologías de red que existen, para poder elegir la más adecuada para el cluster. Además, de tomar en cuenta la latencia, el ancho de banda, velocidad, etc., incluso del costo de elegir una u otra tecnología de red. Posteriormente se describe la memoria y el procesador, componentes fundamentales para el cómputo en un cluster de alto desempeño. Se muestra como un determinado procesador o tipo de memoria puede influir en el desempeño. Finalmente, se describe la escalabilidad. En este apartado se describe lo complejo o fácil que puede ser escalar un cluster de acuerdo a la topología usada.

El cuarto capítulo - **Administración de recursos Software del Sistema** - se describen los principales servicios de la arquitectura Beowulf: NFS (Network File System), NIS (Network Information System), DHCP (Dinamic Host Configuration Protocol) y, el software de comunicación: MPI (Message Passing Interface) o PVM (Parallel Virtual Machine); también se habla del sistema operativo (Red Hat).

La configuración de la red de interconexión entre los nodos y el servidor es a través del protocolo DHCP de forma que el servidor sea el encargado de asignar la dirección IP correspondiente a cada nodo; los nodos añadidos se configuran de forma que puedan arrancar vía NFS. La instalación y configuración del software PVM y XPVM

permite la ejecución de programas paralelos; MPI con el mismo propósito que PVM.

Con la finalidad de proteger al sistema de posibles vulnerabilidades en el mismo, en el quinto capítulo - **Algunos aspecto de seguridad** - se mencionan formas de proteger nuestro sistema, como es el monitorear al sistema, seguridad en cuentas de usuarios, herramientas de seguridad criptográficas, firewall y otros. Estas tareas como administradores son de suma importancia y es preciso que se realicen continuamente.

Agradecimientos: por Roy Balderas Jiménez

A mis padres,
mi más profundo agradecimiento
por su invaluable ayuda,
confianza y amor incondicional.

A mis hermanos,
por que su cariño, fuerza y talento
sigan siendo fuente de inspiración.

A Mariza,
por el amor de todos estos años
y por todos esos sueños
que nos llenan de esperanzas.

A Mamá, Papá, Nancy, Iza, Ramón, Patricia, Daniel y Amanda,
por que su presencia inunda de felicidad mis días.

Agradecimientos: por Jeanette Moreno Sánchez

Teresa,
gracias por luchar conmigo
y respetar mis decisiones.

Arturo,
gracias por guiarme en
el camino y protegernos.

Miriam,
gracias por enseñarme
a ser fuerte cuando es necesario.

Sandra,
gracias por enseñarme
a ser una persona humilde.

Amigas* y *amigos,
gracias por dejarme entrar a su vida y
compartir nuestros éxitos y fracasos.

A mi casa de estudios la UNAM,
gracias por aceptar la diversidad
de pensamiento y por seguir formando
personas que piensan y sienten.

***Una vez más y por todos los días* GRACIAS.**
Jeanette Moreno Sánchez

Agradecimientos: por Martín Placido Mojica

Primero que todo, le doy gracias a *Dios* porque me ha permitido escribir estas líneas de agradecimientos y porque ha demostrado que no he estado, no estoy y no estaré solo en ningún momento.

Le doy gracias *a mi madre* por enseñarme y demostrarme su fortaleza para seguir y siempre seguir luchando a pesar de los grandes obstáculos que se presentan en la vida y, porque siempre le ha dado a mi vida ese ingrediente fundamental que a veces olvidamos, el amor sin condiciones.

Le doy gracias *a mi padre, a mis hermanos y hermanas* que sin su apoyo incondicional no hubiera llegado hasta aquí.

Le doy un agradecimiento especial *a mi hermana Mary* por ser para mi como una segunda madre, por demostrarme no con palabras sino con actitudes que siempre podré contar con ella.

Le doy gracias a la *Universidad y a todos sus maestros* que en su momento, me enseñaron a ser un buen profesional y me dieron las herramientas suficientes para desempeñarme bien en cualquier situación.

Y por supuesto y no menos importante, le agradezco a todos mis *amigos y amigas* que de forma directa o indirecta intervinieron para lograr esta meta.

GRACIAS A TODOS.
Martín Placido Mojica

Capítulo 1
Antecedentes

1.1. La computación distribuida

Son varias las definiciones las que se le han dado a los sistemas distribuidos en la literatura existente al respecto, nunguna de ellas -por cierto- resulta enteramente satisfactoria y ninguna de ellas está en completo acuerdo con el resto. Para nuestro propósito -la exposición breve de antecedentes- baste decir por ahora que ***un sistema distribuido es una colección de computadoras independientes o autónomas interrelacionadas a través de una red y que cuenta con el software diseñado para proveer recursos de cómputo, integrados de tal forma que ante sus usuarios aparece como un único sistema de cómputo***.

Esta definición puntualiza dos aspectos importantes, el primero de ellos tiene que ver con el hardware *"las computadoras son autónomas"* y el segundo se relaciona con el software, *"los usuarios piensan que se encuentran frente a un único sistema"*. Discutiremos con mayor amplitud ambos aspectos a lo largo del presente trabajo, pero antes de ir más lejos con esta definición quizá sea más útil concentrarnos en las características de mayor importancia en los sistemas distribuidos.

1.2. Motivación y características de los sistemas distribuidos.

1.2.1. Soporte para conectar usuarios y compartir recursos

La principal meta y motivación de los sistemas distribuidos es permitir a sus usuarios acceder a recursos remotos, y compartir los recursos propios de una manera controlada.

Existen muchas razones para compartir recursos, una de ellas -la más obvia- es la económica. Por ejemplo, resulta mucho más barato compartir una impresora por

varios usuarios que comprar y mantener impresoras separadas para cada uno de ellos. De igual forma resulta económicamente conveniente compartir recursos costosos como una supercomputadora, un sistema de almacenamiento, etc.

Conectar usuarios y recursos también facilita la colaboración y el intercambio de información, como es evidente cuando observamos el éxito de la Internet.

1.2.2. Capacidad de acelerar el cómputo

Si un cálculo en particular puede ser particionado en un cierto número de *subcálculos*, es probable que éstos puedan ejecutarse concurrentemente, de tal manera que la disponibilidad del sistema nos permita *distribuir* los cálculos a través de los recursos disponibles para ser ejecutados en menor tiempo y de una forma controlada. Adicionalmente es posible evitar que un recurso sea sobrecargado con trabajos que puedan ser distribuidos a través del sistema, implementando un balanceo de carga o carga compartida.

1.2.3. Transparencia

Un importante logro para un sistema distribuido es ocultar el hecho de que sus procesos y recursos están físicamente distribuidos en múltiples computadoras. Un sistema distribuido que es capaz de presentarse ante sus usuarios y aplicaciones como si fuera un solo sistema de cómputo se conoce como *transparente*.

1.2.4. Tolerancia a fallas y fiabilidad

Si un subsistema o recurso del sistema distribuido falla, los restantes deberían ser capaces de continuar operando. Si el sistema está compuesto de un gran número de recursos autónomos la falla en uno de ellos no debería afectar al resto.

Por otra parte si el sistema está compuesto de un cierto número de pequeños subsistemas, cada uno de los cuales es responsable de una función crucial, entonces una falla cualquiera puede provocar la finalización de las operaciones del sistema completo.

En general si existe suficiente redundancia en el sistema (tanto de hardware como de datos) éste puede continuar con su funcionamiento. El mal funcionamiento en cualquier recurso debe ser detectado por el sistema distribuido y entonces éste deberá llevar a cabo las acciones apropiadas para recuperarse de la falla, tambíen debe asegurarse que todas las transferencias y funciones, antes y durante el error ocurran correctamente y finalmente cuando el recurso se recupere o sea reparado deberán existir los mecanismos para integrarlo al resto del sistema sin afectar al sistema entero.

1.2.5. Abierto

Un sistema distribuido abierto es un sistema que ofrece servicios que van de acuerdo a estándares que describen a ese servicio. Por ejemplo en redes de computadoras los estándares gobiernan el formato, los contenidos y el significado de los mensajes enviados y recibidos. Tales reglas se encuentran formalizadas en lo que conocemos como protocolos de comunicación.

Esta importante característica agrega al sistema entero flexibilidad, lo que significa que será fácilmente reconfigurable con diferentes componentes provenientes posiblemente de distintos desarrolladores. En otras palabras un sistema distribuido abierto será también extendible.

1.2.6. Escalabilidad

La escalabilidad es también una de las motivaciones y características principales de un sistema distribuido, la escalabilidad de un sistema puede ser proporcionado en relación a dos diferentes dimensiones.

Primero, puede ser escalado respecto a su tamaño lo cual quiere decir que podremos agregar usuarios y recursos al sistema (esta característica es ampliamente aprovechada por el tipo de sistemas distribuidos de los cuales se ocupa el presente trabajo).

Segundo, puede ser escalado respecto a la complejidad de sus administración lo que significa que puede ser de fácil manejo aún si está compuesto de muchos subsis-

temas o recursos.

Desafortunadamente un sistema que es escalado en una o más de sus dimensiones con frecuencia muestra una pérdida de desempeño conforme éste se va escalando.

1.3. Conceptos de hardware

Aún cuando todos los sistemas distribuidos están compuestos de múltiples Unidades Centrales de Procesamiento (CPU[2]), existen diferentes formas en que el hardware se encuentra organizado, particularmente en la manera en que están interconectados y cómo se comunican.

Varios esquemas de clasificación para describir sistemas de cómputo con múltiples CPU's han sido propuestos a través de los años, para nuestros propósitos hemos considerado sólo a los sistemas construidos a partir de computadoras independientes y hemos dividido todas las computadoras en dos grupos: aquellas que tienen memoria compartida usualmente llamadas multiprocesadoras o de multiprocesamiento y aquellas que no implementan la memoria compartida, y que son llamadas con frecuencia multicomputadoras.

Imagine el siguiente escenario: varios automóviles desean ir del punto A al punto B. Para lograrlo pueden competir por espacio en el mismo camino y terminar uno detrás del otro, o competir por una misma posición (provocando accidentes), o bien pueden conducir por caminos paralelos arribando así casi al mismo tiempo y sin invadir el camino de los otros, o pueden viajar por diferentes rutas usando caminos separados.

Este escenario captura la esencia de la computación concurrente. Existen múltiples tareas que necesitan ser realizadas (automóviles en movimiento). Cada una puede ejecutarse a la vez en un solo procesador (camino), ejecutarse en paralelo en múltiples procesadores (carriles del camino), o bien ejecutarse en procesadores

[2]Por sus siglas del Inglés Central Processing Unit.

distribuidos (caminos separados). Independientemente de la forma en que se ejecuten, las tareas con frecuencia necesitan sincronizarse para eliminar colisiones o para detenerse ante una luz de tránsito o señal de alto.

1.3.1. La esencia de la programación concurrente.

Un programa concurrente contiene dos o más procesos que trabajan juntos para desempeñar una tarea. Cada proceso es un programa secuencial - es decir una secuencia de sentencias que se ejecutan una después de la otra-.

Mientras que un programa secuencial tiene un solo hilo de control, un programa concurrente cuenta con múltiples hilos de control. Los procesos en un programa concurrente trabajan juntos comunicándose entre ellos. Esta comunicación es programada usando *variables compartidas* o *transferencia de mensajes*.

Cuando se usan variables compartidas, un proceso escribe en una variable que es leída por otro proceso. Cuando se usa el paso o transferencia de mensajes, un proceso envía un mensaje que es recibido por otro proceso.

Pero independientemente de la forma de comunicación, los procesos necesitan sincronizarse entre ellos. Existen dos formas básicas de sincronización: **exclusión mutua y sincronización por condición**.

La *exclusión mutua* atiende el problema de asegurar que las secciones críticas de una sentencia no se ejecuten a un mismo tiempo. La *sincronización por condición* resuelve el problema retrasando un proceso hasta que una condición preestablecida sea verdadera.

Como ejemplo, la comunicación entre un proceso productor y un proceso consumidor con frecuencia se implementa usando un *buffer* de memoria compartida. El productor escribe en el *buffer*, el consumidor lee del *buffer*.

La exclusión mutua se requiere para asegurar que el productor y el consumidor no accedan al buffer a la vez, y que por tanto una parte del mensaje escrito no se lea prematuramente.

La sincronización por condición en cambio es usada para asegurar que un mensaje no sea leído por el consumidor hasta después de ser escrito por el productor.

La historia de la programación concurrente ha seguido las mismas etapas que otras áreas experimentales de las ciencias de la computación.

El interés en este tópico ha crecido debido a las oportunidades que se presentaron por la inovación del hardware y se desarrolló en respuesta a los cambios tecnológicos. Sobre la marcha las aproximaciones *ad-hoc* se han ido combinando en una colección de principios centrales y técnicas generales de programación.

La programación concurrente se originó en los años 60's en el contexto de la creación de sistemas operativos. El impulso principal lo dió la invención de unidades de hardware llamadas canales, o dispositivos controladores.

Estos operaban independientemente del procesador y permitían la ejecución de operaciones de entrada/salida concurrentemente con la ejecución contínua de las instrucciones de un programa llevadas a cabo por el procesador central.

Un canal se comunicaba con el procesador central usando una interrupción - es decir una señal de hardware que dice "*detén lo que estás haciendo y comienza la ejecución de una secuencia diferente de instrucciones*".

El desafío que representó la programación -sin olvidar el reto intelectual- que resultó de la introducción de canales fue que ahora partes del programa podían ejecutarse en un orden impredecible.

Así que, si una parte del programa actualiza el valor de una variable, la interrupción puede presentarse y conducir a otra parte del programa a cambiar el valor de esta variable generando una inconsistencia de valores. Este problema en específico es conocido como el *problema de la sección crítica*.

Poco después de la introducción de los canales, los diseñadores de hardware concibieron a las *máquinas multiprocesadoras*. Por un par de décadas éstas fueron demasiado costosas como para ser ampliamente usadas, sin embargo hoy en día prácticamente todas las computadoras con grandes capacidades de cómputo conocidas, cuentan con múltiples procesadores, ellas cuentan con decenas y hasta cientos de procesadores y también son conocidas como *máquinas de procesamiento masivamente paralelo* o MPPs[3]. Cada vez son más las computadoras y estaciones de trabajo que cuentan con más de un procesador.

Las máquinas con multiprocesadores permiten ejecutar a la vez diferentes aplicaciones en diferentes procesadores. También permiten a una sola aplicación ejecutarse con mayor rapidez si ésta puede ser reescrita para aprovechar la presencia de múltiples procesadores. Pero ¿cómo hacer para sincronizar la actividad de procesos concurrentes? ¿Cómo hacer para usar múltiples procesadores para ejecutar una aplicación con mayor rapidez?

Resumiendo, los canales y los múltiples procesadores proveen oportunidades y retos. Cuando se escribe un programa concurrente se tienen que tomar decisiones acerca de qué tipo de procesos emplear, cuántos de ellos usar, y cómo deben interactuar.

Estas decisiones serán afectadas por la aplicación y el hardware en el cual se ejecutará el programa. Cualquiera que sea la decisión que se tome, la llave para desarrollar un programa correctamente es asegurarse de que la interacción entre procesadores está apropiadamente sincronizada.

[3]Por sus siglas del inglés Massively Parallel Processing

Aún cuando la intención del presente trabajo no es examinar todos los aspectos de la programación concurrente, es importante mencionar que todos los ejemplos aquí presentados se concentran en los llamados programas imperativos con concurrencia, comunicación y sincronización explícita.

Está claro que es deber del programador especificar las acciones que tomará cada proceso, y cómo se comunica y se sincroniza, lo cual contrasta con los programas declarativos - programas funcionales o lógicos - en los cuales la concurrencia está implícita y no hay escritura y/o lectura del estado del programa.

En programas declarativos, partes independientes del programa se pueden ejecutar en paralelo; se comunican y sincronizan implícitamente cuando una parte depende del resultado producido por otra.

Aunque esta aproximación declarativa es interesante e importante; es la aproximación imperativa la que se es ampliamente usada. Además para implementar un programa declarativo en una máquina tradicional, irremediablemente se debe escribir un programa en su versión imperativa.

El presente trabajo también se concentra en programas concurrentes en los cuales la ejecución de procesos es asíncrona - es decir cada proceso se ejecuta a su propia velocidad-.

Tales programas pueden ejecutarse intercalando los procesos en un solo procesador o ejecutándolos en paralelo, en un flujo múltiple de instrucciones, esto es en un multiprocesador de múltiple flujo de datos; esta clase de máquinas incluye multiprocesadores de memoria compartida, multiprocesadores de memoria distribuida, y redes de estaciones de trabajo como se describirá más adelante.

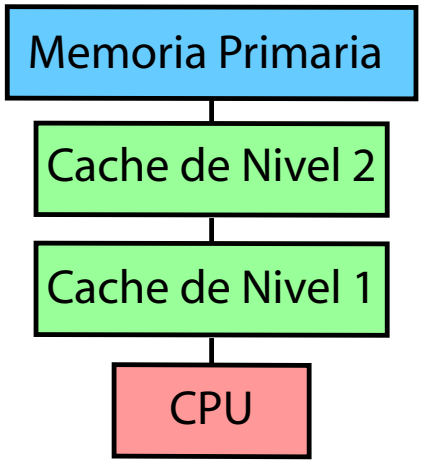

Figura 1.1: Procesador, cache y memoria en una máquina moderna.

1.3.2. Arquitecturas de Hardware.

En esta sección se resumen los principales atributos de las arquitecturas de computadoras modernas. Primero se describen los procesadores individuales y las memorias cache para después examinar a los multiprocesadores de memoria compartida y finalmente se describen las máquinas de memoria distribuida, lo cual incluye a las multicomputadoras y las redes de computadoras, sobre las cuales recaerá la atención del resto del presente trabajo.

1.3.2.1. Procesadores y Caches

Una computadora de un sólo procesador contiene varios componentes. Una unidad central de procesamiento (CPU), una memoria primaria, uno o más niveles de memoria cache, almacenamiento secundario, y una variedad de dispositivos periféricos - monitor, teclado, mouse, modem, cdr, impresora, etc. Los componentes principales con respecto a la ejecución de programas son la CPU y las memorias primaria y cache; la manera como se relacionan éstas se describe en la Figura 1.1

La CPU manda llamar instrucciones de la memoria, las decodifica y ejecuta. Ésta contiene una unidad de control, una unidad aritmético/lógica (ALU), y registros. La unidad de control emite señales que controlan las acciones de la ALU, la memoria del sistema y sus dispositivos externos.

La ALU implementa las instrucciones aritméticas y lógicas definidas en el conjunto de instrucciones del procesador. Los registros contienen las instrucciones, datos y el estado de la máquina (incluyendo el contador del programa).

Una memoria cache es una pequeña y rápida unidad de memoria que es usada para acelerar la ejecución de un programa. Almacena el contenido de aquellas localidades de memoria que han sido recientemente usadas por el procesador.

La razón fundamental para la presencia de la memoria cache es que la mayor parte de los programas muestran localidades temporales, lo que quiere decir que una vez que hacen referencia a una localidad de memoria es muy probable que lo hagan nuevamente en un futuro próximo. Por ejemplo las instrucciones dentro de ciclos que son mandados llamar y ejecutados muchas veces.

Cuando un programa intenta leer una localidad de memoria, el procesador primero busca en la cache si el dato se encuentra ahí -*cache hit*- entonces es leído de la cache. Si el dato no se encuentra -*cache miss*- entonces es leído de la memoria principal por ambos, el cache y la memoria principal.

De forma similar cuando los datos son escritos por un programa son puestos en la memoria cache y en la memoria principal (posiblemente sólo en la memoria principal). En la cache de *escritura directa*, los datos son puestos en memoria inmediatamente; en un cache de *escritura posterior* los datos son almacenados en memoria con demora.

El punto importante es que después de una escritura, los contenidos de la memoria primaria pueden ser inconsistentes temporalmente con respecto al contenido de la memoria cache.

Para incrementar la tasa de transferencia o ancho de banda entre las memorias

cache y primaria, una entrada de datos en una memoria cache contiene típicamente múltiples palabras contiguas. Estas entradas son llamadas líneas de cache o bloques.

En el momento en que se presenta una ausencia de datos en el cache el contenido completo de una línea de cache se transfiere entre la memoria principal y la cache. Esto es efectivo porque la mayor parte de los programas presentan localidad espacial, lo que significa que cuando hacen referencia a una palabra en memoria referencian rápidamente a las palabras vecinas en la memoria.

Los procesadores modernos típicamente contienen dos niveles de cache. Las de nivel 1 se encuentran cercanas al procesador, las de nivel 2 se encuentran entre la cache de nivel 1 y la memoria primaria. La cache de nivel 1 es más pequeña y rápida que la de nivel 2 y con frecuencia ésta organizada de forma diferente.

Por ejemplo, la cache de nivel 1 a menudo es directamente mapeada, en tanto que la memoria cache de nivel 2 es con frecuencia una colección asociativa.

Por otra parte la cache de nivel 1 frecuentemente contiene caches separados para instrucciones y datos, mientras que la de nivel 2 está comúnmente unificada, lo que quiere decir que contiene datos e instrucciones.

Para ilustrar las diferencias en velocidad entre los tipos de memoria podemos decir lo siguiente: los registros pueden ser accedidos en un ciclo de reloj ya que son pequeños e internos a la CPU. El contenido del cache de nivel 1 típicamente puede ser accedido en 1 ó 2 ciclos de reloj. En contraste, toma un orden de 10 ciclos de reloj acceder al cache de nivel 2, y probablemente entre 50 y 100 ciclos de reloj para acceder a la memoria primaria.

Existen diferencias similares en el tamaño de los distintos tipos de memoria: en la CPU existen unas pocas docenas de registros, una memoria cache de nivel 1 contiene unas cuantas docenas de kilobytes, una cache de nivel 2 contiene el orden de

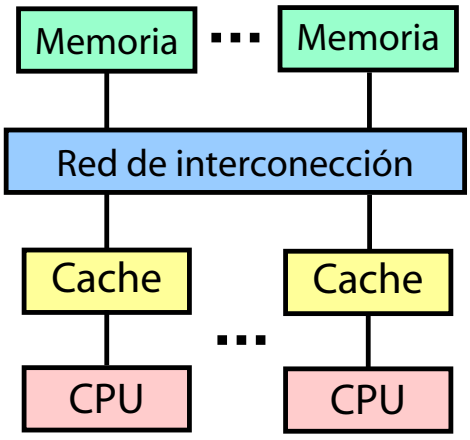

Figura 1.2: Estructura de los multiprocesadores de memoria compartida.

un megabyte, y la memoria primaria contiene de decenas a cientos de megabytes.

1.3.2.2. Multiprocesadores de memoria compartida.

En multiprocesadores con memoria compartida los procesadores y los módulos de memoria están conectados a la manera de una red de computadoras, como se muestra en la figura 1.2.

Los procesadores comparten la memoria primaria, pero cada procesador tiene su propia memoria cache.

En pequeñas computadoras multiprocesadoras que tienen desde uno hasta 30 procesadores la red de interconexión se implementa por un *bus de memoria* o por un *switch crossbar*. Tales multiprocesadores son conocidas como máquinas MAU[4] debido a que tiene un tiempo de acceso a memoria uniforme de cada procesador a cada localidad de memoria, las máquinas MAU también son llamadas de multiprocesamiento simétrico (SMPs.- Symmetric multiprocessors).

Las grandes computadoras multiprocesadoras, de memoria compartida -aquellas que cuentan con decenas a cientos de procesadores- tienen una memoria jerárquicamente organizada.

[4]Por sus siglas en Inglés Memory Access Uniform

En particular la red que interconecta áreas de memoria es una colección de árboles -estructurados- de switches y memorias.

Consecuentemente cada procesador tiene algunas memorias que están cercanas y otras que son lejanas. Esta organización elimina la congestión que podría ocurrir si hubiera un solo bus o switch compartido.

También existen computadoras multiprocesadoras con tiempos de acceso a memoria no uniformes o NUMA[5].

En las máquinas UMA como en las NUMA cada procesador tiene su propia cache; si dos procesadores referencian diferentes localidades de memoria los contenidos pueden ser colocados de manera segura en los caches de ambos procesadores. Pero los problemas pueden crecer cuando dos procesadores referencian a la misma localidad de memoria al mismo tiempo.

Si ambos procesadores leen la misma localidad de memoria cada uno puede cargar una copia del dato que contiene en su cache local. Pero si un procesador escribe en la misma localidad se presenta un problema de consistencia de cache: el cache del otro procesador no contiene el valor correcto.

Así pues cualquiera de los otros caches necesitan ser actualizados con el nuevo valor o la entrada de datos al cache necesitarán ser invalidadas. Cada procesador tiene que implementar un protocolo de consistencia de cache en su hardware. Un método para ello consiste en tener cada cache vigilado en el bus de direcciones de memoria observando las referencias hechas a localidades de este cache.

La escritura también conduce a problemas de consistencia en memoria: ¿cuándo es actualizada realmente la memoria primaria? Por ejemplo si un procesador escribe

[5]Por sus siglas en Inglés nonuniform memory access time

en una localidad de memoria y otro procesador lee posteriormente de esa localidad, ¿el segundo procesador tiene la garantía de que verá el nuevo valor? Existen varios modelos de consistencia diferentes.

La *consistencia secuencial* es el modelo con mayores virtudes ya que éste garantiza que las actualizaciones de memoria ocurrirán en algún orden secuencial y que cada procesador verá el mismo orden.La *consistencia de procesador* es el modelo más débil, éste asegura que las escrituras hechas por cada procesador ocurrirán en memoria en el orden en que ellas se ejecuten por el procesador, pero en este modelo las escrituras ejecutadas por diferentes procesadores pueden ser vistas en diferente orden por el procesador. La *consistencia de liberación* por otra parte es un modelo todavía más débil, éste sólo asegura que la memoria privada es actualizada en puntos de sincronización especificados por el programador.

El problema de la consistencia de la memoria plantea una elección entre la facilidad en la programación y la redundancia en la implementación.

El programador intuitivamente espera consistencia secuencial, debido a que un programa lee y escribe variables sin respetar en donde serán almacenados los valores dentro de la máquina.

En particular cuando a un proceso se le asigna una variable, el programador espera que el resultado de esa asignación sea vista inmediatamente por todos los demás procesos en el programa.

Por otra parte, la implementación de la consistencia secuencial es costosa y provoca lentitud en el desempeño del sistema entero. Esto debido a que en cada escritura, el hardware tiene que invalidar o actualizar otros caches y actualizar la memoria primaria; más aún estas operaciones deben ser atómicas o indivisibles.

Consecuentemente los multiprocesadores típicamente implementan un modelo más

débil de consistencia y dejan al programador la tarea de insertar instrucciones de sincronización de memoria. Los compiladores y bibliotecas con frecuencia toman cuidados especiales en este aspecto de tal manera que el programador de aplicaciones no tiene por que hacerlo.

Como puede verse las líneas de cache frecuentemente contienen múltiples palabras que son transferidas hacia y desde la memoria como una unidad.

Imagine por ejemplo que las variables **x** y **y** ocupan una palabra cada una, y son almacenadas como palabras adyacentes en la memoria, que es mapeada en la misma línea de cache. Supongamos que algún proceso está ejecutándose en el procesador 1 de un multiprocesador y leyendo y escribiendo la variable **x**. Finalmente supongamos que otro proceso ésta ejecutando en el procesador 2 y escribiendo y leyendo la variable **y**. Entonces en cualquier momento que el procesador 1 accede a la variable **x** la línea de cache de ese procesador también contendrá una copia de la variable **y**; y un proceso similar ocurrirá con el procesador 2.

El escenario anterior causa lo que se conoce como f*alsa compartición*, el proceso en realidad no comparte las variables **x** y **y**, pero el hardware trata a ambas variables como una unidad.

Consecuentemente, cuando el proceso 1 actualiza **x**, la línea de cache conteniendo a ambas **x** y **y** en el procesador 2 tiene que ser invalidada o actualizada.

En forma similar cuando el procesador 2 actualiza **y** la línea de cache conteniendo **x** y **y** en el procesador 1 tiene que ser actualizada o invalidada.

Las invalidaciones o actualizaciones en cache disminuyen la velocidad del sistema de memoria, así que el programa se ejecutará mucho más lento que en el caso en el que las dos variables se ubiquen en distintas líneas de cache.

El punto clave es que el programador entienda que los procesos no comparten variables porque en realidad el sistema de memoria implementa la *redundancia* y la *compartición* para lidiar con este tipo de problemas.

Para evitar *la falsa compartición*, el programador tiene que asegurarse que las variables que se escriben para diferentes procesos no deben estar en localidades de memoria adyacentes.

Una manera de asegurarse de ello es usar lo que se conoce como **padding** o relleno, que no es otra cosa que declarar variables ficticias que tomen espacio y separen variables reales de otras adyacentes. Ésta es una situación de trueque entre espacio y tiempo, es decir de desperdiciar espacio para reducir tiempo de ejecución.

Resumiendo, los multiprocesadores emplean caches para mejorar el desempeño, pero de cualquier forma la presencia de la memoria jerarquizada representa problemas de *consistencia de memoria y cache* y como ya observamos la posibilidad de *falsa compartición*. Consecuentemente, para obtener el máximo desempeño en una máquina dada, el programador debe conocer las características del sistema de memoria y debe escribir programas que tomen en cuenta todo esto.

1.3.2.3. Multicomputadoras de memoria distribuida y redes.

En multiprocesadores de memoria distribuida, se presenta de nuevo una red de interconexión, pero en esta ocasión cada procesador tiene su propia memoria privada. Como se muestra en la figura 1.3

La ubicación de la red de interconexión y los módulos de memoria es opuesta a su ubicación en las máquinas multiprocesadoras con memoria compartida.

La red de interconexión soporta paso de mensajes en lugar de *lectura/escritura* en memoria. Así que los procesos se comunican entre ellos enviando y recibiendo

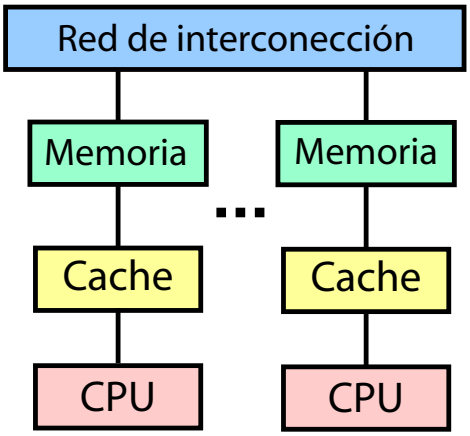

Figura 1.3: Estructura de las máquinas de memoria distribuida.

mensajes. Cada procesador tiene su propia cache, pero debido a que la memoria no es compartida no existen problemas de consistencia en memoria o cache.

Una multicomputadora es una máquina con multiprocesadores de memoria distribuida en la cual los procesadores y la red son físicamente cercanos uno del otro (comúnmente en el mismo gabinete). Por esta razón una multicomputadora también es conocida como máquina fuertemente acoplada[6].

Una multicomputadora es usada por una o al menos unas cuantas aplicaciones al mismo tiempo, y cada aplicación usa un conjunto de procesadores dedicados.

La red de interconexión provee alta velocidad y gran ancho de banda como ruta de comunicación entre procesadores. Y está típicamente organizada como masa o hipercubo (la organización en hipercubo constituyó los primeros intentos de multicomputadoras).

Un sistema de red es de hecho una máquina multiprocesadora de memoria distribuida en la cual los nodos están interconectados por una red de área local (ethernet por mencionar un ejemplo) o bien por una de área amplia como Internet.

[6]Tightly coupled machine

Consecuentemente los sistemas organizados en esos sistemas de red son llamadas máquinas multiprocesadoras débilmente acopladas [7]. Aquí nuevamente observamos como los procesadores se comunican enviando mensajes, pero estos toman una mayor cantidad de tiempo para entregarse que en una multicomputadora y existe también una mayor contención en la red.

Los sistemas más comunes basados en redes de interconexión y que constan de una colección de estaciones de trabajo son conocidos como red de estaciones de trabajo (NOW[8]) o Pila de estaciones de trabajo (COW[9]). Estas estaciones de trabajo ejecutan una sola aplicación, aplicaciones por separado, o una mezcla de ambas.

En el presente una manera cada vez más popular y económica de contar con máquinas multiprocesadoras de memoria distribuida y alto desempeño es construyendo sistemas conocidos como Beowulf.

Un cluster Beowulf -objeto de estudio del presente trabajo- se construye usando hardware de precio accesible y software libre, por ejemplo procesadores Pentium, tarjetas adaptadoras de red, discos y sistemas operativos como Linux. Combinaciones híbridas de máquinas con memoria distribuida y memoria compartida también son posibles en este tipo de arquitectura.

Los nodos de una máquina de memoria distribuida pueden ser a su vez máquinas multiprocesadoras de memoria compartida y no sólo nodos de un único procesador. Las actuales interconexiones de redes pueden soportar paso de mensajes y mecanismos para acceso directo a memoria remota (esto en la actualidad es lo que ha permitido la construcción de las más poderosas máquinas que se han conocido).

La combinación más general es una máquina que soporta lo que podría ser con-

[7] Loosely coupled machine
[8] Network of Workstations
[9] Cluster of workstations

siderada una versión distribuida de una máquina con memoria compartida. Esto hace posible la creación de máquinas fáciles de programar para muchos tipos de aplicación, pero presenta nuevamente los problemas inherentes a la consistencia de la cache y memoria.

1.4. Procesamiento en Paralelo.

En años recientes hemos atestiguado una constante y creciente aceptación del procesamiento en paralelo, ya sea para el cómputo científico de alto desempeño o bien para aplicaciones de propósito general. Esto como resultado de la demanda creciente de computación de alto desempeño, bajo costo, y productividad sustentable.

La aceptación del procesamiento en paralelo ha sido facilitado por dos importantes avances tecnológicos: los procesadores masivamente paralelos o MPPs[10] y el uso extenso de la computación distribuida.[6]

Las computadoras MPPs son actualmente las más poderosas en el mundo. Estas máquinas combinan desde algunas decenas, hasta algunos cientos de CPUs en un único gabinete con cientos de Giga-bytes de memoria. Los MPPs ofrecen un enorme poder de cómputo y son comúnmente usados para ejecutar enormes cálculos, resolviendo problemas que constituyen verdaderos retos computacionales.

El segundo desarrollo importante que ha influido en la resolución de este tipo de problemas es la computación distribuida. En tanto más y más organizaciones cuentan con una red local de alta velocidad interconectando sus sistemas de cómputo, la combinación de los recursos computacionales a través de estas redes supera el poder de procesamiento de una única computadora de alto desempeño.

El factor más importante en la computación distribuida es el costo. El costo de grandes MPPs es típicamente mayor a los 10 millones de dólares y en contraste los usuarios del cómputo distribuido observan un costo mucho menor al usar un conjun-

[10] Por sus siglas en Inglés Massively Parallel Processors

to de computadoras con las que ya cuentan para solucionar sus problemas, si bien no es común para los usuarios del cómputo distribuido encontrar el mismo poder de cómputo de las grandes MPPs, por lo menos son capaces de resolver problemas varias veces más complejos de los que pueden solucionarse usando una sola de sus computadoras de propósito general.

El modelo de *paso de mensajes* se ha convertido en el paradigma de elección para establecer comunicación entre procesos debido principalmente al número y la variedad de multiprocesadores que lo soportan, así como también al uso que hacen de él varias aplicaciones y lenguajes. Este documento concentra su atención en el área de la Computación Distribuida conocida como Clustering[11].

1.5. Arquitecturas

1.5.1. Otros esquemas de Clasificación para los sistemas de cómputo

Con el paso de los años, se han propuesto diversos esquemas de clasificación para los sistemas de cómputo, pero ninguno de ellos ha tenido un éxito completo ni se ha adoptado de manera amplia. Es probable que la taxonomía más citada sea la sugerida por Michael J. Flynn[1] en 1966.

Para Flynn, el elemento esencial en el proceso de cómputo es la ejecución de una secuencia de instrucciones sobre un conjunto de datos[1].

De acuerdo con su clasificación, los sistemas de cómputo se caracterizan, y se distinguen unos de otros por la multiplicidad de hardware con que están provistos para atender *flujo(s) de instrucciones* y *flujo(s) de datos*. La siguiente lista muestra la llamada "clasificación de las cuatro máquinas de Flynn":

- **SISD** [12]- Un flujo de Instrucciones/Un flujo de Datos
- **SIMD** [13]- Un flujo de Instrucciones/Múltiples flujos de Datos

[11] En donde muchas máquinas interconectadas trabajan colectivamente para procesar instrucciones y datos

[12] Por sus siglas en Inglés Single Instruction stream/Single Data stream.

[13] Por sus siglas en Inglés Single Instruction stream / Multiple Data stream.

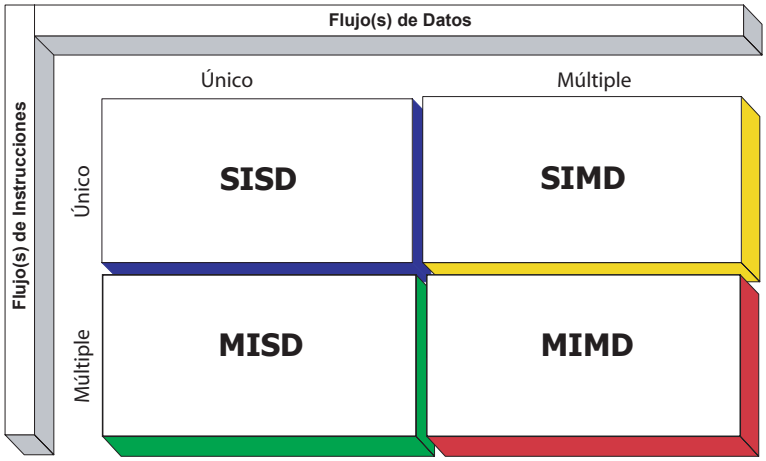

Figura 1.4: Clasificación de los Sistemas de Cómputo de Flynn.

- **MISD** [14]- Múltiples flujos de Instrucciones/Un flujo de Datos (*sin aplicación real*)
- **MIMD** [15]- Múltiple flujo de Instrucciones/Múltiple flujo de Datos

Los sistemas distribuidos pueden ser clasificados en la arquitectura MIMD en tanto que cada uno de sus nodos puede ser clasificado dentro de la arquitectura SISD, ambos esquemas se explican a continuación:

SISD - Esta arquitectura representa a la mayor parte de las computadoras seriales disponibles hoy en día. Las instrucciones son ejecutadas secuencialmente pero pueden ser superpuestas en sus estados de ejecución (usando pipelined[16]). La mayor parte de los sistemas de un único procesador son del tipo *pipelined*.

MIMD - La mayoría de los sistemas de múltiples procesadores y de múltiples computadoras pueden ser clasificadas en esta categoría. Una computadora MIMD

[14]Por sus siglas en Inglés Multiple Instruction stream/Single Data stream.
[15]Por sus siglas en Inglés Multiple Instruction stream/Multiple Data stream.
[16]Yuxtaposición. Técnica utilizada por microprocesadores avanzados a través de la cual el microprocesador comienza a ejecutar una segunda instrucción antes que la primera haya sido finalizada

implica intrínsecamente interacciones entre los n procesadores puesto que todo el flujo de la memoria proviene del mismo espacio de datos compartido por todos los procesadores. MIMD puede ser sintetizado como sigue:

- Cada procesador ejecuta su propia secuencia de instrucciones.
- Cada procesador trabaja en una parte diferente del problema.
- Cada procesador se comunica con los otros procesadores del sistema.
- Es posible que los procesadores deban esperar datos provenientes de otros procesadores.

Si los n flujos de datos provinieron de subespacios desarticulados de la memoria compartida, entonces podríamos llegar a tener la llamada operación SISD múltiple, la cual no es otra cosa que un conjunto de n sistemas uniprocesador (de un solo procesador) SISD independientes. Se dice que una computadora del tipo MIMD es fuertemente acoplada (tightly coupled) si el grado de interacciones entre sus procesadores es alto. De no ser así se considera débilmente acoplada (loosely coupled). Un ejemplo de cada una sería el siguiente:

Los **sistemas de multiprocesamiento débilmente acoplados** no tienen la cantidad de conflictos con la memoria como aquellos que experimentan los sistemas fuertemente acoplados.

En estos sistemas de acoplamiento débil, cada procesador tiene un conjunto de dispositivos de entrada/salida y una gran cantidad de memoria local a través de la cual acceden a la mayor parte de las instrucciones y datos.

Nos referimos al procesador, la memoria local y sus dispositivos de Entrada/Salida como “módulo de cómputo”. Los procesos que se ejecutan en diferentes “módulos de cómputo” comunicados a través de un intercambio o transferencia de mensajes.

El grado de acoplamiento en tales sistemas es muy débil. El factor que determina el grado de acoplamiento es la topología de comunicación del sistema de transferencia de mensajes asociado. Los sistemas débilmente acoplados son usualmente eficientes cuando las interacciones entre tareas son mínimas.

Como quiera que sea, debido a la gran variabilidad de los tiempos de interferencia, el rendimiento o producción total de la jerarquía de los microprocesadores débilmente acoplados puede ser demasiado bajo para ciertas aplicaciones que requieren tiempos rápidos de respuesta.

Cuando se requiere procesamiento en tiempo real o de alta velocidad, los sistemas fuertemente acoplados deben ser usados. Un ejemplo de una arquitectura débilmente acoplada está en los clusters tipo Beowulf. En donde cada nodo es un sistema de cómputo (computadora por separado) interconectado a través de un canal de datos (comunmente Ethernet).

En un sistema de multiprocesamiento fuertemente acoplado los multiprocesadores se comunican a través de una memoria principal y compartida, de esta manera la razón de la velocidad a la cual los procesadores pueden comunicarse de un procesador a otro es del orden del ancho de banda de la memoria.

Los sistemas fuertemente acoplados pueden tolerar un alto grado de interacciones entre las tareas sin un significativo deterioro en su desempeño. Un ejemplo de máquinas de multiprocesamiento fuertemente acoplado es el Transputer que es una máquina en donde dos o más procesadores comparten la misma memoria, tal como sucede en el procesamiento simétrico (Symmetric Multiprocessing).

Debido a que la categoría MIMD en la taxonomía de Flynn incluye una amplía gama de tipos de computadoras, en 1998 E. Johnson propuso una clasificación adicional de tales máquinas basada en su estructura de memoria, distinguiendo la memoria global de la distribuida y el mecanismo usado para comunicación/sincronización dis-

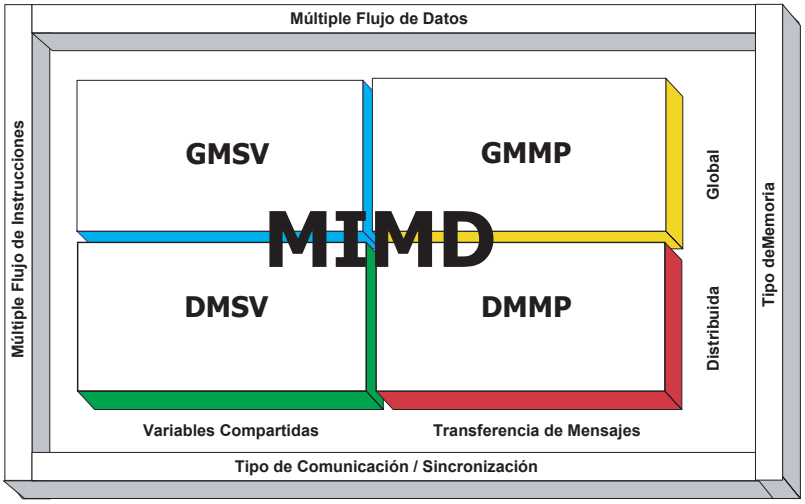

Figura 1.5: Clasificación de los Sistemas de Cómputo MIMD de Flynn-Johnson.

tinguiendo entre el método de variables compartidas y el de transición de mensajes. La clasificación propuesta es la siguiente:

- **GMSV** - Memoria Global/Variables Compartidas[17].
- **GMMP** - Memoria Global/Transición de Mensajes[18].
- **DMSV** - Memoria Distribuida/Variables Compartidas[19].
- **DMMP** - Memoria Distribuida/Transición de Mensajes[20].

En la clasificación anterior, podemos observar que las arquitecturas DMSV y DMMP están débilmente acopladas, en tanto que las dos restantes GMSV y GMMP están fuertemente acopladas. Debido a que la transición de mensajes es el componente clave de la tecnología de clustering del tipo Beowulf, la subclasificación DMMP es la que mejor la define.

Una taxonomía alternativa de la categoría MIMD propuesta por Johnson es la que sugiere Tanenbaum[7]. La taxonomía propuesta por Tanenbaum se distingue

[17]Por sus siglas del Inglés Global Memory/Shared Passing.
[18]Por sus siglas del Inglés Global Memory/Message Passing. (sin aplicación real)
[19]Por sus siglas del Inglés Distributed Memory/Shared Variables.
[20]Por sus siglas del Inglés Distributed Memory/Message Passing.

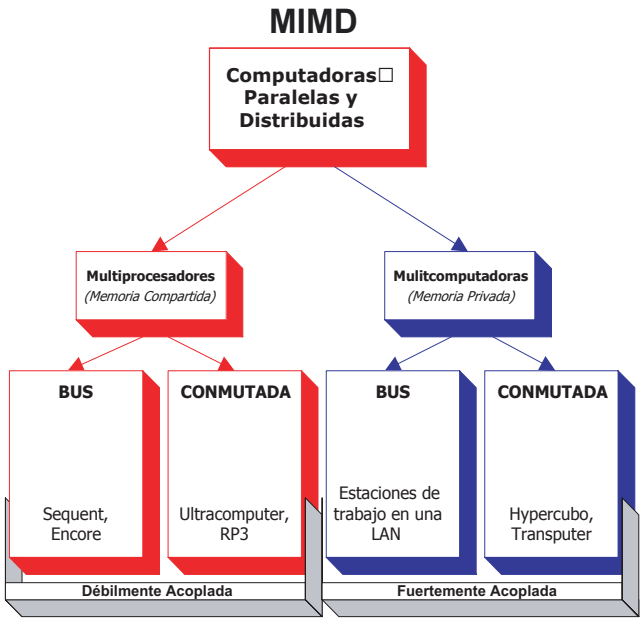

Figura 1.6: Clasificación de los Sistemas de Cómputo Paralelos y Distribuidos de Tanenbaum

de la de Flynn-Johnson ya que ésta clasifica a los sistemas de cómputo basada en su arquitectura de interconexión de red, en tanto que el modelo de Flynn-Johnsosn los diferencia tomando en cuenta el modelo de programación del sistema. Ambos modelos son válidos y presentan información similar, sin embargo el modelo de Flynn-Johnson es más descriptivo ya que atiende las conexiones lógicas entre los procesadores más que las conexiones físicas.

1.5.2. Esquemas de Clasificación para la computación basada en Clusters

En tanto que el modelo de Flynn-Johnson para la categoría DMMP representa a un número de implementaciones de cómputo diferentes, DMMP es también una descripción detallada para el área del cómputo en que se concentra el presente trabajo de tesis, es decir el área de cómputo basada en clusters.

Si quisiéramos definir a las máquinas del tipo DMMP (más allá de como lo hace el modelo Flynn-Johnson) por el camino de la clasificación de las arquitecturas,

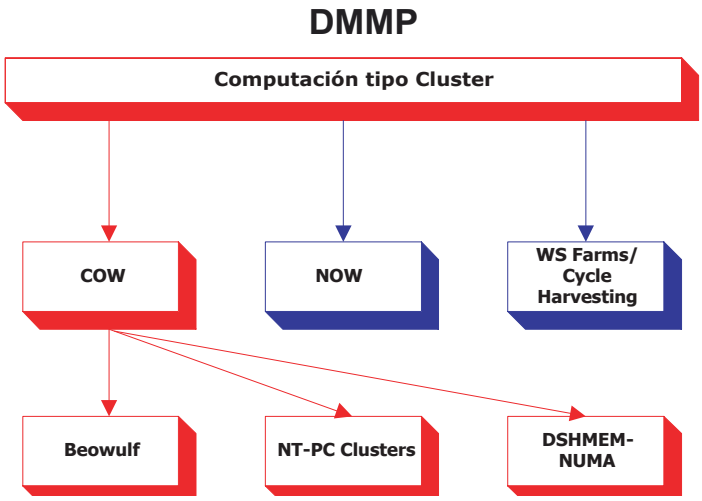

Figura 1.7: Clasificación de los Sistemas de Cómputo tipo Cluster.

estaríamos definiendo implementaciones de máquinas específicas. Una de tales clasificaciones es la propuesta por Thomas L. Sterling et al[2]

Aunque el presente trabajo se concentra específicamente en los clusters tipo Beowulf, por su similitud, examinaremos brevemente las carácterísticas y diferencias entre los clusters de estaciones de trabajo COW[21], y las redes de estaciones de trabajo NOW[22].

En la taxonomía de computadoras en paralelo que ya examinamos brevemente, observamos que la definición de los clusters Beowulf caen en alguna parte entre las computadoras masivamente paralelas MPP (como nCube, CM5, Convex SPP, Cray T3D, Cray T3E, etc.) y las redes de estaciones de trabajo (NOW's) consideramos que el proyecto Beowulf se ha beneficiado de los desarrollos y experiencias de estas dos clases de arquitectura.

Las MPPs son típicamente grandes computadoras y tienen una latencia de red de interconexión más baja que la de un cluster Beowulf. En el caso de las MPPs los programadores deben preocuparse por resolver problemas como localidad, balanceo

[21]Por sus siglas del Inglés Cluster Of Workstations
[22]Por sus siglas del Inglés Networks Of Workstations

de carga, granularidad, y sobrecarga en las comunicaciones, para obtener el mejor desempeño en sus aplicaciones. Aún en máquinas de memoria compartida, muchos programadores desarrollan sus programas haciendo uso de la transferencia de mensajes. Aquellos programas que no requieren de computación y comunicación detallada y experta (de grano fino) usualmente podrán portarse y correr eficientemente en clusters Beowulf.

Por otra parte programar para una NOW usualmente se convierte en un intento de aprovechar los ciclos de cómputo desperdiciados en un conjunto de estaciones de trabajo interconectadas, distribuidas en un laboratorio o en todo un campus. Programar en este ambiente requiere el uso de algoritmos que sean extremadamente tolerantes a problemas de balanceo de carga y a una muy grande latencia en las comunicaciones. Cualquier programa que se ejecute en una NOW podrá ejecutarse al menos con el mismo desempeño en un cluster Beowulf.

Un cluster tipo Beowulf se distingue de una red de estaciones de trabajo NOW por varias sutiles pero muy significativas características. Primero, los nodos en el cluster son dedicados. Esto ayuda a disminuir los problemas de balanceo de carga, debido a que el desempeño de nodos individuales no está sujeto a factores externos. También y dado que la red de interconexión entre nodos se encuentra aislada de la red externa, la carga de red está determinada sólo por la aplicación que se encuentra ejecutándose en el cluster y esto reduce los problemas asociados con la latencia impredecible, común en las NOWs. Todos los nodos en el cluster se encuentran en el ámbito administrativo del cluster.

La red de interconexión en el cluster no es visible desde redes exteriores así que la única autentificación necesaria entre procesadores se realiza para asegurar la integridad del sistema. El administrador de una NOW en cambio debe estar mucho más involucrado en la seguridad de la red.

Otro ejemplo ilustrativo respecto de las diferencias entre clusters Beowulf y NOW's

es que el software en los clusters provee un identificador de procesos global y esto es suficiente para habilitar los mecanismos necesarios para que un proceso en un nodo envíe señales a otro nodo del sistema, todo esto en el dominio del usuario. Esto en cambio no es posible en una NOW.

Finalmente, los parámetros del sistema operativo pueden ser reajustados y puestos a punto para mejorar el desempeño. Por ejemplo una estación de trabajo puede ser puesta a punto para proveer al usuario de forma directa una sensación de mejor rendimiento (respuesta instantánea, buffers pequeños, etc.), pero en el cluster en cambio, los nodos pueden ser reconfigurados para dedicarse a proveer mejor rendimiento en la ejecución de procesos debido a que ellos no interactúan directamente con los usuarios.

1.5.3. Beowulf

1.5.3.1. Historia

En el verano de 1994 Thomas Sterling y Don Becker, como parte de su trabajo en el CESDIS[23] y bajo el patrocinio del proyecto ESS[24], construyeron un Cluster al que llamarón *Beowulf*; constituido por 16 computadoras personales con microprocesadores Intel DX4 e interconectadas por una red Ethernet (channel bonded Ethernet); fue construido con la principal idea de proveer sistemas basados en COTS[25] para satisfacer las necesidades que se esparcian con gran rapidez tanto en la NASA como en la comunidad académica y de investigación y tuvo un éxito instantáneo. Al esfuerzo desarrollado para la construcción de esta primera máquina le siguió un rápido crecimiento hasta convertirse en lo que ahora conocemos como el proyecto Beowulf[3].

Thomas Sterling llamó al proyecto con el nombre de *Beowulf* en honor al personaje de uno de los primeros poemas épicos ingleses de que se tenga memoria alguna; dicho poema narra la historia de un héroe legendario de gran fuerza y valor del

[23] Center of Excellence in Space Data and Information Sciences
[24] Earth and Space Sciences
[25] Commodity off the shelf

siglo VI, quien viniendo de un distante reino, destruyó al monstruo Grendal quien amenazaba y oprimía a un pueblo entero[4]. Como metáfora "Beowulf"se aplica en esta estrategia relativamente nueva de la computación de alto desempeño, que pretende sacar provecho de la tecnología ya existente en el mercado para derrotar los opresivos costos en tiempo y dinero que hasta ahora han acompañado a la supercomputación, liberando así a los científicos, ingenieros y otros, para consagrarse a sus respectivas disciplinas. Como quiera que sea, *Beowulf* tanto en el mito como en la realidad, desafía y conquista el obstáculo dominante en sus respectivos ámbitos, abriendo así el camino a futuros logros[5].

Definición de un Sistema Beowulf

Un Sistema Beowulf es una arquitectura multicomputadora que puede ser usada para cálculos paralelos. Es un sistema que generalmente consiste de un nodo servidor y uno o varios nodos clientes conectados a través de una red Ethernet u otro tipo de interconexión de red. Es un sistema construido mediante hardware conocido por todos o fácil de conseguir (commodity), como cualquier PC capaz de correr Linux, adaptadores Ethernet estándares y switches. No contiene algún componente de hardware específico y es fácilmente reproducible. Beowulf también usa software commodity, como el sistema operativo Linux, PVM y MPI. El nodo servidor controla el Cluster completo y proporciona archivos a los nodos clientes. Es también la consola del Cluster y el enlace (gateway) al mundo exterior. Los grandes sistemas Beowulf pueden tener más de un nodo servidor y probablemente algunos otros nodos dedicados a tareas específicas, como consolas o estaciones de monitorización. En la mayoría de los casos, los nodos clientes de un sistema Beowulf son mudos, y cuanto más mudos mejor. Los nodos clientes son configurados y controlados por el nodo servidor, y sólo hacen lo que se les dice que hagan. En las configuraciones donde los clientes no tienen discos (diskless), los nodos clientes ni siquiera conocen su dirección IP o nombre hasta que el servidor le dice cual es ésta [9].

Capítulo 2
Sistema Beowulf de la Facultad de Ingeniería

El cluster tipo Beowulf de la Facultad de Ingeniería fue diseñado e implementado por un grupo de tesistas, en el año 2001. Este sistema fue el primero en su tipo en ser desarrollado en la Facultad de Ingeniería, específicamente en el Labaratorio de Telemática del departamento de Ingeniería en computación de la División de Ingeniería Eléctrica.

2.1. Diseño e implementación del Cluster

El sistema Beowulf necesitó de muchos parámetros de diseño a considerar para su terminación exitosa. En los apartados siguientes se resumen los más significativos.

2.1.1. Diseño

Es importante considerar los aspectos de diseño para tomar ciertas decisiones entre las que destacan:

- Escoger el tipo de los procesadores que van a conformar las CPUs que se van a utilizar y su velocidad de procesamiento.
- La cantidad de memoria RAM, tanto para el servidor como para los nodos
- Capacidades de Disco Duro, para el servidor y clientes o clientes sin disco.
- Tipo de interconexión entre los nodos y el servidor.

En general, si se puede contar con el tipo de recursos que uno desea, sin importar el costo, entonces se piensa en la aplicación en que se va a trabajar y tomando esto como base se decide cuáles de las características antes mencionadas merecen una prioridad mayor, por ejemplo; puede que la aplicación necesite más memoria que disco duro o viceversa.

También existen casos en que no se cuenta con recursos muy amplios, por lo que el cluster se construye basándose en el equipo existente, y es en la parte de programación donde se puede o no ajustar la aplicación al sistema (dependiendo de la aplicación)[10].

2.1.2. Requerimientos mínimos

CPU

Básicamente un nodo deberá tener como mínimo, una CPU 386 y una tarjeta madre compatible con Intel. Existe la posibilidad de trabajar con otras arquitecturas que soporten Linux, como es SPARC. Los procesadores Intel 386 funcionarían pero no tendrían un buen desempeño, debido a que sus velocidades de procesamiento están muy por debajo del requerido.

Memoria

Los requerimientos de memoria se deciden dependiendo del procesamiento de datos en la aplicación. Para cumplir con los requerimientos mínimos del sistema operativo y las aplicaciones cada nodo deberá contar al menos 8MB de memoria RAM.

Disco duro

Usando un espacio de disco centralizado, los nodos se pueden inicializar desde un floppy, un disco duro de poca capacidad o un sistema de archivos compartidos a través de la red. Entonces los nodos pueden acceder a su partición raíz desde un servidor de archivos a través de la red, normalmente usando el Network File System (NFS). Esta configuración funciona mejor en un entorno con un gran ancho de banda que agiliza la conexión y que permita un buen rendimiento en el servidor de archivos. Para un mejor rendimiento bajo otras condiciones, cada nodo tendría que tener suficiente espacio en el disco local para el sistema operativo, la memoria

virtual y los datos. Cada nodo debería tener como mínimo de 200 MB de espacio en disco.

Red de interconexión

Como mínimo cada nodo tiene que incluir una tarjeta de red Ethernet o Fast Ethernet. Alternativamente, interconexiones de más alto rendimiento, incluyendo las de los tipos Gigabit Ethernet y Myrinet, podrían ser usadas en conjunto con CPU's más rápidas.

Requerimientos del servidor

El servidor es recomendable que tenga recursos mayores que los nodos, además de que es el único que contará con un teclado, monitor, mouse (opcional) así como otros dispositivos que se consideren necesarios[10].

2.1.3. Implementación

Tomando en cuenta los recursos con que se contaban en ese momento y considerando los parámetros de diseño mencionados anteriormente, se prosigue a la implemetación del ***S**istema Beowulf* sin disco.
A continuación se presentan los puntos más representativos de la implementación.

2.1.4. Configuración del servidor y nodos

Antes de instalar el Sistema Operativo se tiene que planificar una serie de aspectos para poder contestar a todas las preguntas durante la instalación. Para realizar esta planificación podemos basarnos en lo siguiente:

- Medios con los que se cuenta, básicamente el hardware con el que contamos. En esta versión de Linux Red Hat 6.2 con el kernel 2.2.25 puede que se detecte automáticamente todos los periféricos sin necesidad de pedirnos información adicional, pero en algunos casos no, y es necesario especificarlos.

- Se debe tener a la mano las especificaciones relativas a:

  ♢ Modelo de la tarjeta de video (SIS 630) y memoria que contiene (8 MB).

- ♢ Tipo de monitor (DELL) y frecuencias de barrido horizontal (31.5) y vertical (60 Hz).
- ♢ Modelo de dispositivos y controladores, del ratón, tarjeta de red (2 tarjetas 3Com) y, dispositivos hardware que tengamos y que se desee que funcionen en Linux en la computadora servidor.
- ♢ Nombre que se va a poner al sistema (scluster).
- ♢ Qué software y servicios se van a instalar.

PVM-3.4.3-4

MPICH-1.2.5

LAM-6.3.1-4

- Definir cómo se va a distribuir el disco ya que al momento de la instalación se requiere y es mejor planearlo con anticipación. Se recomienda dejar el mayor espacio posible a (/) raíz, aquí se creará el sistema de archivos para los nodos y en segundo lugar a la partición /usr, donde se instalarán las herramientas o software de paralelización como PVM.

En nuestro caso, como contamos con un disco de 30 GB, la distribución recomendada sería :

Dispositivo	Punto de montaje	Tamaño
/dev/hda1	/boot	250 MB
/dev/hda2	/	18 GB
dev/hda3	Extended	11.2 GB
/dev/hda5	Swap	600 MB
/dev/hda6	/usr	8.5 GB
/dev/hda7	/var	1 GB
/dev/hda8	/tmp	950 MB

Después de haber planeado lo anterior se procede a la instalación del Sistema Operativo Linux Red Hat en la computadora que va a ser nuestro servidor.

Ya instalado el sistema operativo se procede a la configuración del equipo, es decir, configuración de los ambientes de red y los diferentes servicios que prestará nuestro servidor a los nodos.

2.1.5. Configuración inicial

Las configuraciones siguientes son para la máquina servidor.

1. Configuración de la(s) tarjeta(s) de red.

   a) Editar el archivo */etc/conf.modules*

   Ejemplo: Con una tarjeta NE2000 o compatible se puede utilizar la siguiente configuración:

   alias eth0 ne
   options ne io=0x300 irq=5

   Para nuestro caso, utilizamos 2 tarjetas 3Com (3c509) en el servidor con la siguiente configuración.

   alias eth0 3c59x

   En este caso, no fue necesario declarar la línea marcada como options, pero en el archivo */etc/lilo.conf* se agrego la siguiente línea *append = "ether=5,0, eth0 ether=11,0, eth1´´*

   Se prueba que la tarjeta funcione correctamente con el comando **modeprobe**

   #modeprobe 3c59x

b) Una vez probadas y configuradas las tarjetas, en el directorio:
/etc/sysconfig/network-scripts
debe existir un archivo con el nombre *ifcfg-ethx*, donde se darán de alta las características propias de cada tarjeta.

Si el archivo no existe, se crea y para que el sistema reconozca la tarjeta, se utiliza el comando:

#ifup ethx

Donde x es el número del adaptador de red. En nuestro caso, al utilizar una tarjeta para la red interna del cluster, y otra para la salida hacia el exterior, se crearon 2 archivos: ifcfg-eth0 e ifcfg-eth1

Donde ifcfg-eth1 es el archivo asociado a la tarjeta que proporciona la salida al exterior, y ifcfg-eth0 es el archivo asociado a la tarjeta para la red interna.

A continuación se muestra el contenido de los archivos, para nuestras tarjetas.

ifcfg-eth1:
DEVICE =“eth1”
IPADDR=“192.168.0.3”
NETMASK=“255.255.255.0”
NETWORK=“192.168.0.0”
BROADCAST =“192.168.0.255”
GATEWAY = “192.168.0.254´´
BOOTPROTO = none TYPE = Ethernet ONBOOT= yes
USERCTL = ’yes’

ifcfg-eth0:
DEVICE=“eth0”
IPADDR=“192.268.10.1”
NETMASK= “255.255.255.0”

NETWORK=“192.168.10.0”
BROADCAST =“132.248.54.255”
BOOTPROTO = static
ONBOOT= yes

Para mostrar cómo quedaron configuradas cada una de las tarjetas utilizamos el siguiente comando:
#ifconfig

Para verificar si la tarjeta responde, se utiliza el comando ping para probar la respuesta de la tarjeta.
#ping eth0

2. En el archivo */etc/sysconfig/network* podremos configurar el nombre de la máquina, el dominio y la puerta de enlace (gateway):

   Ejemplo :

   NETWORKING=yes
   HOSTNAME= “scluster”
   GATEWAY=“192.168.0.1”
   FORWARD_IPV4=“yes”
   NISDOMAIN= “scluster”

3. Editar el archivo */etc/hosts*. En este archivo se colocarán la dirección IP, el nombre de la máquina con su dominio, y el alias por el cual se conocerá a la máquina. En los sitios pequeños sin acceso a Internet, es razonable que cada máquina mantenga su propia copia de la tabla con los nombres de las máquinas de la red local con sus correspondientes direcciones IP. Esta tabla se guarda en el archivo */etc/hosts*.

   Ejemplo:

127.0.0.1 localhost.localdomain localhost
192.168.10.1 scluster scluster

4. En el archivo /etc/resolv.conf se define el dominio y la dirección del servidor DNS.

   En nuestro caso el nombre de dominio externo es fi-a.unam.mx. La dirección del servidor DNS, que se utiliza es: 132.248.10.21 132.248.1.31 132.248.204.1

   Ejemplo: search fi-a.unam.mx dgsca.unam.mx cluster.beowulf
   nameserver 132.248.10.2
   nameserver 132.248.204.1

5. Por último se define el nombre del host tanto en el archivo */etc/sysconfig/network* como en el archivo */etc/HOSTNAME*.

2.1.6. Etherboot

Como los nodos esclavos no tienen disco duro es necesario hacer uso de ciertos servicios y utilerías para hacerlos funcionar. El arranque de los nodos por medio de la red se puede conseguir mediante el paquete de distribución gratuita Etherboot. Para empezar se debe comprobar en la documentación del paquete que la tarjeta de red a utilizar esté incluida para que soporte Etherboot. El programa que genera este paquete se puede grabar en una EPROM (Erasable Programmable Read Only Memory -Memoria de sólo lectural programable y borrable), aunque también funciona al simular el código que debería estar en un disco flexible, que es la opción que vamos a utilizar.

Para crear el disco de inicio es necesario colocar un bloque especial de inicio que se proporciona en la distribución. Este pequeño programa de 512 bytes carga en memoria los bloques del disco que lo siguen y comienza la ejecución. Por lo tanto, para construir este disco únicamente hace falta concatenar el bloque de inicio con el binario de Etherboot que contenga el controlador de la tarjeta de red que se tiene.

Un ejemplo podría ser: *#cat floppyload.bin 3c509.lzrom /dev/fdO*

En la versión de Etherboot que se manejó para este trabajo, esto se automatiza mediante un shell script (guión de comandos) que viene incluido con el paquete. De esta manera, ahora se crea la imagen para el disco al teclear el comando:

#make bin32/identificador_tarjeta.fdO

Cambiarse a directorio */tftpboot/etherboot-5.0.6/src* en éste directorio ejecutar el comando

#make

#make bin32/3c509.fd0

Antes de poner el disco de arranque en red, es necesario configurar tres servicios en Linux: DHCP, TFTP y NFS. Los servidores de DHCP y NFS se explican más adelante en cuanto a funcionamiento y configuración.

Para la instalación del Etherboot se crea un directorio en raíz (/) llamado */tftpboot*. Dentro de este directorio se instalará el Etherboot. Para crear los discos de inicio para las tarjetas de red, dentro del directorio Etherboot-versión/src se teclea :

Para la tarjeta del red tipo 3c509

/etherboot-version/src)$make bin32/3c5O9.fdO

Esto enviará al disco flexible, la información necesaria para que inicie la tarjeta de red en el nodo y comience a buscar su dirección IP en el servidor DHCP , solicitando que se le envíe la imagen de arranque del sistema operativo con el que va a trabajar.

Servidor TFTP

Para poder lograr que los nodos obtengan su kernel, es necesario usar un servidor TFTP, comúnmente viene con las distribuciones de Linux, aunque también se puede conseguir el paquete para compilarlo. Normalmente, se lanza el demonio *tftpd* desde *inetd* con una línea como ésta

tftp dgram udp wait root/usr/sbin/tcpd in.tftpd /tftpboot

en el archivo de configuración /etc/inetd.conf.

Si este servicio se encuentra en el archivo /etc/inetd.conf, se remueve el símbolo # de la línea que hace referencia a este servicio.

Una vez configurado, para no reiniciar la máquina se reinicia el servicio de inetd por medio de la orden /etc/rc.d/init.d/inet restart.

Arranque de los nodos por red

Ahora se procede a arrancar el nodo con el disco que contiene la imagen del controlador de la tarjeta de red. Se debe detectar la tarjeta Ethernet y lanzar una petición de BOOTP en forma de broadcast. Si todo va bien, el servidor DHCP responderá al nodo con la información requerida, aunque al no existir el archivo

tftpboot/imagen-transportable_Kernel

fallará en cuanto intente cargar el mismo. Para lograr esto se necesita compilar un núcleo especial, uno que tenga la opción de montar el sistema de archivos raíz vía NFS activado. También hace falta habilitar en el núcleo la opción de obtener la dirección IP desde la respuesta BOOTP original (RARP).

La imagen del Kernel (bzlmage) que se crea al terminar la compilación del nuevo kernel no se puede enviar directamente. Debe ser convertida en una imagen marcada. Ésta es una imagen normal del núcleo con una cabecera especial que le dice al cargador de arranque en red dónde han de almacenarse los bytes en memoria y en qué dirección empieza el programa. Para crear esta imagen marcada se usa un programa llamado mknbi-linux. Después de crear la imagen, habrá que colocarla en el directorio /tftpboot con el nombre especificado en los archivos de configuración del DHCP.

La nueva imagen creada del Kernel después de la compilación está en

/usr/src/Linux/arch/i386/boot, y se copia al directorio /tftpboot.

Dentro del mismo se ejecuta el comando

#/usr/local/lib/mknbi/mknbi –target=linux –output=/tftpboot/vmlinuz-prueba.07 /usr/src/linux/arch/i386/boot/bzImage

Con el cual se hace transportable el Kernel, permitiendo el inicio remoto de los nodos.

Al volver a arrancar el nodo con el disco de inicio, ahora se debe llegar hasta cargar la imagen del núcleo y empezar a ejecutarlo. El arranque continuará hasta el punto en que intenta montar un sistema de archivos desde la raíz. En este punto, el servidor NFS proporciona los sistemas de archivos a exportar que serán montados.

El núcleo de Linux preparado para los nodos espera encontrar sistema de archivos raíz en /tftpboot/"nombre del nodo", por ejemplo: /tftpboot/nodo10

Al volver a arrancar el nodo se deberá ver cómo el núcleo monta un sistema de archivos raíz y sigue arrancando hasta presentar el login de usuario. Al inicio del nodo, habrá configuraciones que no correspondan al nodo, y mediante la adecuada sintonización de los nodos se corrige este problema.

Sistema de archivos para los nodos

Para la creación del sistema de archivos se utilizó el siguiente script proporcionado por el paquete Etherboot.

```
#!/bin/sh
if [ $# != 2 ]
then
echo Usage: $0 directorio_anterior nuevo_directorio
exit 1
fi
cd /tftpboot
if [ ! -d$1 ]
```

```
then
echo $1 no es un directorio
exit 1
fi
umask 022
mkdir -p $2
# solamente crea estos
for d in home mnt proc tmp usr
do
mkdir $2/$d
done
chmod 1777 $2/tmp
touch $2/fastboot
chattr +i $2/fastboot
# liga estos otros
for d in bin lib sbin
do
(cd $1; find $d -print I cpio -pl../$2)
done
# hace la copia de estos directorios
for d in dev etc root var
do
cp -a $l/$d $2
done
cat  EOF
Ahora, en /tftpboot/$2/etc, hay que editar
sysconfig/network
sysconfig/network-scripts/ifcfg-eth0
fstab (tal vez)
conf.modules (tal vez)
y configurar
```

rc.d/rc3.d
EOF

En éste trabajo se instalo un disco duro en un nodo y después se instalo el sistema operativo para ejecutar el script anterior.

2.1.7. Configuración del kernel para los nodos

Para tener acceso a la configuración del Kernel, utilizaremos la opción del menú gráfico, al cual se accede por medio del comando:

$make xconfig; dentro del directorio /usr/src/Linux-2.2.25

Las opciones que son indispensables que estén presentes en el Kernel que se van a dar a los nodos son las siguientes:

- Network device support
  - Ethernet 10 or 100 Mbit
    - 3COM card

      Se cargan todas en el Kernel, no como módulos.
  - Otras Tarjetas lSA
    - NE2000/NE1000 support
    - PCl NE2000

      En el cluster se utilizan tarjetas 3Com, por lo que se activa su soporte dentro del Kernel.

      Se recomienda activar la mayoría de las tarjetas más utilizadas.
- Networking Options
  - TCP/IP Networking
  - lp: Kernel level autoconfiguration
    - BOOTP support

- RARP support

  Estas opciones proveen soporte para la pila de protocolos de TCP/IP y el soporte para poder iniciar de manera remota los clientes.

- File System
  - Network File System support
  - Root file system on NFS

Proporciona soporte para compartir el sistema de archivos vía NFS por medio de la red y que inicie el sistema de archivos como root. Se salva la configuración y se sale del menú. Para compilar el Kernel, en la línea de comandos se escriben las siguientes instrucciones:

$make dep;make clean;make bzlmage

Ya se tiene la imagen del sistema en /usr/src/Linux/arch/i386/boot con el nombre de bzImage.

2.1.8. Configuración servidor NFS

El primer paso es crear el archivo */etc/exports*, si no existe. Este archivo define qué partes del disco del servidor se comparten con el resto de la red, y las reglas mediante las cuales se comparten, es decir, lista las particiones compartidas, las máquinas con las que los comparte y con qué permisos.

Para la configuración de nuestro servidor NFS necesitamos:

- Exportar o compartir los directorios */usr, /home, /tftpboot*
- Los nombres de las máquinas cliente NFS, deben ser todas aquellas máquinas que pertenezcan a la red 192.168.10.0 con máscara 255.255.255.0
- Y los permisos que se necesitan básicamente son rw y no_root_squash.

Por lo que el archivo */etc/exports* de nuestro servidor es el siguiente:

/tftpboot 192.168.10.0/255.255.255.0(rw,no_root_squash)
/home 192.168.10.0/255.255.255.0(rw,no_root_squash)
/usr 192.168.10.0/255.255.255.0(rw,no_root_squash)

Después es necesario que el servidor NFS lea el archivo, para ello debemos iniciar o reiniciar el servicio de NFS, ya sea reiniciando la máquina o por medio de la línea

#/etc/rc.d/init.d/nfsd start o restart

según sea el caso.

Configuración del cliente

Los clientes NFS son fáciles de configurar bajo Linux debido a que no requieren ningún software nuevo o adicional para cargarse. El único requerimiento es que el Kernel sea compatible con el soporte NFS, y como al compilar el nuevo Kernel para los nodos se activó esta opción entonces la imagen del Kernel para nuestros nodos cumplen con este requerimiento.
Para montar un sistema de archivos, es usando el comando *mount* y por medio del archivo */etc/fstab*, el cual permite que se monten los sistemas de archivos remotos al arranque (automáticamente).

En el archivo *fstab* del nodo se declaran las siguientes líneas.

192.168.10.1:/tftpboot/		nfs	defaults	0 0
192.168.10.1:/usr	/usr	nfs	defaults	0 0
192.168.10.1:/home	/home	nfs	defaults	0 0
none	/proc	proc	defaults	0 0
none	/dev/pts	devpts	gid=5,mode=620	0 0

2.1.9. Configuración del servidor DHCP

El archivo de configuración primario del servidor DHCP es por defecto /etc/dhcpd.conf. Este archivo de configuración es un archivo de texto y tiene la siguiente estructura:

Parámetros globales;
Declaración1 {
parámetros relacionados con declaración1
Subdeclaración anidada
}
Declaración2 {
parámetros relacionados con declaración2
Subdeclaración anidada
} ...

El archivo de configuración *dhcpd.conf* para el Cluster es el siguiente:

```
    server-identifier scluster;
shared-network SCLUSTER {
    subnet 192.168.10.0 netmask 255.255.255.0 {
        option broadcast-address 192.168.10.255;
    }
}
subnet 192.168.0.0 netmask 255.255.255.0
    host nodo10 {
    fixed-address 192.168.10.10;
    hardware ethernet 00:10:4B:35:74:A6;
    filename "/tftpboot/vmlinuz-prueba.07";
    next-server 192.168.10.1;
    option host-name "nodo10";
}
host nodo11 {
```

fixed-address 192.168.10.11;
hardware ethernet 00:10:4B:35:75:89;
filename "/tftpboot/vmlinuz-prueba.07";
next-server 192.168.10.1;
option host-name "nodo11";
}
...

Es importante que cada vez que se cree un nuevo nodo se agregue otra entrada host con los datos propios del nodo en este archivo.

Configuración DHCPD nodos(cliente)

Para configurar los nodos hay que abrir el archivo

/tftpboot/nodo-n/etc/sysconfig/network-scripts/ifcfg-ethO

En lugar de proporcionar la dirección IP se especifica que ésta será proporcionada por un servidor DHCP, esto lo logramos modificando la línea
IPADDR="dhcp"

2.1.10. Configuración de red en el nodo

Como la configuración de red se hereda del servidor, es necesario ajustarla a las necesidades del nodo, por lo que es necesario modificar los siguientes archivos según lo visto anteriormente:

/tftpboot/nodo/etc/sysconfig/network:

NETWORKING=yes
HOSTNAME=""
GATEWAY="192.168.10.0"
NISDOMAIN= "scluster"

/tftpboot/nodo/etc/sysconfig/network-scripts/ifcfg-eth0:

DEVICE= "eth0"
IPADDR = "dhcp"
NETMASK= "255.255.255.0"
ONBOOT="yes"
BOOTPROTO="none"
BROADCAST=192.168.10.255
NETWORK=192.168.10.0
USERCTL=no

y para que inicie el nodo en modo texto modificar en el archivo

/tftpboot/nodo/etc/inittab

la línea que da el arranque por default.

 # Default runlevel. The runlevels used by RHS are:
 # 0 -halt (Do NOT set initderault to this)
 # 1 -Single user mode
 # 2 -Multiuser, without NFS (The same as 3, ir you do not have networking)
 # 3 -Full multiuser mode
 # 4 -unused
 # 5-Xll
 # 6 -reboot (Do NOT set initdefault to this)
#
id: 3: initdefault:

2.1.11. Configuración del servidor NIS

Las distribuciones de Linux normalmente vienen con NIS ya compilado e instalado. En ese caso sólo se deberá de habilitar el servicio y asegurarse que *ypserv* es

parte del proceso de arranque de los niveles de ejecución 3 y 5.

Una vez que se ha instalado NIS, se necesita configurarlo. Este proceso se puede dividir en cuatro pasos:

- I. Establecer el nombre del dominio
- II. Arrancar el demonio *ypserv* que inicia NIS
- III. Editar el archivo Makefile
- IV. Ejecutar *ypinit* para crear la base de datos

I. Establecer el nombre del dominio

La forma de definir un dominio NIS es con el comando *domainname*:

#domainname scluster

Donde "dominio"es el dominio NIS empleado. Es necesario que cada vez que se arranque el sistema, el dominio se establezca. Se puede añadir la siguiente línea al archivo */etc/sysconfig/network*:

NISDOMAIN = scluster

La configuración del nombre de dominio debe de ocurrir antes que se inicien los servicios del NIS. El nombre del dominio que utilizamos lleva la denominación scluster.

II. Arranque de ypserv

El programa *ypserv* es el responsable del manejo de las peticiones que se hacen hacia el servidor NIS. Para no reiniciar el servidor, es posible reiniciar el demonio de *ypserv* por medio de los comandos :

$/etc/rc.d/init.d/ypserv start

III. Edición del archivo Makefile

Para que el NIS funcione debidamente, es necesario crear los mapas de los cuales se servirá para realizar su función. Para esto, en el directorio */var/yp* se

encuentra un archivo llamado *Makefile*. Este archivo lista los archivos que se compartirán mediante NIS, así como algunos parámetros adicionales sobre cómo compartirlos.
Algunas de las opciones que se presentan dentro de este archivo son:

UID Y GID mínimos.

En esta sección, se pueden delimitar los UID y GID mínimos que se manejarán dentro de los mapas de NIS.

 # We do not putpassword entries with lower UIDs (the root and system
 # entries) in the NIS password database, for security. MINUID is the
 # lowest uid that will be included in the password maps.
 # MINGID is the lowest gid that will be included in the group maps.
MINUID=500
MINGID=500

Mezcla de contraseñas ocultas con reales, para grupos

Si usa contraseñas *shadow* (shadow password), NIS automáticamente lo manejará tomando el campo encriptado del archivo */etc/shadow* y mezclándolo con la copia compartida de NIS del archivo */etc/passwd*. A menos que haya una razón específica por la que no se quiera activar la compartición de contraseñas encriptadas, MERGE_PASSWD se dejará a su valor por defecto. En nuestro caso, esta opción se dejó en su valor original.

 # ShouId we merge the passwd file with the shadow file?
 # MERGE_PASSWD=true/faIse
MERGE_PASSWD=true

Mezcla de contraseñas de grupo ocultas con grupos reales

De la misma forma en que el archivo /etc/passwd permite contraseñas ocultas (/etc/shadow), el archivo /etc/group permite contraseñas de este tipo. Si se tiene un archivo shadow para las contraseñas de grupo, será necesario configurar MERGE_GROUP al valor de false. En nuestro caso, no es necesario utilizar el *shadow* para los grupos, así que su valor se deja en false.

 # ShouId we merge the group file with the gshadow file?
 # MERGE_GROUP=true/faIse
MERGE_GROUP=faIse

Designación de nombre de archivos

Dentro del Makefile se listan los archivos que se preconfigurarán para compartirse mediante NIS. Sin embargo, que se listen aquí no significa que se compartirán automáticamente. Este listado simplemente establece los nombres de archivos que más tarde se usarán al ejecutar el Makefile.

Parte de archivo *Makefile* es el siguiente:

 # These are the source directories for the NIS files; normaly
#that is /etc but you may want to move the source for the password
 # and group files to (for example) /var/yp/ypfiles. The directory
 # for passwd, group and shadow is defined by YPPWDDIR, the rest is
 # taken from YPSRCDIR.
#
YPSRCDIR = /etc
YPPWDDIR = /etc
YPBINDIR = /usr/lib/yp
YPSBINDIR = /usr/sbin

```
YPDIR = /var/yp
YPMAPDIR = $(YPDIR)/$(DOMAIN)
# These are the files from whJah the NIS databases are built. You may edit
# these to taste in the event that you wish to keep your NIS source files
# seperate from your NIS server's actual configuration files.
#
GROUP = $(YPPWDDIR)/group
PASSWD = $(YPPWDDIR)/passwd
SHADOW = $(YPPWDDIR)/shadow
GSHADOW = $(YPPWDDIR)/gshadow
ADJUNCT = $(YPPWDDIR)/passwd.adjunct
#ALIASE = $(YPSRCDIR)/aliases# aliases could be in /etc or /etc/mail
ALIASES = /etc/aliases
ETHERS = $(YPSRCDIR)/ethers # ethernet addresses (for rarpd)
BOOTPARAMS = $(YPSRCDIR)/bootparams # for booting Sun boxes (bootparamd)
HOSTS = $(YPSRCDIR)/hosts
NETWORKS = $(YPSRCDIR)/networks
PRINTCAP = $(YPSRCDIR)/printcap
PROTOCOLS = $(YPSRCDIR)/protocols
PUBLICKEYS = $(YPSRCDIR)/publickey
RPC = $(YPSRCDIR)/rpc
SERVICES = $(YPSRCDIR)/services
NETGROUP = $(YPSRCDIR)/netgroup
NETID = $(YPSRCDIR)/netid
AMD_HOME = $(YPSRCDIR)/amd.home
AUTO_MASTER = $(YPSRCDIR)/auto.master
AUTO_HOME = $(YPSRCDIR)/auto.home
YPSERVERS = $(YPDIR)/ypservers # List of all NIS servers for a domain
```

Lo que se comparte

Al final de la parte de configuración del archivo Makefile, existe una entrada que lleva la etiqueta *all*. Después de esta etiqueta se mencionarán todos los mapas que se comparten. Si no se quiere compartir algunos mapas, basta con anteponer un signo de # antes del nombre del mapa:

#If you don 't want some of these maps built, feel free to comment
 # them out from this list.
all: passwd group hosts rpc services netid protocols netgrp mail
 # shadow publickey # networks ethers bootparams printcap
 # amd.home auto.master auto.home passwd.adjunct

En nuestro caso, se eliminaron los servicios **netgrp** y **mail**, ya que no son necesarios.

IV. Uso de *ypinit*

Una vez que se tenga preparado el archivo *Makefile* se inicializa el servidor **yp** (NIS) usando el comando *ypinit*:

/usr/lib/yp/ypinit -m

donde la opción -m le dice a *ypinit* que configure el sistema como servidor de NIS maestro. Ejecutándolo sobre el dominio scluster.

Configuración de un cliente NIS

La configuración de los clientes es más sencilla que la del servidor. Los pasos a seguir en general serían los siguientes:

1. Editar el archivo /etc/yp.conf
2. Configurar los servicios de arranque

1. Edición del archivo /etc/yp.conf

El archivo /etc/yp.conf contiene la información necesaria para que el demonio del cliente, *ypbind*, arranque y encuentre el servidor NIS. El cliente puede encontrar al servidor de diferentes maneras:

- ♢ El uso de broadcast
- ♢ La especificación de nombre de la máquina del servidor.

La configuración por broadcast es apropiada para cuando un cliente se mueve a varias subredes. En nuestro caso, las máquinas van a estar en una misma red y no va a haber desplazamiento del cliente. La otra opción busca de manera ya predeterminada el servidor que le dará servicio de NIS. Para utilizar esta última configuración, se edita el archivo /etc/yp.conf y se le agrega la siguiente línea:
ypserver nombre-servidor

Ejemplo:

 # /etc/yp,conf -ypbind configuration file
 # Valid entries are
#
#domain NISDOMAIN server HOSTNAME
 # Use server HOSTNAME for the domain NISDOMAIN.
#
#domain NISDOMAIN broadcast
 # Use broadcast on the local net for domain NISDOMAIN
#
#ypserver HOSTNAME
 # Use server HOSTNAME for the local domain. The
 # IP-address of server must be listed in /etc/hosts.
#
ypserver 192.168.10.1

Donde nombre-dominio es el nombre de su dominio de NIS y nombre-servidor es el nombre de su servidor NIS al que hace referencia el cliente.

2. Configurar los servicios de arranque

El *ypbind* es el demonio que corre en los clientes NIS, pero no debemos olvidar que todo servidor NIS es cliente, porque necesita acceder a la información de los mapas, así que todas las máquinas del dominio deben correr el *ypbind*. Este demonio determina a qué servidor debe conectarse la máquina cliente para realizar sus peticiones de información.

Normalmente, éste se inicia en el script de arranque */etc/rc.d/init.d/ypbind*. Se debe de comprobar que se inicia este servicio tanto en el nivel de inicio 3 como el nivel de inicio 5.

Configuración de mapas

Una vez que se tienen configurados el servidor y el cliente, se puede probar que funcionan correctamente por medio de la utilidad *ypcat*, que mostrará a pantalla el mapa que se comparte. Para esto se debe escoger un mapa que se comparta en el dominio, como por ejemplo, *passwd*.

Una vez que se tiene todo esto configurado, al dar de alta a un usuario nuevo, se hará directamente desde el servidor, y ya no nodo por nodo. Para actualizar todos los mapas con los nuevos datos, sólo se deberá ejecutar *make* dentro del directorio */var/yp* con lo que se actualizarán todos los mapas que se comparten.

2.1.12. Sintonización de los nodos

Eliminación de servicios no utilizados

En el nodo, dentro de su directorio */etc/rc.d/rc3.d* se deshabiltan los servicios que no son necesarios en este nivel de ejecución del sistema. Esta eliminación se

hace de acuerdo a las necesidades que se tengan y dejando sin cambio los servicios básicos del sistema.

En el directorio ya mencionado, aparece una lista de enlaces hacia diferentes programas que se ejecutan. Los servicios que están activados se identifican por una **S** mayúscula al inicio de su nombre, los servicios que se desactivan se le cambian en su nombre la **S** por una **K** al inicio. Así, al iniciar el sistema, sólo los servicios que inician con una **S** son los que se ejecutan.

La eliminación se lleva a cabo tomando en cuenta aquellos servicios que no se utilizan en el nodo, y sólo ocupan los recursos del mismo, siendo éstos más necesarios para el cómputo. Algunos de los servicios eliminados son :

Ipd -servicio de impresión
xfs -servicio para la consola gráfica
apm -administración de energía, etc.

Módulos del sistema

Debido a que el Kernel de Linux, en la plataforma que se está trabajando (Intel), puede trabajar de manera modular, es posible que al compilar un nuevo Kernel no todos los módulos que se encuentra instalados en el servidor serán usados. Al iniciar el sistema del nodo, lista los módulos que el Kernel no puede resolver; por lo que, para eliminar la presencia de esta lista será necesario eliminarlos o removerlos del directorio */lib/modules/version_Kernel/* del nodo. Con la ayuda del comando *depmod*, que muestra los nombres de los módulos que no se pueden resolver, es posible ir removiendo los módulos no utilizados junto con el comando *rm*.

Duplicación de los sistemas de archivos para los nodos

Para que se puedan agregar más nodos al cluster, una vez que se han sintonizado los nodos, se procede a realizar la copia del sistema de archivos que se ha creado para el primer nodo. Para ello se ejecuta el script que se muestra a continuación.

#!/bin/sh

```
if [ $# != 2 ]
then
echo Usage: $0 directorio_anterior nuevo_directorio
exit 1
fi
cd /tftpboot
if [ ! -d  1]
then
echo$1noesundirectorio
exit1
fi
umask022
mkdir − p$2
#solamentecreaestos
fordinhomemntproctmpusr
do
mkdir$2/ d
done
chmod 1777 $2/tmp
touch $2/fastboot
chattr +i $2/fastboot
# liga estos otros
for d in bin lib sbin
do
(cd $1; find $d -print I cpio -pl ../$2)
done
# hace la copia de estos directorios
for d in dev etc root var
do
cp -a $l/$d $2
done
```

A continuación,
en /tftpboot/$2/etc, hay que editar
sysconfig/network
sysconfig/network-scripts/ifcfg-eth0
fstab (tal vez)
conf.modules (tal vez)
y configurar

rc.d/rc3.d

Se agrega el nombre del nodo creado y su dirección IP en el archivo */etc/hosts* y */tftpboot/nodo/etc/hosts*, para que sea reconocido por su nombre y su alias cada nodo.

2.1.13. Sintonización del servidor

El servidor sólo requiere sintonización para cuestiones de seguridad. De esta forma se cierran los puertos no utilizados y considerados inseguros. Para poder cerrarlos, en el archivo */etc/services* se le antepone un # para poder cancelar los servicios.

2.1.14. Instalación de PVM y MPI

Al realizar la instalación del sistema operativo, entre los paquetes que se tienen dentro de las opciones para instalar, está PVM. Para verificar que el paquete ya está instalado, se puede utilizar el comando

#rpm -qi pvm

Si no está el paquete, instalarlo.Una vez instalado sólo hay que agregar las siguientes líneas en el archivo */etc/skel/.bashrc*

export PVM_ROOT =/usr/share/pvm3
export PVM_ARCH=LINUX

Al crearlo en */etc/skel/.bashrc* se permite que al agregar más usuarios al sistema, estas variables se copien a su directorio *home*.

Si se tiene cuentas ya creadas, se agregan estas líneas al *.bashrc* de cada usuario.

Para que funcione el PVM, es necesario que se puedan ejecutar comandos de manera remota por medio del comando *rsh*. Para esto se agrega en el directorio */etc* el archivo *hosts.equiv*, que contiene los nombres y direcciones de las máquinas en las cuales puede ejecutarse comandos de manera remota. De esta manera, cada vez que se agrega un nuevo nodo habrá que agregar su nombre y dirección IP a este archivo.

Para ver más detalles de los ejemplos y de la aplicación real, se puede revisar la tesis del 2001 [10][11]

2.2. Estado actual del Cluster

El Sistema Beowulf de la Facultad de Ingeniería está siendo analizado tanto en software como en hardware para su máximo desempeño, así mismo, se está haciendo una trabajo de investigación para su adecuada administración.

2.2.1. Características del hardware

Quizá la restricción más grande que se tuvo en este proyecto fue, precisamente el hardware sobre el cual se va a construir el cluster. Por tratarse de un proyecto de naturaleza académica, en el cual se busca probar las teorías y técnicas de los clusters para obtener un rendimiento mayor al de soluciones uniprocesador, no se contó con presupuesto para adquisición de equipo. Se tuvo, entonces, que reunir equipo que estaba en desuso, evaluar las características y posibilidades del mismo, y de acuerdo a esto generar una configuración tanto en software como en hardware que permitiera contar, finalmente, con un cluster utilizable como plataforma de cómputo.

Para la construcción del cluster se consiguió, en primer lugar, un equipo AMD Athlon de 850 MHz con 128 MB RAM disco duro de 30 GB y dos tarjetas de red. Este equipo se designó como el nodo central del cluster. Desde este equipo los usuarios crearán y ejecutarán sus programas. Se cuenta con dos tarjetas de red para que una de ellas proporcione acceso a la red pública, y la otra esté dedicada exclusiva-

mente a la red de interconexión del cluster; la red de comunicaciones dedicada es una característica importante de los cluslers tipo Beowulf.

Adicionalmente se consiguieron los siguientes 14 equipos:

- 4 equipos HP Vectra 486/66. 12 a 16 MB RAM. tarjeta de red 3Com 3C509
- 7 equipos HP Vectra 486/50. 12 a 16 MB RAM. tarjeta de red 3Com 3C509
- 1 equipo HP Vectra 486/33. 12 a 16 MB RAM. tarjeta de red 3Com 3C509
- 1 equipos Dell Optiplex 486/66. 16 MB RAM, tarjeta de red 3Com 3C509
- 1 equipo Digital 486/33. 16 MB RAM. tarjeta de red 3Com 3C509

Estos equipos se utilizan como nodos o elementos de procesamiento en el cluster. El conocer las características de estos equipos definió la estructura del cluster así como parte de la configuración de software requerida para el mismo.

Comunicación entre nodos

Un elemento básico en todo cluster es la red o canal de comunicaciones entre nodos. Las características de los equipos con que se cuenta, definieron esta elección. Por el tipo de tarjetas de red con que se cuenta se decidió utilizar una red Ethernet de 10 Mbps para unir a los nodos entre sí. Se emplearon dos concentradores de 10 MBits y como cableado UTP categoría 5.

Desde el punto de vista del rendimiento una red Ethenet simple no es una muy buena elección para un cluster. Los tiempos de latencia son relativamente elevados, del orden de algunos milisegundos.

El ancho de banda de 10 Mbps es poco adecuado para aplicaciones que requieran un mayor nivel de comunicaciones entre nodos, y la arquitectura de bus proporciona un canal de comunicaciones compartido con retransmisión (broadcast) que tiende a saturarse muy rápidamente a medida que la cantidad de mensajes pasados entre nodos se incrementa.

Sin embargo, el uso de la red Ethernet tiene ciertas ventajas y características interesantes. Una de ellas es su facilidad de instalación y bajo costo, que en particular fueron claves para su elección en este proyecto dadas las restricciones con que se realizó.

Por otro lado, la popularidad de la tecnología Ethernet ha llevado a desarrollos que permiten incrementar el desempeño según crezcan las necesidades. Un cluster puede beneficiarse con el uso de switches, que segmentan el tráfico en el bus Ethenet y reducen la saturación y colisiones en el mismo y se puede contar con incrementos de desempeño inmediatos utilizando Fast Ethernet (100 Mbps) y Gigabit Ethernet (1 Gbps).

Red de interconexión

Tipo de Red: Área Local LAN
Tecnología: Ethernet
Esquema de direccionamiento: IP Clase C de tipo privado con la
dirección de red :192.168.10.0 y mascara /24 255.255.255.0
Dispositivo de interconexión: 2 Hubs de 8 y 12 puertos
Cables de interconexión: Par trenzado UTP Cat 5
Conectores RJ45

Consideraciones para equipos sin disco duro

Dado que los nodos de procesamiento no cuentan con disco duro, se decidió configurarlos como estaciones sin disco duro. El uso de estaciones diskless (sin disco), como se conocen comúnmente, está bastante difundido, pues permite un desempeño aceptable para terminales que normalmente fungen como despliegue del trabajo realizado en un servidor multiusuario. Las terminales diskless requieren un mínimo de trabajo de mantenimiento y configuración, y éstos se realizan básicamente en un servidor central, facilitando estas tareas. En este caso, se da un enfoque un tanto diferente. El recurso de interés en las estaciones es su procesador y memoria, como elementos de trabajo básicos del cluster. Adicionalmente, no se pretende que

los usuarios tengan aceso a estas estaciones directamente. La técnica de arranque diskless proporciona ventajas como son la centralización de todos los archivos de los nodos en un servidor central, y cierta economía en los requerimientos de equipo, pues se evita la necesidad de contar con disco duro en cada uno de ellos.

El uso de esta técnica es una extensión del uso del sistema de archivos por red (Network File System o NFS). NFS normalmente se emplea para compartir los directorios de usuarios en redes de estaciones de trabajo, y en clusters suele emplearse para facilitar la distribución de los programas a ejecutar. En nuestro caso los sistemas de archivos de los nodos reciden totalmente en el servidor central. El uso de esta técnica presenta dos desventajas básicas; la primera es que se incrementa el uso de disco duro en el servidor central. En la configuración final del cluster, se requieren aproximadamente 44 MB de espacio por cada nodo agregado; esto comprende los archivos que no pueden compartirse entre nodos y por lo tanto deben mantenerse separados, tales como directorios necesarios para el arranque y los archivos de configuración.

La segunda desventaja es un bajo desempeño en el acceso a archivos por parte de los nodos. Como los nodos no cuentan con almacenamiento secundario local, todo intento de acceso a disco se realiza a través de la red. Ya que en este caso no se cuenta con una red muy rápida, estos accesos pueden tomar bastante tiempo. El hecho de que el acceso a archivos es lento para los nodos debe tomarse en cuenta al momento de diseñar los programas a ejecutar en el cluster; se debe tener precaución con el acceso a archivos en los procesos que se ejecutan en los nodos. En general esta consideración en cuanto al desempeño del acceso a archivos tendrá un impacto que debe tomarse en cuenta, en el overhead de arranque de los procesos; si se diseñan cuidadosamente los programas, esto no repercutirá en el desempeño durante la realización de cálculos.

Integración de hardware

Tenido en cuenta los factores anteriores, la integración del hardware fue sencilla. El cluster se ensambló en instalaciones compartidas por el Laboratorio de Telemática y el Grupo de Usuarios de Linux de la Facultad de Ingeniería, en espacio

para tal efecto. De los 14 equipos con que se dispone, uno cuenta con gabinete estilo minitorre. Los restantes tienen gabinetes de tipo escritorio. Los nodos tipo escritorio se organizaron en dos "torres.ª cada lado del servidor central. El monitor y teclado del servidor se ubicaron al centro.

Los concentradores para la red Ethernet con forman la columna vertebral del mecanismo de interconexión. En este caso se requería un total de 16 puertos, por lo cual se utilizaron dos concentradores: uno de 12 puertos y otro de 8 puertos, los cuales se conectan en cascada, y en ellos se realiza la conexión de los nodos.

El servidor central cuenta con dos tarjetas de red, la que se designó para su uso en la red del cluster se conecta en estos concentradores. La otra tarjeta permite al servidor tener acceso a la red pública y se conecta en las facilidades provistas para ello por el Laboratorio. Para la conexión eléctrica se utiliza un regulador de corriente, al que se conectan tres multicontactos. Cada multicontacto cuenta con 6 enchufes polarizados y aterrizados, de manera que se tiene posibilidad de conectar los 14 nodos. El monitor del servidor central se conecta en el espacio restante en el regulador, que cuenta con 4 contactos.

2.2.2. Características del software

El software que se utilizó nos permite brindar los servicios necesarios del nodo Central hacia los demás nodos, así como la comunicación para establecer la máquina paralela.

Algunos de los servicios instalados son :

- NFS
- DHCP
- NIS
- SSH

El software que se utiliza para la programación en paralelo:

- PVM
- MPI/LAM
- MPICH

Capítulo 3
Administración de recursos hardware del sistema

La administración juega un papel muy importante en el *desempeño* de cualquier sistema. En un sistema de cómputo, la administración adecuada proporciona los lineamientos a seguir para un excelente *performance*.
En este capítulo se hablará de los lineamientos a seguir con respecto a la administración del hardware del *Sistema Beowulf*.

3.1. Características de los componentes

Una de las primeras tareas del administrador de un *Sistema Beowulf* comienza en el diseño del sistema. Se debe tener la capacidad de hacer un buen diseño con los componentes disponibles, sin perder de vista la escalabilidad.
Dependiendo de qué uso se le va a dar al cluster y sabiendo las características principales de cada uno de los componentes, se puede lograr un buen diseño.

3.1.1. En el nodo sin discos

En un diseño sin disco *(diskless)* la administración del hardware se centra en la memoria, tarjeta madre (motherboard), y el procesador de los nodos.
Las características principales son:

- Conocer la memoria
  - Latencia
  - Velocidad de acceso
  - Tipo: DDRAM, SDRAM, etc.
  - Capacidad
  - Paridad
- Conocer el procesador
  - Velocidad del bus interno

  - Memoria cache
  - Niveles de memoria cache
  - tipo: Intel, AMD, etc.

- Conocer la tarjeta madre
  - Velocidad del bus
  - Modelo
  - Tipo de socket: No 5, No 7, etc.

3.1.2. En el nodo con discos

Las características a considerar en este diseño son las mismas que sin disco, con excepción del disco duro.

- Conocer el disco duro
  - Velocidad de las cabezas de lectura
  - Capacidad
  - Tipo: SCSI, IDE, etc.

3.1.3. En el servidor

Se considera que las características de servidor deben ser mejores que las de los nodos, sin embargo, dependiendo de la aplicación que se corra en el cluster, los nodos podrían tener más memoria, un procesador más rápido o podrían ser iguales que el servidor.
Por ejemplo[15]:

- Hardware de los nodos.
  32 máquinas con la siguiente configuración:
  - 2 Pentium III 1 GHz Intel CPUs
  - Supermicro 370 DLE Dual PIII-FCPGA motherboard
  - 2 256 MB 168-pin PC133 Registered ECC Micron RAM

- 1 20 GB Maxtor ATA/66 5400 RPM HD
- 1 40 GB Maxtor UDMA/100 7200 RPM HD
- Asus CD-S500 50x CDROM
- 1.4 MB floppy drive
- ATI Expert 98 8 MB PCI video card
- Mid-tower case

- Hardware del servidor.
  un servidor externo con la siguiente configuración:
  - 2 Pentium III 1 GHz Intel CPUs
  - Supermicro 370 DLE Dual PIII-FCPGA motherboard
  - 2 256 MB 168-pin PC133 Registered ECC Micron RAM
  - 1 20 GB Maxtor ATA/66 5400 RPM HD
  - 2 40 GB Maxtor UDMA/100 7200 RPM HD
  - Asus CD-S500 50x CDROM
  - 1.4 MB floppy drive
  - ATI Expert 98 8 MB PCI video card
  - Full-tower case

Otra característica es el gabinete de los nodos. Los hay de escritorio y mini-torre. Es importante conocer esta característica para administrar adecuadamente el espacio proporcionado. Se recomienda hacer una lista de todas las caracteríticas del hardware disponible lo más detallada posible. Esta lista servirá para la optimización de los recusrsos como se verá en la sección de optimización. Además, algunos de estos parámetros se requieren en la administración del software, sobre todo en la programación paralela.

La elección del diseño, ya sea con o sin disco, tiene ventajas y desventajas con respecto a la administración en sofware y hardware. En esta última se reflejará en

la velodcidad de acceso a los nodos entre otra cosas, como se describirá en la sección de red.

Las características del hardware del cluster de la Facultad se describieron en el *capítulo 2*.

3.2. Optimización de los recursos

Una vez que se conoce el objetivo del *Cluster o Sistema Beowulf* , las características de los componentes disponibles y con base en esto, se ha hecho el diseño con su consecuente implantación, se pasa a la optimización de estos recursos.

Antes que nada, para lograr una buena administración se debe saber las ventajas y desventajas de un *Cluster de sistemas linux y un sistema SMP*.

Ventajas:

- La actual explosión de sistemas en red, significa que la mayoría del hardware para construir un cluster está siendo vendida en grandes cantidades, que da como resultado precios bajos. Los más grandes ahorros vienen del hecho de que sólo una tarjeta de video, monitor y teclado son necesarios para cada cluster en comparación con un sistema SMP (Symmetric Multi-Processers).
- Un cluster puede escalarse a sistemas muy grandes. Mientras que actualmente es díficil encontrar un sistema SMP compatible con Linux con más de cuatro procesadores.
- El hecho de reemplazar una máquina dañada en un cluster es trivial comparado a una falla parcial de un sistema SMP, que produce mucha más alta disponibilidad en las configuraciones de cluster diseñados cuidadosamente. Esto llega a ser importante no sólo para aplicaciones particulares que no pueden tolerar interrupciones de servicio significativos, sino también, para uso general

de sistemas que contienen suficientes procesadores, así que, fallas en una sóla máquina son totalmente comunes.

Desventajas:

- Con unas pocas excepciones, el hardware de la red no está diseñado para procesamiento paralelo. Típicamente la latencia es muy alta y el ancho de banda relativamente bajo comparado a un sistema SMP. Por ejemplo la latencia de un sistema SMP es generalmente no más que unos pocos microsegundos, pero es comunmente de cientos o miles de microsegundos para un cluster. El ancho de banda de comunicación de un sistema SMP es a veces más de MBytes/second; aunque el hardware de red más rápido(Gigabit Ethernet) ofrece velocidad comparable, las redes más comunmente usadas son entre 10 y 1000 veces más lentas. El desempeño(performance) del hardware de red es bastante deficiente como en una red de cluster aislado. Si la red no está aislada de otro tráfico, como a veces es el caso usando "máquinas que pasan a ser parte de la red"en lugar de un sistema diseñado como un cluster, el desempeño puede ser substancialmente mucho peor.

- Hay muy poco software soportado para tratar a un cluster como un único sistema. Por ejemplo, el comando *ps* reporta sólo los procesos que están corriendo en un sistema Linux, no todos los procesos corriendo a través de un cluster de sistema Linux.

Nota: Actulamente hay y se está desarrollado software para la administración de un sistema Beowulf.

Cabe mencionar que la diferencia entre memoria compartida y distribuida es muy importante para la optimización del sistema en hardware y software.
Este trabajo de investigación se enfoca en el modelo de memoria distribuida ya que el servidor es de un solo procesador con muchos nodos. Se describirá con más detalle en la subsección de *Procesador y memoria*. Además, existe software para servidores de más de un procesador y en Linux existe un kernel especial llamado *SMP* para trabajar con servidores de múltiples procesadores, por lo tanto, hay que tener muy

presente esta diferencia.

Para más detalles consultar el capítulo I: "Marco Teórico"

Nota: Desde la transformación de muchas compañías de supercomputadoras paralelas, el COTS(Commercial Off-The-Shelf) es comunmente discutido como un requerimiento para los sistemas computacionales paralelos. Siendo puro fanatismo. El concepto fundamental de COTS es realmente minimizar el desarrollo de tiempo y costo. La mayoría de las veces, procesamiento paralelo COTS se refiere a un cluster en el cual los nodos son PCs comunes o génericas, pero la interfaz de red y el software son en cierto modo personalizados, típicamente corriendo Linux y códigos de aplicaciones que están libremente disponibles, pero no literalmente COTS[17].

3.2.1. En el nodo sin discos

El diseño más común de un sistema Beowulf es sin disco (diskless).

Principios para configurar el hardware

Configurar un gran cluster requiere pensar que es lo que se quiere; dónde se van a poner las máquinas, no se deben poner en cualquier lugar, si se piensa utilizar sólo para pequeñas pruebas esataría bien, pero si se está planeando para desarrollar un cluster de N nodos se debe estar seguro que el ambiente debe tener las condiciones necesarias para el buen funcionamiento del cluster.

Se puede preparar uno o más racks de para alojar a las máquinas y configurar la apropiada topología de red. Se necesitará asegurar que hay suficiente infraestructura para soportar tal número de máquinas. Que el sistema de aire acondicionado soporte la carga y que en caso de una falla de poder, el UPS puede apagar correctamente todos los sistemas requeridos. Se puede querer invertir en un "KVM Switch"para facilitar el acceso a las consolas de las máquinas.

Incluso, si no se tiene el número de nodos que justifiquen estas inversiones, se debe asegurar que siempre se pueda tener acceso fácilmente a los diferentes nodos, nunca

se sabe cuándo se tiene que cambiar el ventilador o un disco duro de una máquina en problemas[16].

3.2.2. En el nodo con discos

El administrador debe asegurar el alto desempeño en el cluster, cuando se ejecute una aplicación. Una forma de asegurar tal objetivo es diseñando un *RAID* por medio de software o hardware.

RAID(Redundant Array of Inexpensive Disks: Arreglo Redundante de Discos No Expandibles) es una tecnología simple para incrementar el ancho de banda y la integridad de E/S del disco. Aunque hay muchas variantes diferentes, todas tienen dos conceptos claves en común. Primero, cada bloque de datos es alineado a través de un grupo de $n+k$ discos, de tal forma que cada drive sólo tenga que leer o escribir $1/n$ de los datos produciendo n veces el ancho de banda de un drive. Segundo, el dato redundante es escrito, así que el dato puede ser recobrado si el drive del disco falla; esto es importante porque de otra manera, si algún disco de los $n+k$ drives hubiera fallado, todo el sistema de archivos se hubiera perdido.
A parte del hardware RAID especializado soportado, Linux también soporta software RAID 0, 1, 4 y 5 a través de múltiples discos alojados por un único sistema Linux.
RAID a través de los drives del disco sobre multiples máquinas en un cluster no está soportado directamente.

Para más detalles ver:
http://www.dpt.com/uraiddoc.html.
http://linas.org/linux/raid.html.
Software RAID mini-HOWTO.
Multi-Disk System Tuning mini-HOWTO[17].

Este arreglo de discos se hace en el servidor, que es parte la optimización de sus recursos.

La gran ventaje de usar disco en los nodos es que no habrá tráfico NFS y la desventaja es una instalación y manutención muy complicada. La manutención de las configuraciones podría ser más fácil con *scripts* y utilidades complejas para actualizar todos los sistemas de archivos[26]. Sin embargo, en la actualidad hay, y se están desarrollando, herramientas más agradables para la administración de cluster de alto desempeño.

3.3. Red

Una buena administración de la red, se reflejará en el desempeño general del cluster. Como estamos trabajando en un cluster de sistemas Linux, las vías de comunicación con la memoria, el procesador y el disco duro de los nodos (en el caso standalone; máquinas que funcionan de manera independiente) y del servidor son a través de la infraestructura de la red. Por lo tanto, conocerla es fundamental. Para ello hay varios términos importantes:

Ancho de banda de la comunicación

El ancho de banda de un sistema de comunicaciones es la máxima cantidad de datos que puede ser transmitida en una unidad de tiempo, una vez que la transmisión de datos ha comenzado. El ancho de banda para transmisiones series, es a veces medido en "baudios" o "bits/second(b/s)" lo cual generalmente corresponde de 1/10 a 1/8 en muchos "bytes/second". Por ejemplo, un modem de 1200 baud trasfiere cerca de 120b/s, mientras que una conexión de red ATM a 155 Mb/s esta cerca de 130,000 veces más rápida, transfiriendo cerca de 17Mb/s.
Un ancho de banda grande permite que grandes bloques de datos puedan ser transferidos eficientemente entre los procesadores.

Latencia en la comunicación

La latencia en un sistema de comunicación es el tiempo mínimo tomado para

transferir un objeto, incluyendo algún software de envío y recepción. *La latencia es muy importante en el procesamiento paralelo porque está determina "the minimum useful grain size", el tiempo de ejecución mínimo para un segmento de código para redituar la aceleración a través de una ejecución paralela. Básicamnete, si un segmento de código se ejecuta en menos tiempo de lo que le toma transmitir su valor resultante (ejemplo:latencia), ejecutando este segmento de código serialmente sobre el procesador que necesita el valor del resultado podría ser más rápido que la ejecución paralela; la ejecución serial podría evitar la saturación de la comunicación.*

Hardware de Red

Una red de computadoras es un campo explorado, que la mayoría ya conoce. Un gran número de tecnologías y productos de red están siendo desarrollados y, la mayoría está disponible en formas que pueden ser aplicadas para hacer un cluster de procesamiento paralelo, fuera de un grupo de máquinas (por ejemplo; cada una de las PCs corriendo Linux).

Desafortunadamente, ninguna de las tecnologías de red resuelve todos los problemas de la mejor manera, de hecho, el rango de aproximación, costo y desempeño es al principio díficil de creer. Por ejemplo, usando hardware estándar disponible comercialmente, el costo por máquina anexada se clasifica de menos de $5 a más de $4,000. El ancho de banda y la latencia liberadas por cada una, también varía en más de cuatro órdenes de magnitud.

3.3.1. Algunas características para la red de un cluster

La mayoría del hardware de red esta vinculado vía un driver del kernel, soportando típicamente comunicación TCP/UDP, algunas otras redes usan más interfaces directas (ejemplo, librerías) para reducir la latencia por "bypassing" del kernel.

Años atrás, solía ser considerado perfectamente aceptable acceder a una unidad de punto flotante vía una llamada al sistema operativo, pero ahora esto es clara-

mente absurdo, se podría pensar que esto entorpecería cada comunicación entre los procesadores ejecutando un programa paralelo que requiere una llamada al sistema operativo. El problema es que estas computadoras todavía no tienen integrado estos mecanismos de comunicación, así que, acercamientos de "no-kernel" tienden a tener problemas de portabilidad. Se va a escuchar mucho sobre esto en un futuro cercano, sobre todo, en la forma de la nueva arquitectura Interfaz Virtual (Virtual Interface, VI), la cual es un método estándar para la mayoría de las operaciones de interfaz de redes para enlazar las capas comunes de llamadas al sistema operativo. El estándar VI está respaldado por Compaq, Intel y Microsoft y, seguramente tendrá un fuerte impacto en los diseños de "System Area Network (SAN)" en los próximos años.

Máximo ancho de banda

El máximo ancho de banda es un número que todo administrador de un *Cluster o Sistema Unix* debe tener cuidado, como ya se señaló anteriormente.

Mínima latencia

El valor de la mínima latencia es muy importante, incluso más que el máximo ancho de banda, en un cluster de alto desempeño. En la mayoría de los casos, la latencia de la red es sólo de un pocos microsegundos; los números más grandes reflejan capas de la interfaz de hardware y software ineficientes.

Disponible como:

Los productos que se necesitan están disponibles ampliamente con muchos proveedores, con el precio como principal factor a distinguir, productos de múltiples vendedores están disponibles por más de un competidor, pero hay diferencias significativas y problemas potenciales de interoperatividad. Las redes de un sólo proveedor se dejan a merced de éste. Los diseños de dominio público quieren decir que, incluso si no se puede encontrar a un proveedor para adquirir los productos,

también se pueden comprar partes y hacerlo uno mismo.

Interfaz puerto/bus usado:

¿Cómo se conecta (hook-up) la red? En la actualidad el desempeño más alto y más común es una tarjeta de bus PCI. También están las tarjetas con bus EISA, VESA Bus local (bus VL) e ISA. La tarjeta ISA fue la primera y, todavía es comunmente usada para tarjetas de bajo desempeño. EISA todavía se encuentra como el segundo bus en un lote de máquinas PCI, así que hay pocas de estas tarjetas. Actualmente, no se encuentran productos VL.

También existen algunas interfaces que se pueden usar sin tener que abrir la PC. Interfaces *IrDA* y *USB* están apareciendo con mucha frecuencia. El puerto parallelo estándar (The Standard Parallel Port "SPP"), usado comunmente para conetar la impresora, se ha visto bastante su uso de latencia como una extensión externa a el bus ISA; esta nueva funcionalidad es mejorada por el estándar IEEE 1284, el cual especifica mejoramientos en EPP y ECP. También, existe el antiguo, confiable y lento puerto serial RS232.

Estructura de la red

Un *bus* es un cable, conjunto de cables o fibra. Un *hub* o mejor aún un *switch*, es una pequeña caja que sabe cómo conectar diferentes alambres/fibras conectadas en éste. Hubs intercambiados permiten múltiples conexiones para estar transmitiendo activamente datos simultáneamente.

Costo por máquina conectada:

Esta característica puede verse exagerada, pero en algunos casos es importante considerarla, sobre todo cuando no se cuenta con mucho presupuesto.

Aquí un ejemplo: Suponiendo que no se toma en cuenta la conexión de la red,

el costo es de $2,000 para adquirir una PC que será utilizada en el cluster.
Sumando una *Fast Ethernet* trae un costo por nodo de aproximadamente $2,400; añadiendo una *Myrinet* en lugar de ésta trae un costo aproximadamente de $3,800. Si se tiene aproximadamente $20,000 para gastar, lo que significa que se podría tener cualquiera de estas dos opciones; 8 máquinas conetadas por una Fast Ethernet o 5 máquinas conectadas por una Myrinet. También, puede ser razonable tener múltiples redes; por ejemplo con $20,000 podrías comprar 8 máquinas conectadas por ambos, una Fast Ethernet y un TTL_PAPERS. Elegir la red o el conjunto de red adecuadamente, es lo más probable para crear un cluster que correrá aplicaciones más rápido.

Una vez que se describieron algunas características a considerar en la red, el paso siguiente será proporcionar algunas de muchas tecnologías soportadas por Linux, que es el sistema operativo empleado en esta investigación.

3.3.2. Tecnologías soportadas por Linux

ArcNet

Soporte para Linux: kernel drivers
Ancho de banda máximo: 2.5 Mb/s
Latencia mínima: 1,000 microseconds
Disponible como: multiple vendor hardware
Puerto/Bus de Interfase: ISA
Estructura de red: hub o bus no conmutado (anillo logico)
Costo por máquina conectada: $200 dlls

ARCNET es una red de área local que esta destinada principalmente para usarse en sitemas de control embebidos en tiempo real. Como Ethernet, la red está físicamente organizada en cualquiera de estas dos formas; como capas sobre uno o varios hubs, sin embargo, a diferencia de Ethernet éste usa un protocolo lógicamente basado en tokens (token-based) estructurando la red como un anillo. La

cabecera de los paquetes son pequeños (3 o 4 bytes) y el mensaje puede ser transportado tan pequeño como un solo byte de información. De este modo, ARCNET proporciona más desempeño consistente que Ethernet, con retrasos limitados, etc,. Desafortunadamente, está es más lenta que Ethernet y menos popular, haciendola más cara.
Más información en http://www.arcnet.com/

ATM

Soporte para Linux: kernel driver, librería AAL*
Ancho de banda máximo: 155 Mb/s (soon, 1,200 Mb/s)
Latencia mínima: 120 microsegundos
Disponible como: multiple vendor hardware
Puerto/Bus de interfase: PCI
Estructura de red: switched hubs
Costo por máquina conectada: $3,000 dlls

A menos que se haya estado en coma en años pasados, probablemente se ha escuchado mucho sobre cómo *ATM (Asynchronous Transfer Mode)* es el futuro.
ATM es más barata que *HiPPI* y más rápida que *Fast Ethernet* y puede ser usada sobre distancias muy grandes que las compañias telefónicas custodian.
El protocolo de red ATM también está diseñado para proveer una interfaz de software de más baja sobrecarga (lower-overhead) y para administrar más eficientemente mensajes pequeños y para comunicaciones en tiempo real (por ejemplo audio y video digital). También, es una de las redes de gran ancho de banda que actualmente soporta Linux. Las malas noticias son que ATM no es barata y, hay todavía algunos problemas de compatibilidad a través de los proveedores.
Más información:http://lrcwww.epfl.ch/linux-atm/

CAPERS

Soporte para Linux: librería AFAPI
Ancho de banda máximo: 1.2 Mb/s
Latencia mínima: 3 microsegundos
Disponible como: hardware comercial
Puerto/Bus de interfase: SPP
Estructura de red: cable entre 2 máquinas
Costo por máquina conectada: $2 dlls

CAPERS (Cable Adapter for Parallel Execution and Rapid Synchronization) es un (spin-off) del proyecto *PAPERS*, http://garage.ecn.purdue.edu/ papers/ en la Escuela de Ingeniería Eléctrica y de Cómputo de la Universidad de Pardue (Purdue University School of Electrical and Computer Engineering). En esencia, esto define un protocolo de software usando un cable ordinario "LapLink" SPP-a-SPP para implantar las librerías PAPERS en dos PCs Linux.
Con TTL_PAPERS, para mejorar el sistema de seguridad, hay un parche pequeño recomendado para el kernel, pero no requerido.
Más información: http://garage.ecn.purdue.edu/ papers/giveioperm.html.

Ethernet

Soporte para Linux: kernel drivers
Ancho de banda máximo: 10 Mb/s
Latencia mínima: 100 microsegundos
Disponible como: hardware comercial
Puerto/Bus de interfase: PCI
Estructura de red: switches, hubs o bus
Costo por máquina conectada: $100 dlls (sin hubs, $50 dlls)

Desde algunos años hasta ahora, Ethernet a 10 Mbits/s ha sido la tecnología de red estándar. Tarjetas adecuadas pueden conseguirse fácilmente y, actualmente un amplio número de PCs tienen un controlador Ethernet construido en la tarjeta

madre. Para redes de uso moderado, conexiones Ethernet pueden ser organizadas como una "multi-tap bus"sin un hub; tales configuraciones pueden servir hasta 200 máquinas con un mínimo costo, pero *no es apropiado para procesamiento paralelo*. Añadiendo un hub sin intercambio (unswitched hub) realmente no ayuda al desempeño. Sin embargo, hubs intercambiados (switched hubs) que pueden proveer un ancho de banda completo para conexiones simultáneas, el costo disminuye por puerto. Linux soporta un inmenso rango de tarjetas Ethernet, pero es importante estar consciente de las variaciones de las interfaces en hardware que pueden producir diferencias significativas en el desempeño.
Más información : http://cesdis1.gsfc.nasa.gov/linux/drivers/.

Una manera interesante de mejorar el desempeño es ofrecida por un cluster de 16 máquinas Linux, trabajo hecho en el proyecto Beowulf, http://cesdis.gsfc.nasa.go, http://cesdis1.gsfc.nasa.gov/linux/ drivers/.v/linux/ beowulf/beowulf.html en NASA CESDIS. Aquí, Donald Becker, quien es el autor de muchos controladores de tarjetas Ethernet, ha desarrollado soporte para *carga compartida (load sharing)* a través de múltiples redes Ethernet que se siguen (shadow) cada una de la otra (por ejemplo; compartir las mismas direcciones de red). Esta carga compartida está incluida en la distribución estándar de Linux y, es transparente por debajo del nivel de operación del socket, porque el costo del hub es importante, teniendo cada una de la máquinas conectadas a dos o más hubs hubless o hub unswitched, las redes Ethernet pueden ser una manera de costo efectivo (cost-effective) para mejorar el desempeño. De hecho, en situaciones donde una máquina es la causante del tráfico en la red, la carga compartida usando redes "shadow" trabajan mucho mejor que usando una sola red con hub switched.

Ethernet (Fast Ethernet)

Soporte para Linux: kernel drivers
Ancho de banda máximo: 100 Mb/s
Latencia mínima: 80 microsegundos

Disponible como: hardware comercial
Puerto/Bus de interfase: PCI
Estructura de red: switched or unswitched hubs
Costo por máquina conectada: $400 dlls

A pesar de que realmente hay muy pocas diferencias tecnológicas llamadas por sí mismas *Fast Ethernet*, este término, la mayoría de las veces se refiere a un hub basado en Ethernet a 100 Mbits/s, que es, en cierto modo compatible con dispositivos y cables más antiguos de 10 Mbits/s 10 BaseT. Como se puede esperar, cualquier cosa llamada Ethernet es generalmente (priced for a volume market) pagada en el mercado y, estas interfaces son generalmente una pequeña fracción del precio de las tarjetas ATM de 155 Mbits/s. La idea es que usando un grupo de máquinas dividiendo el ancho de banda de un solo bus a 100 Mbits/s (usando un hub unswitched) reditua desempeño que aún puede no ser tan bueno en promedio como usando una Ethernet de 10 Mbits/s con hub switched que puede dar a cada una de las conexiones de las máquinas 10 Mbits/s completos.

Los Hubs Switched que pueden proveer 100 Mbits/s para cada una de las máquinas simultáneamente son caros, pero los precios están disminuyendo cada día y, estos switches hacen rendir mucho más alto el ancho de banda total de la red que los hubs unswitched.
El factor que hace a los switches ATM tan caros es que ellos deben intercambiarse para cada uno (relativamente pequeños) de los elementos ATM, algunos switch Fast Ethernet toman ventaja de la esperada frecuencia de intercambio más baja, usando técnicas que pueden bajar la latencia a través del switch, pero, toman múltiples milisegundos para cambiar la ruta del switch. *Si se tiene cambios de patrón de ruteo frecuentemente, evitar estos switches.*
Más información: http://cesdis1.gsfc.nasa.gov/linux/drivers/.

También, como se describió para Ethernet, el proyecto Beowulf, http:// cesdis.gsfc.nasa.gov/linux/beowulf/beowulf.html de la NASA ha estado desarrollando soporte que ofrece mejorar el desempeño por carga compartida a través de múltiples

Fast Ethernets.

Ethernet (Gigabit Ethernet)

Soporte para Linux: kernel drivers
Ancho de banda máximo: 1,000 Mb/s
latencia mínima: 300 microsegundos
Disponible como: multiple vendor hardware
Puerto/Bus de interfase: PCI
Estrucutra de red: switched hubs o FDRs
Costo por máquina conectada: $2,500 dlls

Se podría pensar que Gigabit Ethernet, http://www.gigabit-ethernet.org/, tiene una buena tecnología, razón para ser llamada Ethernet, pero el nombre hace reflejar precisamente el hecho de que esto intenta ser una tecnología de red de computadoras barata, que se puede encontrar en cualquier parte con soporte nativo para IP. Sin embargo, actualmente los precios reflejan el hecho que el hardware para Gb/s es todavía una cosa díficil de contruir.

A diferencia de otras tecnologíoas Ethernet, Gigabit Ethernet provee un nivel de control de flujo que debería hacer más confiable la red. FDRs o repetidores Full-Duplex, simplifican líneas multiplexadas (multiplex lines) usando buffer y localizando el flujo de control para mejorar el desempeño. La mayoría de hubs intercambiados (switched hubs) están siendo contruidos como módulos con una nueva interfaz, para existir con los switch con capacidad de gigabit.

Más información :
http://www.acacianet.com/
http://www.baynetworks.com/,
http://www.cabletron.com/,
http://www.networks.digital.com/,

http://www.extremenetworks.com/,
http://www.foundrynet.com/,
http://www.gigalabs.com/,
http://www.packetengines.com/
http://www.plaintree.com/,
http://www.prominet.com/,
http://www.sun.com/, and http://www.xlnt.com/.

Hay un drive para Linux; http://cesdis.gsfc.nasa.gov/linux/drivers/yellowfin.html, para el Packet Engines "Yellowfin" G-NIC; http://www.packetengines.com/. Pruebas anteriores bajo Linux lograron cerca de 2.5x ancho de banda más alto de lo podría ser logrado con la mejor Fast Ethernet a 100 Mb/s; con redes gigabit, la sintonía (tuning) adecuada del uso del bus PCI es un factor crítico. Hay pequeñas dudas sobre el mejoramiento de los drives y los drives de Linux para otras NICs.

Myrinet

Soporte para Linux: library
Ancho de banda máximo: 1,280 Mb/s
Latencia mínima: 9 microsegundos
Disponible como: single-vendor hardware
Puerto/Bus de interfase: PCI
Estructura de red: switched hubs
Costo por máquina conectada: $1,800 dlls

Myrinet, http://www.myri.com/, es una red de área local (LAN) diseñada también para servir como un SAN (System Area Network), por ejemplo; una red dentro de un rack lleno de máquinas conetadas como un sistema paralelo. Las versiones de LAN y SAN usan diferente medio físico y tienen algunas características diferentes; *generalmente, la versión de SAN podría ser usada dentro un cluster.* Myrinet es completamente convencional en estructura, pero tiene una reputación

para ser bien implementada. Se dice que los drivers para Linux se desempeñan muy bien, aunque grandes anomalías en las variaciones del desempeño han sido reportadas con implementaciones de diferentes bus PCI para los nodos de las computadoras.

Actualmente, Myrinet es claramente la red favorita para grupos de cluster que no son severamente un reto presupuestal o financiero (budgetarily challenged). Si la idea de una PC Linux es un Pentium Pro o un Pentium II con al menos 256 MB RAM y un RAID SCSI, el costo de una Myrinet es bastante razonable. Sin embargo, usando más configuraciones de PC ordinarias, se puede encontrar que la elección está entre N máquinas unidas por Myrinet o 2N máquinas unidas por múltiples Fast Ethernet y TTL_PAPERS. Realmente depende de cuál es el presupuesto y de qué tipo de cómputo se desea sobre la mayoría.

Parastation

Soporte para Linux: HAL or socket library
Ancho de banda máximo: 125 Mb/s
Latencia mínima: 2 microsegundos
Disponible como: single vendor hardware
Puerto/Bus de interfase: PCI
Estrucutura de red: hubless mesh
Costo por máquina conectada: $1,000 dlls

El proyecto ParaStation http://wwwipd.ira.uka.de/parastation en el Departamento de Informática de la Universidad de Karlsruhe (University of Karlsruhe Department of Informatics) está construyendo una red personalizada de baja latencia compatible con PVM. Primero se contruyó un prototipo para PC de dos procesadores usando una interfaz de tarjeta EISA y PCs corriendo Unix BSD y, tiempo depués se construyó un cluster más grande usando Alphas DELL. Desde enero de 1997, Parastation ha estado disponible para Linux. Las tarjetas PCI están siendo fabricadas en coperación con una compañía llamada Hitex (ver: http://www.hitex.com/parastation/).

El hardware de Parastation implanta ambos; mensajes de transmisión rápidos y confiables y una sincronización simple de barrido (simple barrier synchronization).

PLIP

Soporte para Linux: kernel driver
Ancho de banda máximo: 1.2 Mb/s
Latencia mínima: 1,000 microsegundos
Disponible como: hardware comercial
Puerto/Bus de interfase: SPP
Estructura de red: cable entre 2 maquinas
Costo por máquina conectada: $2 dlls

Sólo por el costo de un cable *LapLink*, PLIP (Parallel Line Interface Protocol) permite a dos máquinas Linux comunicarse a través de puertos paralelos usando sockets basados en software. *En términos de ancho de banda, latencia y escalabilidad, está no es una tecnología de red muy seria, sin embargo, el costo muy cercano al cero y la compatibilidad del software son útiles. Los drivers son parte de la distribución del kernel Linux estándar.*

SCSI

Soporte para Linux: kernel drivers
Ancho de banda máximo: 5 Mb/s sobre 20 Mb/s
Disponible como: multiple vendor hardware
Puerto/Bus de interfase: PCI, EISA, ISA
Estructura de red: dispositivo de bus SCSI interno compartido

El SCSI (Small Computer Systems Interconnect) es esencialmente un bus de E/S que es usado para drivers de disco, CD ROMS, scanner, etc. Hay tres estándar separados SCSI-1, SCSI-2 y SCSI-3; velocidades Fast y Ultra; y anchuras en la ruta

de datos de 8, 16, o 32 bits (con compatibilidad de FireWire también mencionado en SCSI-3).
Todo esto es algo confuso, pero, conocemos que un buen SCSI es algo más rápido que un EIDE y puede tomar más dispositivos, más eficientemente. Si mucha gente no lo utiliza, es porque es bastante simple para dos computadoras compartir un solo bus SCSI. Este tipo de configuración es muy útil para compartir drives del disco entre máquinas e implantar un *fail-over* (Teniendo una máquina que toma la peticiones de una base de datos, cuando la otra máquina falla).

Actualmente, éste es el único mecanismo soportado por productos de cluster de PCs de Microsoft, WolfPack. sin embargo, la imposibilidad de escalar a sistemas más grandes hace que el SCSI compartido no sea de interés para el procesamiento paralelo en general.

SHRIMP

Soporte para Linux: Interfase de memoria mapeada a nivel de usuario
Ancho de banda máximo: 180 Mb/s
Latencia mínima: 5 microsegundos
Disponible como: prototipo de investigación
Puerto/Bus de interfase: EISA
Estructura de red: mesh backplane

El proyecto SHRIMP, http://www.CS.Princeton.EDU/shrimp/, en el Departamento de Ciencias de la Computación de la universidad de Princeton (Princeton University Computer Science Department) está contruyendo una computadora paralela usando PC corriendo Linux como elementos de procesamiento. El primer SHRIMP (Scalable, High-Performance, Really Inexpensive Multi-Processor) fue un prototipo simple de dos procesadores usando un "dual-ported RAM"sobre una tarjeta de interfaz habitual EISA. Actualmente, hay un prototipo que se escalará a configuraciones más grandes usando una tarjeta de interfaz común para conectar a un hub

que es esencialmente la misma trama de enrutamiento de red usada en el modelo de Intel (ver:http://www.ssd.intel.com /paragon.html). Un esfuerzo considerable ha entrado en desarrollo de baja sobrecarga (low-overhead) en hardware de comunicación mapeada de memoria virtual (virtual memory mapped communication) y software de ayuda.

SLIP

Soporte para Linux: kernel drivers
Ancho de banda máximo: 0.1 Mb/s
Latencia mínima: 1,000 microsegundas
Disponible como: hardware comercial
Pueto/Bus de interfase: RS232C
Estructura de red: cable entre 2 maquinas
Costo por máquina conectado: $2 dlls

A pesar de que SLIP (Serial Line Interface Protocol) está firmemente plantado en el extremo inferior del espectro de desempeño, SLIP (o CSLIP or PPP) permite a dos máquinas ejecutar comunicación con socket vía puertos seriales ordinarios RS232. Los puertos seriales RS232 pueden ser conectados usando un cable serial de un modem nulo RS232 o, éstos incluso pueden ser conectados vía dial-up a través de un modem. *En cualquier caso, la latencia es alta y el ancho de banda es bajo, así que, SLIP debería ser usado sólo cuando ninguna otra alternativa esté disponible. Vale notar, sin embargo, que la mayoría de las PCs tiene dos puertos RS232, así que sería posible conetar un grupo de máquinas simplemente conetándolas como conjunto lineal o como un anillo. Incluso, hay software de carga compartida llamado EQL.*

TTL_PAPERS

Soporte para Linux: Librería AFAPI
Ancho de banda máximo: 1.6 Mb/s

Latencia miníma: 3 microsegundos
Disponible como: diseño public-domain, single-vendor hardware
Puerto/Bus de interfase: SPP
Estructura de red: árbol de hubs
Costo por máquina conectada: $100 dlls

El proyecto PAPERS (Purdue's Adapter for Parallel Execution and Rapid Synchronization), http://garage.ecn.purdue.edu/ papers/, en la Escuela de Ingeniería Eléctrica y de Cómputo de la Universidad de Purdue (University School of Electrical and Computer Engineering) está construyendo hardware y software escalable, de baja latencia y con funciones de comunicación agregadas, que permitan a una supercomputadora paralela ser construida usando PCs/estaciones de trabajo sin modificarlos, como nodos de ésta.

Ha habido más de una docena de tipos diferentes de hardware PAPERS construidos que conetan PCs/estaciones de trabajo vía el SPP (Standard Parallel Port), aproximadamente siguiendo dos líneas de desarrollo. Las versiones llamadas *PAPERS* apuntan a un desempeño más alto, usar cualquiera de las tecnologías es apropiado; el trabajo actual utiliza FPGAs y diseños de interfaz de bus PCI de gran ancho de banda que también están bajo desarrollo. En contraste, las versiones llamadas *TTL_PAPERS* están diseñadas para ser fácilmente reproducidas fuera de la universidad de Purdue, y son diseños notablemente simples de dominio público que pueden ser construidos usando lógica ordinaria TTL. Algunos de tales diseños están construidos comercialmente, http://chelsea.ios.com/ hgdietz/ sbm4.html.

A diferencia de los diseños de hardware personalizado de otras universidades, los cluster TTL_PAPERS han sido ensamblados en muchas universidades de USA a korea de Sur. El ancho de banda está severamente limitado por las conexiones SPP, pero los PAPERS implementan comunicaciones de funciones agregadas con muy baja latencia, aún el sistema más rápido de mensaje orientado (message-oriented) no puede proveer un desempeño comparable sobre estas funciones agregadas. De este modo, el PAPERS es particularmente bueno para sincronizar el despliegue de una pared de video, el programa tiene acceso a una red de un gran ancho de ban-

da, evaluando propiedades globales en investigaciones géneteicas, etc. Aunque, los cluster PAPERS han sido construidos usando máquinas IBM PowerPC AIX, DEC Alpha OSF/1 y HP PA-RISC HP-UX, máquinas basadas en Linux son las mejores plataformas soportadas.

Usando programas de usuario TTL_PAPERS AFAPI, directamente accede los registros del puerto del hardware SPP bajo Linux, sin una llamada al sistema operativo para cada acceso. Para hacer esto, AFAPI primero toma permiso del puerto usando cualquiera de las dos opciones; iopl() o ioperm(). El problema con estas llamadas es que ambas requieren el programa del usuario para ser privilegiado, produciendo un agujero de seguridad potencial. La solución es un parche opcional del kernel, http://garage.ecn.purdue.edu/ papers/giveioperm.html, que permite un proceso privilegiado para controlar el permiso del puerto para algún proceso.

USB (Universal Serial Bus)

Soporte para Linux: kernel driver
Ancho de banda máximo: 12 Mb/s
Disponible como: Hardware comercial
Puerto/Bus de interfase: USB
Estructura de red: bus
Costo por máquina conectada: $5 dlls

El USB (Universal Serial Bus, http://www.usb.org/) es un "hot-pluggable conventional-Ethernet-speed", bus para más de 127 periféricos que van desde teclados hasta cámaras de videoconferencias. Realmente no está claro cómo múltiples computadoras pueden estar conectadas cada una a la otra usando USB. En cualquier caso, los puertos USB están llegando a ser rápidamente tan estándar en tarjetas madres de PC como el RS232 y el SPP. El desarrollo de un driver para Linux está discrito en http://peloncho.fis.ucm.es/ inaky/USB.html.
En algunos casos, USB es casi la versión de FireWire de bajo desempeño y costo cero que se puede comprar en estos días.

WAPERS

Soporte para Linux: librería AFAPI
Ancho de banda máximo: 0.4 Mb/s
Latencia mínima: 3 microsegundos
Disponible como: diseño public-domain
Puerto/Bus de interfase: SPP
Estructura de red : patron de cableado entre 2-64 máquinas
Costo por máquina: $5 dlls

El WAPERS (Wired-AND Adapter for Parallel Execution and Rapid Synchronization) es un "spin-off" del proyecto PAPERS, http://garage.ecn.purdue. edu/ papers/, en la Escuela de Ingeniería Eléctrica y de Cómputo de la Universidad de Purdue (Purdue University School of Electrical and Computer Engineering). Si se implanta apropiadamente, el SPP tiene cuatro bits de salida colector-abierto que pueden ser conectados juntos a través de las máquinas para implantar una AND alambrada grande de 4 bits. Esta AND conectada es eléctricamente delicada y, el máximo número de máquinas que pueden ser conectadas de esta manera crítica, depende de las propiedades analógicas de los puertos; típicamente más de 7 u 8 máquinas pueden ser conectadas por WAPERS. *Aunque el costo y la latencia son muy bajos el ancho de banda es un problema; el WAPERS es mucho mejor como una segunda red para agregar operaciones que como la única red en un cluster. Al igual que los TTL_PAPERS, para mejorar el sistema de seguridad, hay un parche para el kernel recomendado, pero no requerido: http://garage.ecn.purdue.edu/ papers/giveioperm.html* [16]

Hay un acuerdo casi universal de que el cuello de botella (bottleneck) del cómputo de los cluster Beowulf es la red por la cual se comunican las máquinas. Ethernet a 10Mbps es generalmente demasiada lenta para todo, pero es la forma más ordinaria de paralelismo y, Ethernet a 1000Mbps y ATM son vistas demasiadas

caras, así que, fast Ethernet (100Mbps) ha sido la mejor opción (al menos hasta que las tecnologías más rápidas bajen de precio). *Por simplicidad y bajo costo, la gran mayoría de de clusters Beowulf usan una topología estrella con toda la comunicación direccionada (routed) a través de un switch central.*

La topología estrella tiene un pequeño problema. Si toda la comunicación es direccionada a través del hub o switch central, entonces este switch puede llegar a ser el cuello de botella computacional. El problema es que mientras la comunicación paralela incrementa, el switch puede llegar a saturarse, causando retrasos en la transmisión de los mensajes. El resultado es un retraso en la latencia de la comunicación promedio (el tiempo para transmitir un bit de una máquina a otra) para el sistema completo.

Reducir la latencia en la comunicación ha conducido a una variedad interesante de topologías de red, incluyendo ring, árboles y muchas estructuras híbridas. Por ejemplo el cluster *LoBos*[22] en el Instituto Nacional de salud y el cluster *Wulfpack*[23] en la Escuela Medica Johns Hopkins, cada uno usa una topología *anillo-estrella híbrida*.
Esta topología provee ligas directas entre cada uno de los nodos y a sus dos más cercanos vecinos e intercambia (1-hop) ligas entre cada uno de los nodos y sus nodos no-vecinos. Permitiendo a los nodos vecinos comunicarse directamente, esta topología reduce el tráfico de los mensajes a través del switch y, por lo tanto reduce la latencia de comunicación promedio del sistema completo[19].

En los párrafos anteriores se han descrito las ventajas y desventas de algunas tecnologías de red, que es una parte muy impotante en el alto desempeño de un cluster. El administrador del Sistema Beowulf debe conocerlas para tener una visión más amplia y, así poder tomar decisiones adecuadas en el diseño de la infraestructura de la Red.

El Sistema Beowulf de la Facultad de Ingeniería, usa una infraestructura de red tipo Ethernet a 10 Mb/s y topología estrella. Como se describió, Ethernet a

10 Mb/s no es la opción más recomendable para procesamiento paralelo, aunque hay algunas variantes que pueden mejorarla, pero fue la opción más ajustada a los recursos disponibles en la Facultad.

3.4. Procesador y memoria

El procesador y la memoria son la parte fundamental del cómputo en un Cluster o Sistema Beowulf, así como la red en la comunicación del Cluster.

3.4.1. Memoria

Comenzaremos con unas definiciones:

SPMD (Single Program, Multiple Data)

Es una versión restringida de MIMD en el cual todos los precesadores están corriendo el mismo programa. A diferencia de SIMD, cada uno de los procesadores ejecuta código SPDM que puede tomar una ruta de control de flujo diferente a través del programa.

Memoria compartida

La memoria compartida es un modelo para interactuar entre procesadores dentro de un sistema paralelo. Sistemas como máquinas Pentium con multi-procesador corriendo Linux, comparten una sóla memoria entre sus procesadores, así que un valor escrito en la memoria compartida para un procesador puede ser directamente accedido por algun otro procesador.
Alternativamente, de forma lógica, la memoria compartida puede ser implantada para sistemas en los cuales cada uno de los procesadores tiene su propia memoria, convirtiendo cada referencia de memoria non-local en una apropiada comunicación inter-procesador. *Cualquiera de las dos; la implantación de memoria compartida es generalmente considerada más fácil de usar que la de paso de mensajes (message*

passing).
Físicamente la memoria compartida puede tener ambas; gran ancho de banda y baja latencia, pero sólo cuando múltiples procesadores no tratan de acceder el bus simultáneamente; en estos términos, el arreglo de los datos tadavía puede afectar seriamente el desempeño, y efectos del cache pueden hacerlo díficil para determinar cuál es el mejor esquema [17].

Cantidad

Elegir la correcta cantidad de memoria es una de las tareas más importantes durante el diseño de un sistema Beowulf. Si no se tiene la suficiente memoria para guardar los trabajos que se van a correr, se perderá desempeño del sistema debido al intenso *swapping*. Y es precisamente lo que no se quiere. Cada página la cual tiene que ser leída del disco duro, costará tiempo de ejecución valioso. Leer del disco duro es mucho más lento que leer de la memoria RAM. Idealmente no se quiere intercambiar (to swap) todo, pero se debería reservar algo de espacio *swap* sólo en el caso de que se necesite correr algo más grande de lo planeado[26].

Velocidad

La velocidad de memoria también es muy importante. Si se elige un procesador rápido corriendo sobre un bus rápido, será más que probable que la memoria será el cuello de botella dentro de los nodos. Se recomienda usar algo como SDRAM a 16ns[26].

Cache L2 en los nodos de cómputo

Las cualidades a considerar de un nodo de cómputo incluyen tipo de procesador, tamaño y velocidad del cache nivel 2 *(L2)*, número de procesadores por nodo, velocidad del bus frontal (front-side bus) (FSB), escalabilidad del subsistema de memoria y, velocidad del bus PCI (Peripheral Component Interconnect).

Algunas aplicaciones paralelas son amigables con el cache (cache-friendly), esto es; el problema puede ser fácilmete acomodado por la memoria cache L2. Nodos de cómputo con gran velocidad en la memoria cache L2 pueden incrementar el desempeño de estas aplicaciones. Por otro lado, aplicaciones con patrones de acceso de memoria aleatoria o rango amplio (wide-range), serán más probables que se beneficien con la velocidad del procesador, sistema de bus y subsistema de memoria más rápidos que una memoria cache L2 más grande[25].

Impacto de la memoria cache L2 y el subsistema de memoria

A continucación un ejemplo del desempeño de la memoria cache L2 y el subsistema de memoria, con servidores *DELL*[26]. Este ejemplo servirá para comparar, si es que se puede, con el cluster de la Facultad.

El impacto de la cache L2 y el subsistema de memoria en el desempeño y escalabilidad de un *cluster Beowulf*. En la figura 3.1, se usa un programa llamado *HINT* [27] para comprender las características de desempeño de dos tipos de nodos de cómputo. También, se aplicaron programas *NPB*[28] para estudiar más a detalle el impacto de la memoria cache L2 y el subsistema de memoria (memory subsystem).

Los ocho programas benchmark NPB tienen diferetes grados de relación amigable con la memoria cache (cache-friendliness). La figura 3.3 muestra el desempeño de IS y LU en dos Cluster Beowulf Dell. Un cluster consiste de 8 nodos PowerEdge 2400 con 16 procesadores Pentium III a 600MHz con 256 KB de cache; el otro cluster

[26]Cluster Beowulf DELL

[27]INTegración Jerárquica (Hierarchical INTegration, HINT). Un programa benchmark, desarrollado por el Dr.John Gustafson y otros investigadores en el Laboratorio Ames del Departamento de Energía de los Estados Unidos (U.S. Department of Energy's Ames Laboratory).

[28]Una Suite Benchmark Paralela de Simulación Numerica Aéreo espacial (Numerical Aerospace Simulation (NAS) Parallel Benchmark (NPB) suite) desarrollada por el Centro de Investigación Ames de la NASA. La suite NPB consiste de ocho programas derivados de la dinámica de fluídos computacional (computational fluid dynamics, CFD). Cada uno de los ocho programas, cinco núcleos (kernels) y tres aplicaciones simulando CFD, representan algunos aspectos particulares de la gran computación paralela de aplicaciones aéreo físicas. Los cinco núcleos, EP, FT, MG, CG e IS, simulan la esencia computacional de diferentes métodos númericos usados por las aplicaciones CFD. Las aplicaciones simuladas CFD, LU, SP y BT, reproducen mucho del movimiento de los datos y los cálculos encontrados en los códigos completos de CFD.

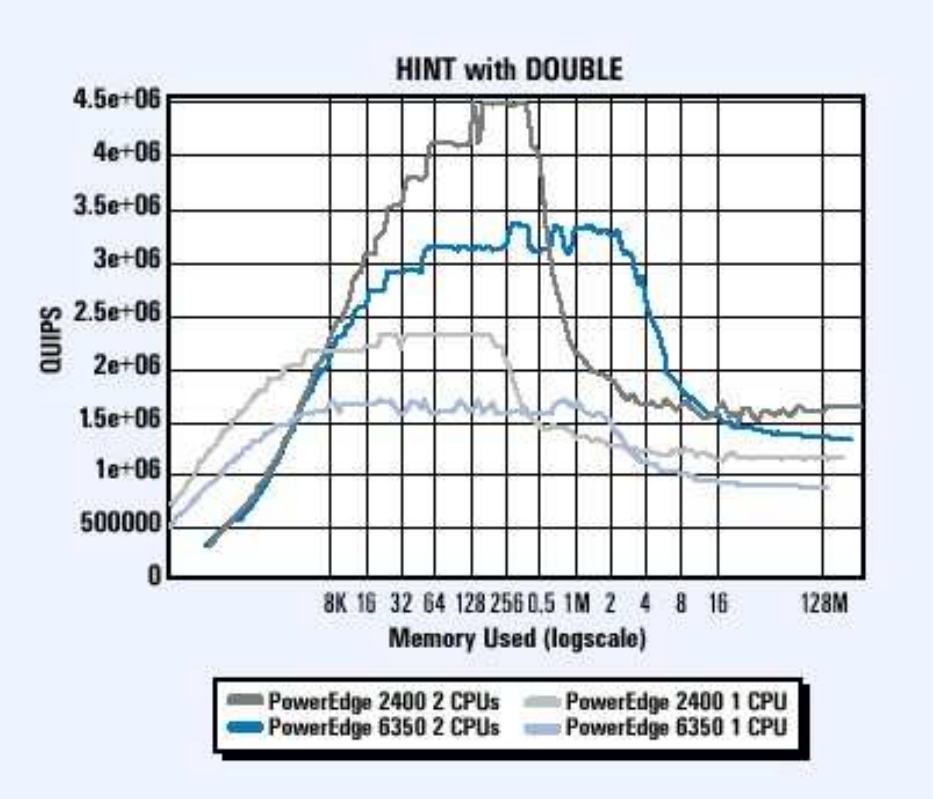

Figura 3.1: Curvas HINT de PowerEdge 2400 Dell y servidores 6350.

tiene 8 nodos PowerEdge 6350 con 16 procesadores Pentium II Xeon de 450 MHz con 2MB de L2 de cache. Se seleccionó IS y LU porque ellos son dos excelentes ejemplos de cache amigable (cache-friendliness); el programa LU es el más accesible con la cache de los programas NPB y, el programa IS es el menos accesible con la cache.

La figura 3.3 usa la misma representación de aproximación como la figura 3.2, comparando el desempeño total y el desempeño por procesador entre los dos *cluster Beowulf Dell* de dos generaciones de servidores SMP basados en arquitectura Intel. Para IS, el cluster de nodos PowerEdge 6350 se desempeñó mucho mejor que el cluster de nodos PowerEdge 6350. Sin embargo, para aplicaciones de cache accesible (cache-friendly) tales como LU, el cluster de PowerEdge6350s se desempeñó mucho mejor que su competidor. Éste también, muestra mucha mejor escalabilidad de usar una configuración 8 por 1 (8-by-1) a una de 8 por 2 (8-by-2). Esto puede ser visto en las curvas de desempeño por procesador en la figura 3.3[25].

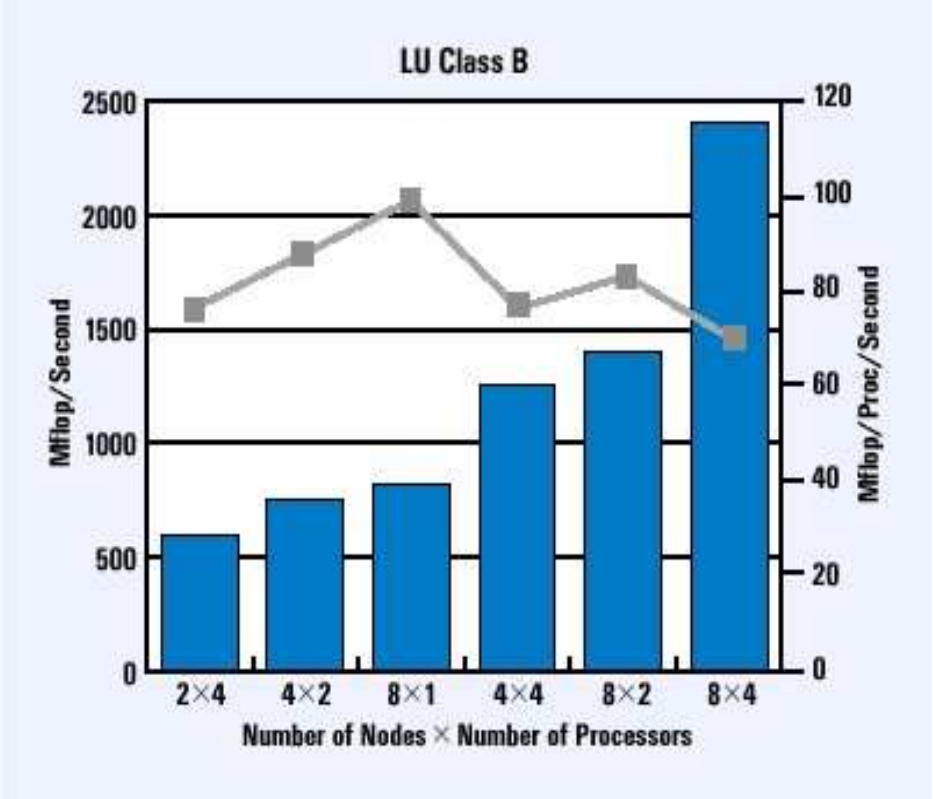

Figura 3.2: Desempeño de un programa LU.

3.4.2. Procesador

SIMD dentro de un registro (ejemplo usando MMX)

Otra forma de diseñar un máquina paralela es con SIMD (Single Instruction stream, Multiple Data stream) dentro de un registro (SWAR). Esto no es una idea nueva. Dada una máquina con k-bits de registros, rutas de datos y unidades de funciones, se ha sabido que operaciones de registros ordinarios pueden funcionar como funciones paralelas SIMD sobre n, k/n-bit valores de campos enteros. Sin embargo, esto es sólo con el reciente "push" de multimedia que trabaja de 2x a 8x más rápido, ofrecida por las técnicas de SWAR que han llegado a ser de interés para la tendecia de la computación.

Las versiones de 1997 de la mayoría de los procesadores incorporan soporte de hardware por SWAR.

Aquí algunos de ellos:

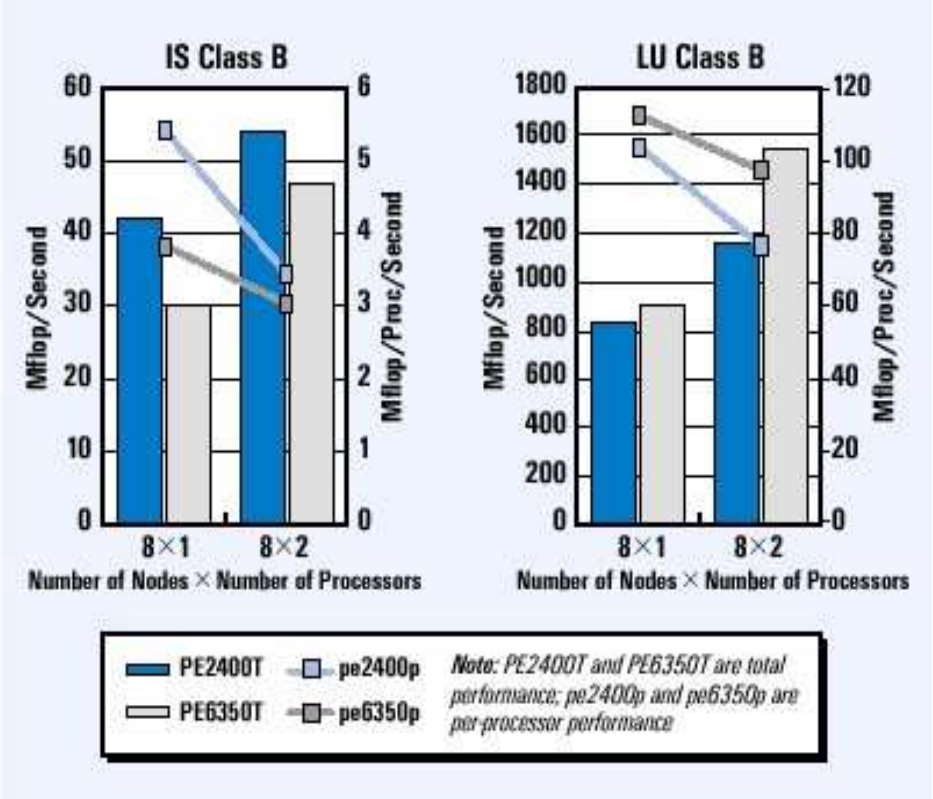

Figura 3.3: Desempeño de programas LU e IS de NPB sobre dos Cluster Beowulf Dell

AMD K6 MMX (MultiMedia eXtensions)
Cyrix M2 MMX (MultiMedia eXtensions)
Digital Alpha MAX (MultimediA eXtensions)
Hewlett-Packard PA-RISC MAX (Multimedia Acceleration eXtensions)
Intel Pentium II & Pentium with MMX (MultiMedia eXtensions)
Microunity Mediaprocessor SIGD (Single Instruction on Groups of Data)
MIPS Digital Media eXtension (MDMX, pronounced Mad Max)
Sun SPARC V9 VIS (Visual Instruction Set)

Hay pocos agujeros en el soporte de hardware proveído por los nuevos procesadores, peculiaridades como sólo soportando algunas operaciones para algunos tamaños de campo. Es importante recordar, sin embargo, que se necesita algún

soporte de hardware para muchas operaciones SWAR para ser eficiente. Por ejemplo; las operaciones "bitwise" no son afectadas por el particionamiento lógico de un registro [17].

SWAR: ¿Para qué es bueno?

A pesar de que cada procesador moderno es capaz de ejecutar con al menos algún paralelismo SWAR, el hecho triste es que incluso el mejor conjunto de instrucciones SWAR-mejorado no soporta paralelismo de propósito general. De hecho, mucha gente ha notado que el desempeño difiere entre Pentium y "Pentium con tecnología MMX" es en ocasiones debido a cosas como el cache L1 más grande que coincide con la aparición del MMX, así que, realmente, ¿para qué es bueno el SWAR (o MMX)?

- Sólo en los enteros, el más pequeño es el mejor. Dos valores de 32 bits caben en un registro MMX de 64 bits, pero hacen ocho caracteres de un byte o incluso un tarjeta de ajedrez completa es digna de valores de un bit. *Nota: habrá versiones de punto flotante de MMX, aunque se ha dicho muy poco sobre esto en esta sección de la investigación. Cyrix ha posteado un conjunto de "slides", ftp://ftp.cyrix.com/developr/mpf97rm.pdf, que incluyen unos pocos comentarios sobre MMFP. Aparentemente, MMFP soportará dos números de punto flotante de 32 bits para ser empacados dentro de un registro MMX de 64 bits; combinando esto con dos líneas de entubamiento MMFP (pipelines MMFP) producirán cuatro FLOPs de una precisión única por reloj o ciclo.*

- SIMD o paralelismo estilo vector. La misma operación es aplicada a todos los campos simultáneamente (por ejemplo; equivalente a un "SIMD enable masking"), pero ellos complican la compilación y afectan el desempeño.

- Localización de patrones de referencia a memoria preferentemente empacados. SWAR en general, y MMX en particular son terribles en accesos ordenados aleatoriamente (randomly-ordered accesses); recolectar un vector x[y] (donde y es un índice ordenado) es prohibitivamente caro.

Hay serias restricciones, pero este tipo de paralelismo ocurre en muchos algoritmos paralelos (no sólo en aplicaciones multimedia). Para el tipo correcto de algoritmo, SWAR es más efectivo que un SMP o cluster paralelo y no cuesta nada usarlo[17].

MMX SWAR bajo Linux

Para Linux, los procesadores IA32 son el principal interés. Las buenas noticias es que AMD, Cyrix e Intel, todos implementan las mismas instrucciones MMX. Sin embargo, el desempeño varía en MMX; por ejemplo, el K6 sólo tiene una línea de entubamiento MMX (MMX pipeline) y el Pentium con MMX tiene dos. Realmente, sólo las malas noticias es que Intel todavía esta corriendo estos inapropiados MMX comercialmente.

Hay realmente tres aproximaciones para usar MMX para SWAR

Usar rutinas de una librería de MMX. En pariticular, intel ha desarrollado muchas "librerías de desempeño", http://developer.intel.com/drg/tools/ad.htm, que ofrecen una variedad de rutinas "hand-optimized"para tareas de multimedia comunes. Con un pequeño esfuerzo, muchos algoritmos de non-multimedia pueden ser rediseñados para habilitar la mayoría de las porciones que requieran un intenso cálculo, para ser implementadas usando uno o más de estas rutinas de librería. Estas librerías actualmente no están disponibles para Linux, pero podrían ser desarrolladas.

Usar instrucciones MMX directamente. Esto es algo complicado por dos hechos. El primer problema es que MMX puede no estar disponible en el procesador, así que una implantación alternativa también debe ser proveída. El segundo problema es que el ensamblador IA32 generalmente usado bajo Linux, actualmente no reconoce las instrucciones MMX.

Usar un lenguaje de alto nivel o un compilador de módulos que pueda generar directamente apropiadas instrucciones MMX. Tales herramientas están actualmente bajo desarrollo, pero ninguna es todavía funcional bajo Linux.
Por ejemplo, en la Universidad de Purdue (http://dynamo.ecn.purdue.edu/ hankd/SWAR/) se está actualemte desarrollando un compilador que tomará funciones escritas en un lenguaje 'C' explícitamente paralelo y generará módulos SWAR que son accedidos como funciones de 'C', hacer uso de cualquier cosa que soporte SWAR está disponible, incluyendo MMX. El primer prototipo de compilador de módulos fue construido en otoño de 1996, sin embargo, traer esta tecnología a un estado usable está tomando mucho más tiempo del que se esperaba originalmente[17].

3.5. Escalabilidad

Una vez que está funcionando el cluster apropiadamente, existe otra tarea del adminisistrador igual y, en ocasiones más importante que las ya mencionadas; *la escalabilidad*. A continiación dos de muchas acepciones:

Escalabilidad. La habilidad para crecer en la capacidad total y encontrar el uso más eficiente demandado como sea necesario. Cuando una aplicación o departamento requiera anexar recursos computacionales, servidores adicionales pueden ser fácilmente agregados al cluster. Muchos cluster continuan creciendo y ahora están comprendidos por cientos de servidores [18].

Escalabilidad. Cuando la carga total excede las capacidades del sistema en el cluster, sistemas adicionales pueden ser sumados al cluster. En la actualidad, clientes quienes planean expandir la capacidad de su sistema, deben hacer un fuerte compromiso con servidores caros y "high-end" que proveen espacio para adicionar CPUs, drives y memoria. Usando la tecnología de cluster, los clientes podrán sumar incrementalmente sistemas estándar más pequeños, como se necesite para encontrar los requerimientos necesarios de un procesamiento completo[21].

Mientras la Computación necesite un continuo crecimiento en complejidad y

se requiera niveles más altos de capacidad de cómputo a precios más bajos, el diseño de un cluster de alto desempeño ofrece una solución que no sólo provee una poderosa implantación de cómputo, sino que también a un costo razonable, cuando se compara con el método tradicional de usar computadoras especialmente diseñadas para cómputo pararalelo.

Ha habido un incremento tremendo en el desempeño de las computadoras normales (commodity) y en el hardware de red en los últimos años. Y contrariamente, la industria ha visto un decremento gradual en los precios de los componentes.

Todo lo mencionado sirve para la escalabilidad de un cluster, cualquiera que sea su tipo, que en este trabajo de investigación es un *cluster tipo Beowulf*. La escalabilidad es de acuerdo a las necesidades de los usuarios, es decir, se podría querer ejecutar una aplicación que necesite sólo incrementar la memoria o la velocidad del procesador. Sin embargo, la escalabilidad del número de nodos y, por consiguiente el incremento de la red, no es tan sencillo. Se debe tener en cuenta el tráfico de la red, el ancho de banda y la latencia en la comunicación, además se debe saber si es el desempeño se incrementará al aumentar el número de nodos.

3.5.1. Escalabilidad de la topología

Se tomó como ejemplo ilustrativo el Cluster MBH'99[19], para ver algunas dificultades que se tienen al incrementar el número de nodos:

La topología que se usó en este ejemplo del cluster es del tipo *estrella hipercubo híbrida (hypercube-star hybrid)*, que es parecidad al *cluster Loki[20]*, que pudo lograr 1.2 Gflops. A grandes rasgos, la topología de hipercubo se basa en lo siguiente:

En un hipercubo de N-dimensiones, cada una de las dos máquinas tiene una liga directa a cada una de sus N máquinas vecinas más cercanas y, una liga indirecta a tadas las otras máquinas. En la figura 3.4, ilustra un hipercubo de 3-dimensiones.

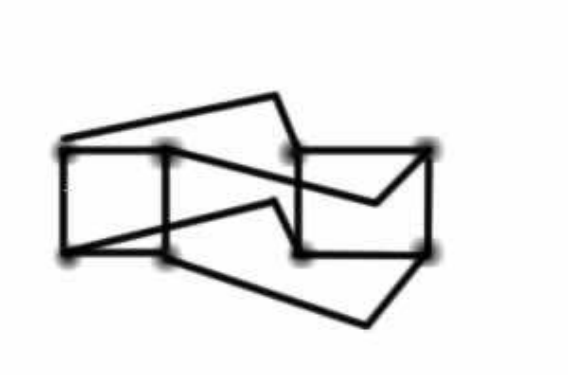

Figura 3.4: Hipercubo de 3-Dimensiones.

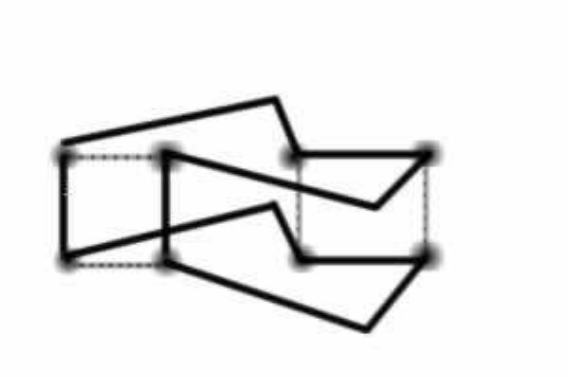

Figura 3.5: Un Anillo mapeado dentro de un hipercubo de 3-Dimensiones.

Además, provee costo efectivo (cost-effective), comunicación eficiente y, *algoritmos paralelos* basados en otras topologías pueden ser mapeados (mapped) dentro de un hipercubo. La figura 3.5 muestra cómo un anillo puede ser mapeado dentro de un hipercubo de 3-D de la figura 3.4.

El costo menos obvio de elegir la topología hipercubo para un cluster Beowulf es la relativa dificultad de sumar nuevos nodos al cluster, comparado a la topología estrella o anillo-estrella híbrida. Algunos la llaman *problema de escalabilidad*.

Para superar el desempeño de una sola máquina, un cluster de PCs de bajo costo debe consistir de muchas máquinas. También, siendo posible sumar PCs ordinarias al cluster es una manera atractiva y barata para incrementar el poder de

Figura 3.6: Topología Bus (Stone SouperComputer de ORNL).

procesamiento de un cluster.

Esta aproximación está siendo usada por la *"Stone SouperComputer"*[24], un cluster Beowulf en los Laboratorios Nacionales Oak Ridge (Oak Ridge National Laboratories, ORNL). Este cluster consiste completamente de máquinas ordinarias (actualmente 128 de ellas). El cluster crece, mientras la gente dona sus máqinas obsoletas para sus necesidades, haciendo la escalabilidad muy importante. Para resolver el problema de la escalabilidad, la *"Stone SouperComputer"* usa la *topología de bus* Ethernet estándar a 10 Mbps mostrada en la figura 3.6.

Este Sistema Beowulf es muy parecido al cluster de Facultad de Ingeniería, con la diferencia de que el cluster de la facultad tiene una topología tipo estrella

Con esta topología, cada máquina en el cluster requiere sólo un puerto de 10Mbps. Si cada máquina ordinaria tiene tal puerto, el costo hardware por nodo es de cero pesos. Por supuesto, su ancho de banda está un poco limitado, pero este cluster está bien para un paralelismo muy funcional (for highly coarse-grained parallelism).

El cluster *Stone SouperComputer* con la topología antes descrita, es muy escalable, pero en el caso del cluster *MBH'99* no es tan fácil. Para sumar máquinas al hipercubo, su dimensión debe incremetarse de *N* a *N+1* (Por ejemplo, de 8 máquinas a 16, de 16 a 32, etc.), la cual suma un puerto al número de puertos de red necesarios por cada máquina. Si dos tarjetas de red (las más baratas) están siendo usadas, entonces una tarjeta adicional debe ser instalada en cada máquina. Más allá del es-

fuerzo físico de abrir el gabinete para instalar la nueva tarjeta sobre N máquinas, la configuración del sistema operativo de cada máquina debe ser también modificado, lo cual es una labor un poco exhaustiva.

El número límite de ranuras (slots) para las tarjetas sobre una tarjeta madre (motherboard), usualmente seis o menos, incrementan el problema. Otros subsistemas (ejemplo, video) también usan estas ranuras, así que, si un cluster *estrella-hipercucbo híbrido* (hypercube-star hybrid) tiene cinco ranuras en la motherboard, un hipercubo de 3-D es lo más alto que se puede lograr (la tarjeta de video y una tarjeta de red ligada al switch, cada una usa una ranura, dejando sólo tres para las ligas directas).
Este problema puede ser direccionado usando tarjetas de red multipuerto (ejemplo, 2-puertos o 4-puertos). Sin embargo, estas tarjetas cuestan más por puerto que una tarjeta de un puerto, así, esto tiene implicaciones en el presupuesto y, cada uno de los sistemas de las máquinas debe ser todavía reconfigurado para usar el nuevo puerto.

La topología *anillo-estrella híbrida* (ring-star hybrid) representa un buen compromiso entre *Stone Souper-Computer* con una escalabilidad fácil, pero ancho de banda bajo, de ORNL y una *estrella-hipercubo híbrida* con una escalabilidad más díficil, pero ancho de banda alto.

Como se describe en la figura 3.7, cada máquina en un *anillo-estrella híbrido* requiere tres puertos de red: dos para las ligas directas y una que liga al switch, sin considerar cuántas máquinas estén en el cluster. Esto llega a ser un beneficio cuando uno desea sumar nuevas máquinas al cluster. Todo lo que es requerido para integrar la nueva máquina es:

- Asegurarse que la máquina tiene tres tarjetas de red;
- Crear una abertura en el anillo para desconectar la máquina M-1 de la máquina 0;

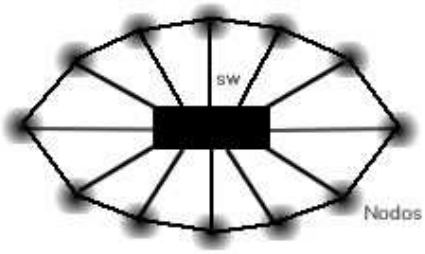

Figura 3.7: Un anillo-estrella híbrido.

- Conectar la máquina 0 a la nueva máquina y, la nueva máquina a la máquina M-1;
- Conectar la nueva máquina al switch; y
- Configurar las tablas de direccionamiento sobre la máquina 0, máquina M-1 y la nueva máquina.

A diferencia de la topología *estrella-hipercubo híbrida*, sólo dos de las máquinas en una topología *anillo-estrella* deben ser modificadas para sumar una nueva máquina.

La única complicación ocurre cuando una nueva máquina está lista para ser sumada y todos los puertos en el switch están ocupados. En este caso, se puede hacer cualquiera de estas dos opciones; el switch puede ser actualizado a uno con más puertos o, un nuevo switch puede ser agregado y los switch unidos vía sus puertos *uplink*. Las máquinas originales pueden entonces ser dejadas en algún lugar o, ser distribuidas igualmente a través de dos switches. La figura 3.8 muestra el cluster de la figura 3.7 con dos nodos adicionales y, las máquinas distribuidas igualmente a través de dos switches de 12 puertos.

Más nodos pueden ser agregados a cada switch. Cuando ambos switches estén ocupados, el proceso puede ser repetido.
Cuando el cluster crece más allá de tres switches, éstos pueden ser organizados dentro de un árbol para que de esta manera se minimice el número de saltos (hops)

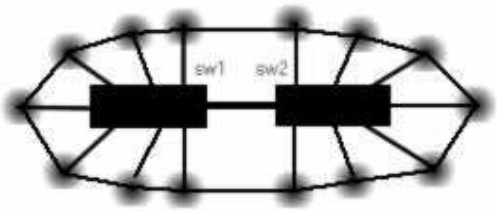

Figura 3.8: Una estrella Doble-Anillo (Multi-Switch) híbrida.

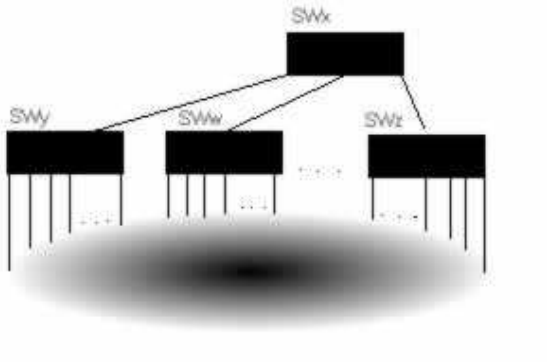

Figura 3.9: Un Multi-Switch Anillo-Árbol híbrido.

entre alguna de las dos máquinas, como se muestra en la figura 3.9.

Esta reorganización puede ser hecha sin modificar algunas de las máquinas del cluster. El anillo-estrella híbrida es, por lo tanto, una topología de cluster extremadamente escalable[19].

Cabe mencionar que, cuando se suma un nuevo nodo al cluster, también se suma memoria y un procesador. El tamaño de la memoria principal y la velocidad de procesamiento del procesador está en función de la aplicación que se ejecute en el cluster y, esto es precisamente lo que proporciona el desempeño de cluster. Como se había mencionado, muchos procesadores con su respectiva memoria, pueden dar el mismo desempeño que si se utilizara la mitad de ellos.

3.5.2. Benchmark de desempeño

Características de desempeño de un nodo de cómputo

Antes de elegir los nodos de cómputo, se debe comprender sus características de desempeño para aplicaciones de intenso cómputo, una de las aplicaciones paralelas típicas sobre un cluster Beowulf. El programa benchmark INTegración Jerárquica (Hierarchical INTegration, HINT), es una herramienta útil para comprender estas características.

HINT usa un método de aproximación para resolver un problema de integración. El objetivo de la integración es obtener la mejor respuesta de calidad en la mínima cantidad de tiempo, para un gran rango de tiempo como sea posible. *Calidad* es el recíproco del error, el cual combina la pérdida de precisión y discretización (discretization) del error.

Durante el proceso de cálculo, la velocidad es definida como Mejoramiento de Calidad Por Segundo (Quality Improvement Per Second, QUIPS), y es calculada como una función de tiempo o de tamaño de memoria del problema. HINT puede ser ejecutada con alguna precisión de algun tipo de dato: Punto-flotante, entero y así sucesivamente[25].

Características de desempeño de un Cluster Beowulf

La versión paralela de HINT también puede ser usada para estudiar las características de desempeño de un *cluster Beowulf*, especialmente cuando éste está construido completamente del mismo tipo de nodos de cómputo. La figura 3.10 muestra las curvas HINT de un cluster Beowulf construido de ocho servidores PowerEdge 6350 y dos versiones de sistemas SGITM Origin2000-series.

El cluster Beowulf tiene 32 procesadores Pentium II a 450MHz. Cada SGI Origin2000 tiene 32 procesadores. La versión más rápida usa procesadores MIPS R10000 a 250 MHz y la más lenta usa procesadores R10000 a 200 MHz. Las curvas HINT muestran que usando los mismos procesadores, el *cluster Beowulf Dell* tiene mucho mejor pico de desempeño que las versiones más lentas de SGI Origin2000, y

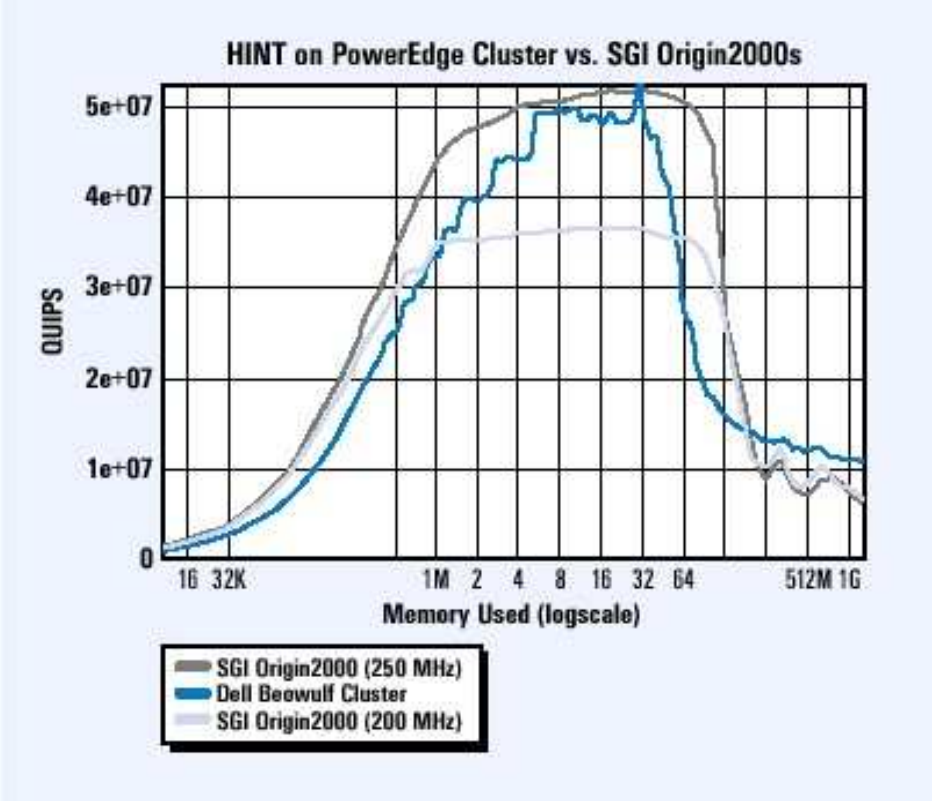

Figura 3.10: HINT sobre un Cluster Beowulf Dell y SGI Origin2000s.

ligeramente un pico de desempeño más bajo que las versiones más rápidas de SGI Origin2000. Las curvas HINT de estas dos SGI Origin2000, tienen un alcance más amplio que el cluster Dell por su cache L2 más grande (4 MB por procesador MIPS contra 2 MB por procesador Pentium II Xeon).

Las curvas HINT de la figura 3.10 también muestran el desempeño del subsistema de memoria de estos tres sistemas. Cuando el tamaño del problema es más grande de 200 MB, el cluster Beowulf Dell tiene mejor desempeño. Esto toma ventaja del subsistema de memoria independiente de los ocho servidores PowerEdge 6350[25].

Capítulo 4
Administración de recursos software

Gran parte de la actividad que se desarrolla en el Área de Sistemas corresponde a la administración del sistema operativo, al soporte a los nodos y al servidor, con los objetivos de garantizar la continuidad del funcionamiento de las máquinas y del *software* al máximo rendimiento.

Para ello se configuró la red de interconexión entre los nodos y el servidor a través del protocolo *DHCP* (Dinamic Host Configure Protocol) de forma que el servidor sea el encargado de asignar la dirección IP correspondiente a cada nodo, en función de la dirección *MAC* (Medium Access Control) de la tarjeta de red Ethernet.

Los nodos añadidos se configuraron de forma que puedan arrancar vía *NFS* (Network File System) a través del protocolo *DHCP*, obteniendo así su dirección IP.
Instalación del software *PVM* (Parallel Virtual Machine) y *XPVM* (A Graphical Console and Monitor for PVM). Configuración de dicho software, de forma que permita la ejecución de programas paralelos basados en el reparto de la carga computacional mediante paso de mansajes entre los distintos nodos del cluster.

Instalación y configuración del software *MPICH* y *LAM/MPI* con el mismo propósito que *PVM*.

4.1. Sistemas de administración de red

Uno de los cambios más destacables del uso de la computadora en los últimos años ha sido la expansión de la conectividad de red con *TCP/IP* desde la mesa de despacho a toda la organización. La infraestructura necesaria para soportar el crecimiento de la red, encaminadores, puentes, conmutadores y concentradores, ha crecido a una velocidad similar.
El personal técnico lucha por mantenerse con las demandas de conectividad y los

cambios, movimientos y reconfiguraciones de red frecuentes, que caracterizan el entorno actual. Estas circunstancias han generado una necesidad de mecanismos que permitan automatizar la configuración de nodos y la distribución del sistema operativo y del software en la red. La forma efectiva de conseguirlo es almacenar los parámetros de configuración e imágenes del software en una o más estaciones de *arranque* de red. Al arrancar, los sistemas interactúan con un servidor de arranque, recogen los parámetros de arranque y, opcionalmente, descargan el software apropiado.

4.1.1. Linux

El sistema operativo es el ambiente de software que permite el acceso al procesador y memoria, provee servicios a las aplicaciones de programas, presenta una interfaz al usuario final, y controla las interfaces externas a los dispositivos y al ambiente externo. Los Beowulfs usan un sistema operativo tipo Unix de código abierto, es decir, un sistema operativo donde el código fuente es disponible de forma gratuita y puede ser estudiado y mejorado por la comunidad. Linux, en particular, es el sistema operativo más popular en los sistemas Beowulf [26].

La tarea mínima de mantenimiento es comprobar regularmente el sistema y los archivos de registro de cada aplicación buscando condiciones de error y eventos inusuales. Por lo general, es posible hacer esto escribiendo un par de scripts de órdenes y ejecutándolos periódicamente mediante la orden *cron*. Se pueden encontrar algunos de estos scripts en las distribuciones fuente de algunas aplicaciones, sólo se tendrán que adecuarlos a las necesidades que se tengan.

La salida de cualquiera de los trabajos de *cron*, se deberá enviar a una cuenta de administración.
La seguridad del sistema comienza con buena administración del mismo. Esto incluye comprobar la propiedad y permisos de todos los archivos y directorios vitales, etc. Cuando un servicio se hace accesible a la red, asegurarse de darle menor privilegio. Esto significa, en una palabra que no se deberán permitir acciones que no

son imprescindibles, para que se trabaje como se diseñó el servicio originalmente. Por ejemplo, el usuario debería hacer sus programas con su cuenta, o alguna otra cuenta privilegiada, sólo si realmente se necesitara. También, si se quiere usar un servicio sólo para una aplicación muy limitada, el administrador del sistema no debe vacilar en configurar el servicio tan restrictivamente como la aplicación especial lo permita. Por ejemplo, si se quiere permitir a máquinas sin disco arrancar desde un nodo en especial, se debe facilitar el servicio **TFTP** *(Trivial File Tranfer Protocol)*; de modo que se puedan obtener los archivos de configuración básicas del directorio */boot*. Sin embargo, cuando se usa sin restringir, TFTP permite a cualquier usuario de cualquier lugar del mundo leer cualquier archivo del sistema.

Otra fuente a considerar deberá ser aquellos programas que permiten registrarse en el sistema, o la ejecución de órdenes con autentificación limitada. Las órdenes *rlogin, rsh y rexec*, son muy útiles pero ofrecen un ligero método de autentificación para aquellos que hagan uso de ellas. Un método de autentificación se basa en la confianza del nombre del nodo llamado, el cual fue obtenido de un servidor de nombres. Las herramientas *ssh* usan un método de auntentificación mucho más confiable, además de proporcionar otros servicios como encriptación y compresión.
En el cluster se decidió instalar Linux Red Hat 6.2 ya que ésta es la versión más actual, que cumple con la condición de no rebasar las capacidades del hardware con el que se cuenta. Ésta versión es la más popular en la construcción de los cluster Beowulf para el ámbito académico.

4.1.2. DHCP Dinamic Host Configure Protocol

Algunas computadoras sólo necesitan unas cuantas variables de configuración antes de arrancar. Otras, puede que deban disponer de una lista detallada, más larga, de valores de parámetros. A veces, las estaciones de trabajo, los hosts con Unix y otros sistemas operativos necesitan descargar completamente los sistemas operativos. Otros sistemas como los encaminadores, puentes, conmutadores o incluso los concentradores puede que necesiten información de configuración de arranque y descargar software.

La inicialización debe ser robusta y flexible. Dependiendo del tamaño de la red, su topología y requisitos de disponibilidad, podría ser más conveniente centralizar la información de arranque en un único servidor, distribuirla por la red en varios servidores o replicarla.
Cada computadora conectada a una red TCP/IP debe conocer la siguiente información:

- Su dirección IP.
- Su máscara de red.
- La dirección IP de un router.
- La dirección IP de un servidor de nombres.

Esta información se guarda normalmente en archivos de configuración, a los que accede el sistema operativo en el arranque; pero, *¿qué ocurre con las computadoras sin disco?*
Podría guardarse el sistema operativo y el software de red en la ROM de la ethernet, pero esa información no es conocida de antemano por el fabricante.
Se dispone de tres protocolos que permitirán transferir esta información por la red:

- **RARP** (Reverse Address Resolution Protocol) sólo proporciona la dirección IP a la computadora sin disco. Debido a esto RARP no está implementado en la mayoría de los sistemas y se ha eliminado totalmente de TCP/IP v6.
- **BOOTP** (Bootstrap Protocol) es un protocolo cliente/servidor diseñado para proporcionar los cuatro tipos de información mencionados anteriormente, a una computadora sin discos.
- **DHCP** (Dinamic Host Configuration Protocol) es una extensión de BOOTP, es decir, la mejora.

El arrancar por red reduce los costos de mantenimiento del software en gran cantidad de máquinas los directorios son mantenidos en el servidor, esto conlleva a la

ventaja de poder ser actualizados en una sola máquina.

*DHCP*es utilizado para grandes redes. El demonio actúa dándole información de la red a las estaciones de trabajo, tales como IP Address, Subnet Mask, DNS Server, Gateway, etc.; ha sido creado por el grupo de trabajo Dynamic Host Configuration del IETF (Internet Enginering Task Force, organización de voluntarios que define protocolos para su uso en Internet).

Al igual que otros protocolos similares, utiliza el paradigma cliente-servidor, para que los nodos clientes obtengan su configuración del nodo servidor.El protocolo de Configuración Dinámica de Hosts (DHCP) permite la transmisión de la configuración de los nodos sobre una red TCP/IP. Este protocolo se encarga de la configuración automática de los parámetros de red, utilizando direcciones.

DHCP es compatible con BOOTP (un cliente puede realizar una petición estática BOOTP a un servidor DHCP).

DHCP es un protocolo que permite asignar direcciones IP dinámicas, de forma totalmente automática. Por ello no pierde las prestaciones de BOOTP, su predecesor, sino que las amplía permitiendo nuevas formas de asignación de direcciones y nuevas opciones para poder pasar a los clientes toda la información necesaria. DHCP es un protocolo implementado en los principales sistemas operativos así como en otros dispositivos.

DHCP está formado por dos partes: de un protocolo para el intercambio de los parámetros de red específicos de cada host y un mecanismo para la asignación de direcciones de red.

Un servidor *DHCP* tiene dos bases de datos. La primera es estática, al igual que BOOTP y la segunda contiene una pila de direcciones IP disponibles. Esta segunda base de datos hace a *DHCP* dinámico. Cuando un cliente *DHCP* pide una dirección

IP temporal, *DHCP* la toma de la pila de direcciones IP disponibles y se la asigna durante un periodo negociado.

El servidor admite tres tipos de configuración de direcciones IP:

1. **Estática**. Se configura en el servidor la dirección de red que corresponde con la dirección LAN del cliente (equivalente a BOOTP).

2. **Dinámica, por tiempo ilimitado**. Se indica un rango de direcciones que se asignan a cada cliente de carácter permanente, hasta que el cliente la libera.

3. **Dinámica, arrendada**. Las direcciones se otorgan por un tiempo ilimitado. Un cliente debe renovar su dirección para poder seguir utilizándola.

Cuando el servidor *DHCP* recibe una petición, primero revisa su base de datos estática. Si existe una entrada para esa dirección física, le devuelve la dirección IP estática correspondiente. Si no encuentra la entrada, el servidor selecciona una IP disponible de la base de datos dinámica y añade la nueva asociación a la base de datos.

- **Alquiler:**

  La dirección asignada desde la pila es temporal. El servidor DHCP emite un alquiler por un periodo determinado de tiempo. Cuando el alquiler termina, el cliente debe dejar de usar la IP o renovar el alquiler. El servidor tiene la opción de aceptar o denegar la renovación.

- **Operación:**
  El cliente realiza los siguientes pasos:

  Envía un mensaje DHCPDISCOVER broadcast usando el puerto destino.

  Aquellos servidores que puedan dar este tipo de servicio responden a un mensaje DHCPOFFER, donde se ofrece una IP que será bloqueada. En estos mensajes también puede ofrecer la duración del alquiler que por defecto es de una hora. Si los clientes no reciben dicho mensaje, intenta

establecer conexión cuatro veces más, cada dos segundos, si aún así no hay respuesta, el cliente espera cinco minutos antes de intentarlo de nuevo.

El cliente elige una de las IPs ofertadas y envía un mensaje DHCPREQUEST al servidor seleccionado.

El servidor responde con un mensaje DHCPACK y crea la asociación entre la dirección física del cliente y su IP. Ahora el cliente usa la IP hasta que el alquiler expire.

Antes de alcanzar el 50 % del tiempo del alquiler, el cliente envía otro mensaje DHCPREQUEST para renovar el alquiler.

Si el servidor responde con DHCPACK, el cliente puede seguir usando la IP durante otro periodo de tiempo. Si se recibe un DHCPNACK, el cliente debe dejar de usar esa IP y empezar de nuevo el proceso de obtención de una IP.

Si después de transcurrir el 87.5 % del alquiler no se recibe respuesta, se manda otro DHCPREQUEST. Si se recibe un DHCPACK antes de que expire el tiempo de alquiler, se obtiene más tiempo de alquiler. En caso contrario, se debe comenzar de nuevo el proceso de obtención de una IP. El cliente puede terminar el alquiler antes de que expire el tiempo. En este caso, el cliente envía un mensaje DHCPRELEASE al servidor.

- **Formato del paquete:**

Para hacer DHCP compatible con BOOTP, los diseñadores de DHCP han usado casi el mismo formato de paquete. Sólo se ha añadido un bit de control al paquete. Sin embargo, se han añadido opciones extra para permitir las diferentes interacciones con el servidor. Los campos diferentes de DHCP son los siguientes:

Flag: 1 bit. El primero del campo sin uso, que permite al cliente el reforzar que la respuesta del servidor sea *broadcast* en vez de *unicast*. Si la respuesta es *unicast*, la dirección destino será la del cliente y este no la conoce, por lo

que puede descartar el mensaje. Al ser *broadcast*, todas las computadoras reciben y procesan el mensaje.

- **Opciones:**

  Se han añadido varias posibilidades a la lista de opciones. La opción con etiqueta 53 es la que define el tipo de interacción entre el cliente y el servidor. Otras opciones definen parámetros con el tiempo de alquiler, etc. El campo de opción en DHCP puede tener hasta 312 bytes.

La asignación de direcciones IP se configura de un modo u otro, dependiendo de cada situación. Puede interesar un direccionamiento estático para clientes sin disco o por facilidades administrativas, pero controlando la asignación de cada dirección a cada cliente (es más cómodo para el administrador configurar un servidor, que cada cliente; interesa el direccionamiento estático para evitar que se conecten clientes no identificados o por otras razones, como la configuración DNS).

El direccionamiento dinámico por tiempo ilimitado se utiliza cuando el número de clientes no varía demasiado, facilitando mucho la tarea del administrador.

El arrendamiento de direcciones se emplea para racionar las direcciones IP, minimizando el costo administrativo. En función de la frecuencia de inserciones/eliminaciones de clientes y de la cantidad de direcciones disponibles se concederá un mayor o menor tiempo de arrendamiento. El tiempo será bajo (ej. 15 minutos); si se conecta/desconectan los clientes con mucha frecuencia e interesa que esté disponible el máximo número de direcciones. Por el contrario se utilizará un tiempo largo para que cada cliente mantenga su dirección IP (ej. en una Universidad un tiempo de 4 meses, tiempo máximo que está desconectado en vacaciones para asumir que el cliente ya no está en la red). Un portátil puede tener una dirección permanente o de larga duración en su red habitual de trabajo y tiempos cortos en otras redes.

La ventaja de instalar un sistema DHCP en la red, ahorra un trabajo de configuración para la red. Todas las computadoras piden información de la red y se configuran automáticamente, muy recomendable para una administración fácil.

`Ejemplo teórico del funcionamiento del arranque por red a través del protocolo DHCP.`

Para que una computadora, que actúa como cliente, pueda arrancar por red, el servidor deberá pasarle la siguiente información:

- Información de red. Que podría estar formada por lo siguiente: dirección IP, servidor de arranque y el archivo del que debe arrancar. Estos datos pueden variar dependiendo de las necesidades de cada uno.

- Un sistema de archivos con el que se trabaja.

- La imagen (núcleo del sistema operativo) para realizar el arranque. Se tiene una red formada por computadoras sin disco teniendo éstos una ROM para arrancar por red, se diferenciarán unos de otros gracias a la dirección Ethernet.

`Un ejemplo de intercambio sería el siguiente`:

- Diskless Computer(DC):Hola, mi dirección hardware es 00:60:08:C7:A3:D8, por favor dame mi dirección IP.

- Servidor DHCP: (busca la dirección en su base de datos). Tu nombre es host1, tu dirección IP es 192.168.1.1. En el caso que necesite un archivo del que se supone que debe arrancar se alojará en */tftpboot/*.

- La petición de DHCP se realiza en forma de broadcast dentro de la red local, de forma que cualquier servidor de DHCP que pudiera responder a la petición, lo haría.

- Después de obtener la dirección IP, la DC debe conseguir la imagen del sistema operativo y lanzarlo a ejecución. En esta fase, se usa otro protocolo TCP/IP,

TFTP (Trivial File Tranfer Protocol). Éste es una versión reducida del FTP (File Transfer Protocol). El TFTP no contempla autentificación, trabaja a través de UDP (User Datagram Protocolo) en vez de TCP (Transmision Control Protocol).

- La implementación de UDP en una computadora de arranque sin disco puede ser suficientemente reducida como para caber en una ROM. Existe la posibilidad de simular esta ROM a través de un disquete. Debido a que UDP es un protocolo orientado a la transmisión por bloques la transferencia se realiza bloque a bloque, de la siguiente forma:

- DC: Dame el bloque 1 de */tftpboot/maquina1*

- TFTP servidor: Aquí lo tienes

- DC: Dame el bloque 2, y así en adelante, hasta que se transfiere el archivo completo para almacenarlo en la RAM.

El funcionamiento consiste, básicamente, en el reconocimiento de cada bloque, y la pérdida de paquetes se soluciona mediante su retransmisión al cabo de un tiempo establecido. Cuando todos los bloques han recibidos, la ROM de arranque de la red pasa el control a la imagen del sistema operativo.

Finalmente, para poner en funcionamiento un sistema operativo, se le debe proporcionar un sistema de archivos raíz. El protocolo utilizado por Linux y otros sistemas UNIX es normalmente NFS (Network File System), aunque no es el único.
En este caso el código no necesita estar grabado en la ROM, sino que forma parte del sistema operativo que se acaba de cargar. El sistema operativo debe ser capaz de ejecutarse con un sistema de archivos raíz NFS, en vez de un disco real.

4.1.3. Etherboot

Etherboot es un paquete software, cuya función es la creación de imágenes ROM que pueden ser descargables a través de una red Ethernet para ser ejecutadas en

computadoras x86. Algunos adaptadores de red tienen un enchufe donde puede ser instalado o conectado un chip ROM.
Etherboot es código que puede ser grabado en una ROM. Etherboot es usado normalmente para realizar arranque sin disco o diskless. Esto es beneficioso en varias situaciones, por ejemplo:

- Una X-Terminal.
- Cluster de computadoras.
- Routers.
- Varias clases de servicios remotos, por ejemplo, un servidor de cinta que sólo puede ser accedido a través del protocolo RMT.
- Máquinas trabajando en entornos desfavorables para los discos duros.
- Plataformas de usuario donde las particiones remotas son montadas a través de la red y se obtienen bajas velocidades en comparación con las de discos.
- Mantenimiento software para cluster de igual configuración a la estación de trabajo central.

Etherboot inicia computadoras más rápidamente que un disquete ya que no hay retardos en los giros del disco, etc. Se puede observar que con una Ethernet de 10Mbit/s se envía un kernel de 500kB en un par de segundos. Con una Ethernet de 100Mbit se obtienen mejores resultados aún.

En comparación con el arranque desde dispositivos como puede ser un disco Flash, Etherboot posee la ventaja de la administración del software centralizado.

Etherboot trabaja con discos RAM, sistemas de archivos NFS, o discos locales. Es un componente tecnológico que puede ser combinado con otras tecnologías para actuar como se desea.

Etherboot se utiliza generalmente para cargar Linux, FreeBSD o el DOS. No obstante los formatos del archivo del protocolo y del cargador del programa inicial son generales.
Los componentes que Etherboot necesita son:

- Un cargador de carga inicial, usualmente una EEPROM de una tarjeta de red o instalado en la flash BIOS.
- Un servidor DHCP o BOOTP, que asigne una dirección IP cuando reciba una dirección MAC.
- Un servidor TFTP, encargado de transmitir la imagen del kernel y otros archivos requeridos durante el proceso de arranque.

Funcionamiento.

A continuación se describe el funcionamiento del software Etherboot.

- Busca un servidor DHCP que en función de su dirección MAC le asignará una dirección IP.
- Una vez asignada la dirección IP, solicitará la transmisión del archivo con la imagen del núcleo. Esta transmisión la realizará el TFTP.
- Recibido el archivo con la imagen del núcleo, será el núcleo el encargado de seguir con el proceso de arranque, es decir, solicitará una IP a través de DHCP y un servidor le asignará en función de su dirección MAC y solicitará la transmisión del sistema de archivos vía NFS.

Los servicios minímos que deben estar corriendo en el cliente una vez que ha arrancado correctamente son: *identd, inet, netfs, network, portmap*. Si se eliminan algunos de estos servicios el cliente no funcionará correctamente.

4.1.4. NFS Network File System

Los clusters Beowulf, por lo general usan el protocolo de sistema de archivos en red, para proveer servicios de sistema de archivos distribuidos.

El Sistema de Archivos de Red es probablemente el servicio de red más prominente que usa RPC. Permite acceder a archivos en anfitriones remotos exactamente en la misma manera que se accedería si fueran locales. Una mezcla de soporte en el núcleo y demonios en espacio de usuario en el lado del cliente, junto con un servidor NFS en el lado del servidor, hace esto posible. Este acceso a los archivos es completamente transparente al cliente y funciona con varias clases de servidores y arquitecturas anfitrionas.

NFS fue desarrollado por Sun Microsystems en los 80's. Linux NFS es principalmente obra de *Rick Sladkey*[29], quien escribió el código del núcleo de NFS y gran parte del servidor de NFS. Lo último se deriva del servidor NFS de espacio de usuario **unfsd**, originalmente escrito por Mark Shand, y el servidor NFS Harris **hnfs**, escrito por Donald Becker.

La segunda versión del protocolo NFS (NFSv2), fué ampliamente adoptada por la mayoría de los sistemas UNIX. El NFSv3, fué publicado en 1993 e implementado en diversos sistemas operativos UNIX, hasta convertirse en el estándar de facto UNIX [27].

Exiten otras alternativas como AFS ó Andrew File System (IBM), y CODA (basado en AFSv2), pero debido a que la primera no es de código abierto y la segunda es actualmente inestable, se usó NFS para la construcción del cluster.

NFS ofrece varias características útiles:

- Los datos accedidos por todos los usuarios pueden mantenerse en un anfitrión central, con los clientes montando este directorio en tiempo de arranque. Por

[29]Se puede contactar a Rick en jsr@world.std.com

ejemplo, se pueden mantener todas la cuentas de usuario en un anfitrión y hacer que todos los anfitriones de la red monten el directorio */home* desde ese anfitrión. Si se instala NFS junto a NIS, los usuarios pueden entrar en cualquier sistema y trabajar en un conjunto de archivos.

- La información que consume grandes cantidades de disco puede mantenerse en un único anfitrión.
- Los datos administrativos pueden almacenarse en el servidor. No hay necesidad de usar *rcp* para instalar el mismo archivo en 20 máquinas diferentes.

NFS funciona de la siguiente forma

Primero, un cliente intenta montar un directorio de un anfitrión remoto en un directorio local justo de la misma manera que si fuese un dispositivo físico. Sin embargo, la sintaxis usada para especificar el directorio remoto es diferente. Por ejemplo, para montar*/home* desde el anfitrión en */users*, el administrador escribe la siguiente orden:

$$\#mount - tnfsscluster : \textbf{\textit{/home /users}}$$

mount tratará de conectar con el demonio remoto sobre **rpc.mountd** en el nodo vía RPC. El servidor verificará si tiene permiso para montar el directorio en cuestión, en cuyo caso, devuelve un descriptor de archivo. Este descriptor será usado en todas las peticiones subsecuentes que se hagan sobre los archivos bajo */users*.
Cuando alguien accede a un archivo sobre NFS, el núcleo manda una llamada de RPC a **rpc.nfsd** (el demonio de NFS) en la máquina servidor. Esta llamada toma el descriptor de archivo, el nombre del archivo a acceder y los identificadores de usuario y grupo del usuario como parámetros. Éstos se usan en la determinación de

los derechos de acceso al archivo especificado. Para prevenir que usuarios no autorizados lean o modifiquen archivos, los identificadores de usuario y grupo deben ser igual en ambos anfitriones.
En la mayoría de las implementaciones de Unix, la funcionalidad NFS de cliente y servidor se implementan como demonios a nivel de núcleo que arrancan desde el espacio de usuario al arrancar la máquina. Éstos son los*Demonios NFS* (**rpc.nfsd**) en el anfitrión servidor, y *Block I/O Demon* (**biod**) en el anfitrión cliente. Para mejorar el rendimiento, biod realiza la E/S usando prelectura y postescritura asíncrona; también, varios demonios **rpc.nfsd** usualmente se ejecutan concurrentemente.

La implementación actual de NFS de Linux es un poco diferente del NFS clásico en la que el código de servidor se ejecuta enteramente en espacio de usuario, así que ejecutar múltiples copias simultáneamente es más complicado. La implementación actual **derpc.nfsd** ofrece una característica experimental que permite soporte limitado para múltiples servidores. Olaf Kirch desarrolló el soporte para servidor NFS basado en el núcleo ofrecido en la versión 2.2 del núcleo de Linux. Su desempeño es significativamente mejor que la de la implementación en el entorno de usuario existente.

Preparando NFS

Antes de que se pueda usar NFS, ya sea como servidor o cliente, se debe asegurar que el núcleo tenga incluido el soporte de NFS compilado. Los núcleos más nuevos tienen una interfaz simple en el sistema de archivos *proc* para esto, el archivo */proc/filesystems*, el cual se puede visualizar usando la orden cat:

$$\$cat/proc/filesystems$$

$$minix$$

$$ext2$$

$$msdos$$

$$nodev \quad proc$$

$$nodev \quad nfs$$

Si falta la palabra nfs en esta lista, se tendrá que compilar su propio núcleo con NFS habilitado, o quizás se necesitará cargar el módulo del núcleo si su soporte de NFS fue compilado como un módulo.

Los Demonios NFS

Si se quiere dar servicio de NFS a otros anfitriones, se deben ejecutar los demonios **rpc.nfsd y rpc.mountd** en la máquina. Como los programas basados en RPC, no son gestionados por *inetd*, sino que son iniciados en tiempo de arranque y se registran a sí mismos con el mapeador de puertos; por consiguiente, se tiene que asegurar de arrancarlos sólo después que **rpc.portmap** se esté ejecutando. Normalmente, se usaría algo como el ejemplo siguiente en uno de los scripts de arranque de red:

$$if \quad [-x/usr/sbin/rpc.mountd]; then$$

$$/usr/sbin/rpc.mountd; echo - nmountd \quad fi$$

$$if \quad [-x/usr/sbin/rpc.nfsd]; then$$

$$/usr/sbin/rpc.nfsd; echo - nnfsd \quad fi$$

La información de propiedad de los archivos que un demonio de NFS proporciona a sus clientes usualmente contiene sólo identificadores numéricos de usuario y de

grupo. Si tanto el cliente como el servidor asocian los mismos nombres de usuario y grupo con esos identificadores numéricos, se dice que éstos comparten el espacio uid/gid . Por ejemplo, éste es el caso cuando usa NIS para distribuir la información **passwd** a todos los anfitriones de su red de área local. Sin embargo, en algunas ocasiones, los IDs en los diferentes anfitriones no coinciden. En lugar de actualizar los uids y gids del cliente para que coincidan con los que hay en el servidor, puede usar el demonio mapeador **rpc.ugidd** para solucionar las disparidades. Usando la opción *map-daemon*, se puede mandar a **rpc.nfsd** que mapee el espacio uid/gid del servidor al espacio uid/gid del cliente con la ayuda de **rpc.ugidd** en el cliente. Desafortunadamente el demonio **rpc.ugidd** no es suministrado con todas las distribuciones modernas de GNU/Linux, así que si se necesita y no se tiene, se necesitará compilarlo a partir de las fuentes.

rpc.ugidd Es un servidor basado en RPC que se inicia desde los scripts de arranque de la red, como **rpc.nfsd** y **rpc.mountd**:

$$if \quad [-x/usr/sbin/rpc.ugidd]; then$$

$$/usr/sbin/rpc.ugidd; echo - nugidd \quad fi$$

rpc.ugidd Es un servidor basado en RPC que se inicia desde sus guiones de arranque de la red, como **rpc.nfsd** y **rpc.mountd**:

$$if \quad [-x/usr/sbin/rpc.ugidd]; then$$

$$/usr/sbin/rpc.ugidd; echo - n"ugidd" \quad fi$$

El Archivo exports

El servidor determina el tipo de acceso que se permite a los archivos del servidor. El archivo*/etc/exports* lista los sistemas de archivos que el servidor permitirá a los clientes montar y usar. Por omisión, **rpc.mountd** desaprueba el montaje de todos los directorios, lo cual es una actitud bastante sensata. Si se desea permitir a uno o más anfitriones montar un directorio de NFS, debe *exportarlo* es decir, especificarlo en el archivo exports. Un ejemplo del archivo puede parecer como éste:

$$\#exports \quad file \quad for \quad vlager$$

$$/home \quad vale(rw) \quad vstout(rw) \quad vlight(rw)$$

$$/usr/X11R6 \quad vale(ro) \quad vstout(ro) \quad vlight(ro)$$

$$/usr/TeX \quad vale(ro) \quad vstout(ro) \quad vlight(ro)$$

$$/ \quad vale(rw, no\_root\_squash)$$

$$/home/ftp \quad (ro)$$

Cada línea define un directorio y los anfitriones a los que se les permite montarlo. Un nombre de anfitrión es usualmente un nombre de dominio completamente cualificado pero puede contener adicionalmente los comodines * y ?. El anfitrión puede ser especificado también usando un rango de direcciones IP en la forma dirección/máscara de redk. Si no se da un nombre de anfitrión, como con el directorio */home/ftp* en el ejemplo previo, cualquier anfitrión coincide y se le permite montar el directorio.

Cuando se verifica a un anfitrión cliente contra el archivo **exports, rpx.mountd** busca el nombre del anfitrión cliente usando la llamada **gethostbyaddr**. Con DNS, esta llamada devuelve el nombre canónico del anfitrión cliente, así debe asegurarse de no usar alias en exports.

En un entorno NIS el nombre devuelto es la primera coincidencia de la base de datos de anfitriones, y sin DNS o NIS, el nombre devuelto es el primer nombre de anfitrión encontrado en el archivo **hosts** que coincida con la dirección del cliente.
El nombre del anfitrión es seguido por una lista opcional de señales separadas por comas, encerradas entre paréntesis. Algunos de los valores que éstas señales pueden tomar son:

- **secure** Esta señal insiste en requerir que se haga desde un puerto origen reservado, por ejemplo, uno que sea menor que 1024. Esta señal está puesta por omisión.
- **insecure** Esta señal revierte el efecto de la señal secure.
- **ro** Esta señal provoca que el montaje de NFS sea para sólo lectura. Esta señal está activada por omisión.
- **rw** Esta opción monta la jerarquía de archivos en lectura-escritura.
- **root_squash** Este característica de seguridad deniega a los superusuarios en los hosts especificados cualquier derecho de acceso especial mapeando las peticiones desde el uid 0 en el cliente al uid 65534 (es decir, -2) en el servidor. Este uid debe ser asociado con el usuario nobody.
- **no_root_squash** No mapea las peticiones desde uid 0. Esta opción está habilitada por omisión, así los superusuarios tienen acceso de supervisar a los directorios exportados de su sistema.
- **link_relative** Esta opción convierte los enlaces simbólicos absolutos (donde el contenido del enlace comienza con un slash) en enlaces relativos. Esta opción sólo tiene sentido cuando está montado el sistema de archivos entero de un anfitrión; por otra parte, algunos de los enlaces podrían apuntar a ninguna parte, o peor aún, a los archivos que nunca debieran apuntar. Esta opción está habilitada de forma predeterminada.

- **link_absolute** Esta opción deja todos los enlaces simbólicos como son (la conducta normal para los servidores de NFS suministrados por Sun).
- **map_identity** Esta opción le indica al servidor asumir que el cliente usa el mismo uid y gid que el servidor. Esta opción está habilitada por omisión.
- **map_daemon** Esta opción indica al servidor de NFS asumir que el cliente y el servidor no comparten el mismo espacio uid/gid. **rpc.nfsd** entonces construye una lista que mapea los IDs entre cliente y servidor preguntando al demonio **rpc.ugidd** del cliente.
- **map_static** Esta opción le permite especificar el nombre de un archivo que contiene un mapa estático de uids. Por ejemplo, map-static=/etc/nfs/vlight.map especificaría el archivo */etc/nfs/vlight.map* como un mapa de uid/gid.
- **map_nis** Esta opción causa que el servidor de NIS haga un mapeado de uid y gid.
- **anonuid y anongid** Estas opciones permiten especificar el uid y el gid de la cuenta anónima. Esto es útil si tiene un volumen exportado para montajes públicos. Cualquier error que ocurra al procesar el archivo **exports** se informa al demonio **syslogd** con el nivel *notice* en cualquier momento en que **rpc.nfsd** o **rpc.mountd** se inicien.

Se ha de tener en cuenta que los nombres del anfitrión se obtienen a partir de la dirección IP del cliente a través de resolución inversa, por lo cual la resolución de nombres tendrá que estar configurada adecuadamente.

Usando un espacio de disco centralizado, los nodos se pueden inicializar desde un disquete, un pequeño disco duro o un sistema de archivos en red. Entonces los nodos pueden acceder a su partición raíz desde un servidor de archivos a través de la red, normalmente usando NFS. Esta configuración funciona mejor en un entorno con mucho ancho de banda en las conexiones y con un gran rendimiento en el servidor de archivos.

La configuración de estos sistemas se realiza en dos partes: cliente y servidor. Las máquinas cliente necesitan un sistema de archivos para montar como raíz. Comúnmente este sistema se haya en un disco local de la máquina. Sin embargo, los sistemas diskless no disponen de una partición raíz, y dependen por tanto de otro sistema para arrancar. NFS es una de las posibles alternativas que permiten arrancar un sistema Linux y montar como sistema raíz un directorio remoto. De esta manera, las máquinas cliente no requieren un disco duro local.

Otro uso de esta facilidad consiste en arrancar un sistema Linux desde una máquina cualquiera con acceso a la red del servidor de NFS.

Optimizando NFS

Normalmente, si las opciones *rsize* y *wsize* no son especificadas, NFS leerá y escribirá en bloques de 4096 ó 8192 bytes. Algunas combinaciones de kernels y tarjetas de red, no pueden manejar estos bloques tan grandes, por lo que serán óptimos. Debe experimentarse hasta encontrar un *rsize* y un *wsize*, que trabaje tan rápido como sea posible. Se probará la velocidad de sus opciones con algunos comandos simples. Dado el comando *mount* con anterioridad y suponiendo que se tiene acceso al disco, puede hacerse lo siguiente para probar el rendimiento de escritura:

$$timeddif = /dev/zeroof = /mnt/testfilebs = 16kcount = 4096$$

Esto crea un archivo de 64 Mb con ceros de contenido este procedimiento debe realizarse varias veces (5-10 veces) para promediar los tiempos. Después se puede probar el rendimiento de lectura leyendo el archivo de regreso:

$$timeddif = /mnt/testfileof = /dev/nullbs = 16k$$

Después de hacer esto unas cuantas veces, se promedian los tiempos. Entonces se desmonta y se monta de nuevo con los parámetros *rsize* y *wsize* más grandes. Deberán ser múltiplos de 1024, y no más grandes de 16384 bytes, este es el tamaño máximo que soporta NFS version 2. Después de montar el directorio remoto nuevamente, se cambia el directorio de montaje (cd) y se ejecutan comandos como *ls*, se explora el sistema de archivos un tiempo razonable para asegurarse de que todo está como se esperaba. Después de verificar que los valores dados *rsize* y *wsize* funcionan, puede probarse nuevamente la velocidad. Algunas plataformas de servidores, tienen óptimos y diferentes tamaños.

Un truco para incrementar la escritura NFS, es deshabilitando las escrituras síncronas en el servidor. Las especificaciones de NFS dicen que las solocitudes de escritura, no serán consideradas hasta que la información escrita, esté en un medio no volátil (normalmente el disco). La escritura asíncrona provocará una aceleración en las escrituras NFS. El demonio *nfsd* de Linux, nunca ha soportado escrituras síncronas, ya que la implementación de archivos de sistema Linux no se presta para esto, pero en servidores con otras plataformas, se puede incrementar este método con la siguiente línea en el archivo */etc/exports*:

$$/dir - async, access = linuxbox$$

NOTA: Esto incrementará el riesgo de pérdida de información.

4.1.5. NIS Network Information Service

Cada vez aumenta el número de máquinas, instaladas como parte de una red de computadoras. Para simplificar la administración de la red, muchas redes (la mayoría de ellas basadas en Sun), corren el Servicio de Información de Red. Las máquinas

Linux, pueden tomar completa ventaja de este sevicio o proveerlo ellas mismas. Éstas máquinas pueden actuar de lleno como clientes NIS+, aunque este soporte se encuentra en estado beta[28].

La elección entre NIS y NIS+ es fácil; se usa NIS si se tienen grandes necesidades de seguridad, ya que NIS+ es más complicado de administrar. Es decir, es sencillo de manejar por el lado del cliente, pero sucede lo contrario cuando se trata del servidor[28].

NIS+ es una nueva versión del servicio de información de red, y es desarrollado por Sun. La mayor diferencia entre NIS y NIS+, es que NIS+ soporta información encriptada y auntentificación sobre RPC seguro. El modelo NIS+, está basado en una estructura de árbol. Cada nodo en el árbol corresponde a un objeto NIS+, de los cuales se tienen 6 tipos: *directorio, entrada, grupo, liga y privado.*

El directorio que conforma la raíz de NIS+, es llamado *directorio raíz.* Existen dos tipos de directorios NIS+ especiales: *org_dir y groups_dir.* El primero consiste en la administración de tablas, tales como: *passwd, mail_alias y hosts*; mientras que el segundo, consta de grupos objeto NIS+, los cuales son usados para control de acceso. La colección de *org_dir, groups_dir* y el directorio padre, es referida como un dominio NIS+[28].

El Sistema de Información de Red (NIS) fue desarrollado por Sun. NIS proporciona prestaciones de acceso a bases de datos genéricas que pueden utilizarse para distribuir, por ejemplo, la información contenida en los archivos *passwd* y *groups* a todos los nodos de su red. Esto hace que la red parezca un sistema individual, con las mismas cuentas en todos los nodos. De manera similar, se puede usar NIS para distribuir la información de nombres de nodo contenida en */etc/hosts* a todas las máquinas de la red.
NIS está basado en *RPC*[30], y consta de un servidor, una biblioteca de la parte cliente,

[30]Remote Procedure Call(Llamadas a Procedimientos Remotos)

y varias herramientas de administración. Originalmente NIS se llamaba *Páginas Amarillas* (Yellow Pages), o YP, que todavía se utiliza para referirse a él. Desafortunadamente, ese nombre es una marca registrada de British Telecom, que exigió a Sun abandonar ese nombre. Al pasar el tiempo, algunos nombres se aferran en la mente de la gente, y así YP permanece como prefijo en los nombres de la mayoría de las órdenes relacionadas con NIS, como **ypserv** y **ypbind**.

NIS guarda la información de la base de datos en archivos llamados *mapas*, que contienen pares clave-valor. Un ejemplo de par clave-valor es el identificativo de un usuario (username) y la forma encriptada de su contraseña. Los mapas se almacenan en un nodo central que corre el servidor NIS, desde él los clientes deben obtener la información mediante varias llamadas RPC. Con bastante frecuencia, los mapas se almacenan en archivos *DBM.*[31]

Los mapas suelen generarse a partir de archivos de texto maestros como el */etc/hosts* o el */etc/passwd*. Para algunos archivos se crean varios mapas, uno para cada tipo de clave de búsqueda. Por ejemplo, se puede buscar en el archivo *hosts* tanto nombres de nodo como direcciones IP. Así pues, de él se derivan dos mapas NIS, llamados *hosts.byname* y *hosts.baddr*. La siguiente Tabla muestra una lista de mapas comunes y los archivos a partir de los que se generan.

Hay mapas para los que la gente usa normalmente *apodos*, que son más cortos y por tanto más fáciles de escribir. Se debe tener en cuenta que estos apodos sólo los entienden **ypcat** e **ypmatch**, dos herramientas para comprobar su configuración NIS. Para obtener una lista completa de los apodos que entienden estas herramientas, se ejecuta la siguiente orden:

$$\$ypcat - x$$

$$Use \text{“}passwd\text{”} for \text{“}passwd.byname\text{”}$$

[31] Es una sencilla biblioteca para manejo de datos que usa técnicas de dispersión (hashing) para aumentar la velocidad de las operaciones de búsqueda. Existe una implementación libre de DBM del proyecto GNU llamada gdbm, que es parte de la mayoría de las distribuciones de Linux.

$$Use\text{“}group\text{”}\ for\text{“}group.byname\text{”}$$

$$Use\text{“}networks\text{”}\ for\text{“}networks.byaddr\text{”}$$

$$Use\text{“}hosts\text{”}\ for\text{“}hosts.byaddr\text{”}$$

$$Use\text{“}protocols\text{”}\ for\text{“}protocols.bynumber\text{”}$$

$$Use\text{“}services\text{”}\ for\text{“}services.byname\text{”}$$

$$Use\text{“}aliases\text{”}\ for\text{“}mail.aliases\text{”}$$

$$Use\text{“}ethers\text{”}\ for\text{“}ethers.byname\text{”}$$

El servidor NIS se llama tradicionalmente **yperv**. Para una red mediana, normalmente un solo servidor es suficiente; las redes grandes pueden elegir ejecutar varios de estos servidores en máquinas diferentes y en segmentos de red diferentes para reducir la carga en las máquinas servidor y en los enrutadores. Estos servidores se sincronizan haciendo a uno de ellos el *servidor maestro*, y a los otros *servidores*

Archivo Maestro	**Mapa(s)**	**Descripción**
/etc/hosts	hosts.byname, hosts.byaddr	Corresponde direcciones IP con nombres de nodo
/etc/networks	networks.byname, networks.byaddr	Corresponde direcciones IP de red con nombres de red
/etc/passwd	passwd.byname, passwd.byuid	Corresponde contraseñas encriptadas con indicativos de usuario
/etc/group	group.byname, group.byuid	Corresponde IDs de grupo con nombres de grupo
/etc/services	services.byname, services.bynumber	Corresponde descripciones de servicio con nombres de servicio
/etc/rpc	rpc.byname, rpc.bynumber	Corresponde números de servicio Sun RPC con nombres de servicio RPC
/etc/protocols	protocols.byname, protocols.bynumber	Corresponde números de protocolo con nombres de protocolo
/usr/lib/aliases	mail.aliases	Corresponde alias de correo con nombres de alias de correo

Cuadro 4.1: Tabla de algunos mapas NIS estándar y sus archivos correspondientes

esclavos. Los mapas se crean sólo en el nodo del servidor maestro. Desde él se distribuyen a todos los esclavos.

Hay un término distintivo en NIS que se refiere a una colección de todos los nodos que comparten parte de sus datos de configuración de sistema a través de NIS: el *dominio NIS*.
Los dominios NIS tienen una función puramente administrativa. En general son transparentes a los usuarios, excepto al compartir contraseñas entre todas las máquinas del dominio. Por tanto, el nombre que se le da a un dominio NIS es relevante sólo para los administradores. Normalmente, cualquier nombre servirá, mientras sea distinto a cualquier otro dominio NIS de la red local. Otro proceder común es usar simplemente el dominio DNS como dominio NIS.
Para establecer y mostrar el dominio NIS de su nodo, puede usar la orden *domainname*. Cuando se invoca sin argumentos, imprime el dominio NIS actual; para establecer el dominio, hace falta ser superusuario.

Los dominios NIS determinan a qué servidor NIS consultará una aplicación. Por ejemplo, el programa **login** de un nodo del cluster debe, por supuesto, consultar sólo al servidor NIS del cluster (o a uno de ellos, si hay varios) la contraseña de un usuario.
Pero, *¿cómo averigua un cliente a qué servidor conectarse?* La solución más simple sería utilizar un archivo de configuración que diga el nombre del nodo que hace de servidor. Sin embargo, esta solución es algo inflexible porque no permite a los clientes utilizar diferentes servidores (del mismo dominio, claro) dependiendo de su disponibilidad. Por tanto, las implementaciones de NIS cuentan con un demonio especial llamado **ypbind** para detectar un servidor NIS adecuado dentro del dominio NIS. Antes de realizar una consulta a NIS, una aplicación averigua primero qué servidor usar mediante **ypbind**.

ypbind busca servidores haciendo un *broadcast* a la red IP local; se asume que el primero en responder es el más rápido, y es el utilizado en todas las consultas NIS

subsiguientes. Después de que ha transcurrido un cierto intervalo de tiempo, o si el servidor deja de estar disponible, **ypbind** busca de nuevo servidores activos.

La Parte Cliente en NIS

Si se está familiarizado con escribir o portar aplicaciones de red, se puede notar que la mayoría de los mapas NIS listados anteriormente corresponden a funciones de la biblioteca C. Por ejemplo, para obtener la información de **passwd**, generalmente se utilizan las funciones **getpwnam** y **getpwuid**, que devuelven la información de cuenta asociada con el nombre de usuario o el ID numérico de usuario, respectivamente. Bajo circunstancias normales, estas funciones realizan la búsqueda requerida en el archivo estándar,*/etc/passwd*.
Sin embargo, una implementación NIS de estas funciones modifica este comportamiento y realiza una llamada RPC al servidor NIS, que busca el nombre de usuario o el ID de usuario. Esto ocurre transparentemente para la aplicación. La función puede tratar a los datos NIS como si hubiesen sido añadidos al archivo original */etc/passwd* por lo que ambos juegos de información están disponibles para la aplicación, o como si lo hubiese reemplazado completamente, por lo que la información del **passwd** local es ignorada y sólo se utilizan los datos de NIS.

Ejecutando un Servidor NIS

Existen dos configuraciones posibles del servidor NIS: maestra y esclava. La configuración esclava es una máquina que proporciona una copia de seguridad, por si el servidor maestro falla. Después de instalar el programa (ypserv) en */usr/sbin*, debe crear el directorio que contendrá los archivos de mapas que va a distribuir su servidor. Al configurar un dominio NIS para el dominio scluster, los mapas irían en */var/yp/scluster*. El servidor determina si está sirviendo un dominio NIS particular comprobando si existe el directorio de los mapas. Si quiere deshabilitar el servicio para algún dominio NIS, asegúrese de eliminar el directorio. Normalmente los ma-

pas se almacenan en archivos DBM para agilizar las búsquedas. Se crean a partir de los archivos maestros utilizando un programa llamado **makedbm** (del servidor de Tobias) o **dbmload** (del servidor de Peter).

El paquete **ypserv** de Peter Eriksson contiene un Makefile (llamado ypMakefile) que se encarga por usted de la conversión de la mayoría de los archivos maestros. Se debe instalar como Makefile en el directorio de mapas y editarlo para reflejar los mapas que quiere que el servidor NIS comparta. Al principio del archivo encontrará el objetivo *all* que lista los servicios que ofrece ypserv. Por defecto la línea se parecerá a esto:

$$all : ethers hosts networks protocols rpc services passwd group netid$$

Si no quiere producir, por ejemplo, los mapas **ethers.byname** y **ethers.byaddr**, simplemente se borra el prerrequisito ethers de esta regla. Para comprobar su configuración, puede empezar con sólo uno o dos mapas, como los mapas **services**. Después de editar el **Makefile**, estando en el directorio de mapas, se teclea **make**. Esto generará automáticamente los mapas y los instalará. Se debe asegurar de actualizar los mapas cada vez que cambien los archivos maestro, o de otra manera los cambios permanecerán invisibles a la red.

Seguridad en el Servidor NIS

NIS solía tener un defecto grave de seguridad: dejaba su archivo de contraseñas legible por prácticamente cualquier persona en toda Internet, lo que suponía un gran número de posibles intrusos. Si un intruso sabía su dominio NIS y la dirección de su servidor, simplemente tenía que enviar una consulta al mapa **passwd.byname** y recibir al instante todas las contraseñas encriptadas del sistema. Con un programa rápido para *crackear* contraseñas con el *crack*, y un buen diccionario, averiguar unas

cuantas contraseñas de usuario no es problema.
De todo esto trata la opción *securenets*. Esta opción simplemente restringe el acceso a su servidor NIS a ciertos nodos, basándose en su dirección IP o números de red.

La última versión de **ypserv** implementa esta característica de dos maneras. La primera consta de un archivo de configuración especial llamado */etc/ypserv.securenets* y la segunda utiliza convenientemente los archivos */etc/hosts.allow* y */etc/hosts.deny*[32]. Así, para restringir el acceso a los nodos de dentro del cluster, el administrador de red añadiría esta línea al *hosts.allow*:

$$ypserv : 172.\ 16.\ 2$$

Esto permitiría a todos los nodos de la red 172.16.2.0 acceder al servidor NIS. Para denegar el acceso al resto de nodos, la correspondiente línea en el **hosts.deny** sería:

$$ypserv : ALL$$

Las direcciones IP no son la única manera de especificar nodos y redes en **hosts.allow** y ***hosts.deny***[33]. Sin embargo, advierta que *no puede* utilizar nombres de nodo o de dominio en la entrada **yperv**.

Para configurar la seguridad **securenets** utilizando el método */etc/ypserv.securenets*, se necesita crear el archivo de configuración,*/etc/ypserv.securenets*. Este archivo de configuración es simple en su estructura. Cada línea describe un nodo o red de nodos que tendrán permiso de acceso al servidor. Cualquier dirección no descrita con una entrada en este archivo tendrá denegado el acceso. Una línea que comience por # será tratada como comentario. El siguiente ejemplo muestra cómo sería un sencillo archivo */etc/ypserv.securenets*:

[32]Para habilitar el uso del método */etc/hosts.allow*, puede que tenga que recompilar el servidor. Por favor, lea las instrucciones del archivo README incluido en la distribución

[33]Si requiere de más información, consulte la página del manual **hosts_access(5)** de su sistema

Ejemplo archivo ypserv.securenets

#permitir conexiones desde el nodo local −− *necesario*

*host*127. 0. 0. 1

#lo mismo para 255. 255. 255. 255. 127. 0. 0. 1

#permitir conexiones desde cualquier nodo de la red del Cluster

255. 255. 255. 0 172. 16. 1. 0

La primera entrada de cada línea es la máscara de red a utilizar, siendo host una palabra clave especial que significa “máscara de red 255.255.255.255&rdquo. La segunda entrada de cada línea es la dirección IP a la que aplicar la máscara de red.

Una tercera opción es utilizar el mapeador de puertos (portmapper) seguro en lugar de la opción **securenets** de **ypserv**. El *mapeador de puertos seguro (portmap-5.0)* [34] utiliza también el esquema de **hosts.allow**, pero ofrece esto a todos los servidores RPC, no sólo a ypserv. Sin embargo, no se debe utilizar la opción **securenets** el mapeador de puertos seguro al mismo tiempo, por la sobrecarga que esto supondría.

Configurando un Cliente NIS con la libc de GNU

El primer paso debe ser decirle al cliente NIS de la libc de GNU qué servidor usar para el servicio NIS. El comportamiento predeterminado es consultar al servidor de la red local. Si es probable que el nodo que está configurando se vaya a mover de un dominio a otro, como un portátil, se debería dejar el archivo */etc/yp.conf* vacío, y el nodo consultará en la red local qué servidor NIS es el que procede.

Una configuración más segura para la mayoría de nodos es especificar el nombre del servidor en el archivo de configuración */etc/yp.conf*. Un archivo muy sencillo para

[34] El mapeador de puertos seguro está disponible vía FTP anónimo en ftp.win.tue.nl en el directorio */pub/security/*.

un nodo de la red del Cluster sería así:

 # yp.conf- configuración de YP para la biblioteca GNU libc.
ypserver servidor
La sentencia *ypserver* le dice a su nodo que use el nodo especificado como servidor NIS para el dominio local. En este ejemplo se ha especificado servidor como servidor NIS. Por supuesto, la dirección IP correspondiente a servidor debe especificarse en el archivo hosts; alternativamente, puede usar la propia dirección IP con el argumento server. En la forma que se muestra en el ejemplo, la orden **ypserver** le dice a **ypbind** que use el servidor nombrado sin tener en cuenta cuál es el dominio NIS actual.

Utilizando los Mapas passwd y group

Una de las aplicaciones más importantes de NIS es sincronizar la información del usuario y de su cuenta en todos los nodos de un dominio NIS. Por consiguiente, normalmente sólo se mantendrá un archivo */etc/passwd* pequeño, al cual se añade la información global de los mapas NIS. Sin embargo, no es suficiente con habilitar las búsquedas NIS para este servicio en el archivo **nsswitch.conf**. Antes de fiarse de la información de contraseñas distribuida por NIS, se debe asegurar de que todos los números ID de usuario que haya en el archivo local **passwd** concuerdan con los del servidor NIS. La consistencia de los IDs también es importante para otros propósitos, como montar particiones NFS desde otros nodos de su red.

4.2. Software de Comunicación

El desarrollo de las principales bibliotecas de computación paralela se ha basado, fundamentalmente, en una máquina virtual o bien en el paso de mensajes. Solamente se van a instalar los paquetes más comúnes. Un cluster es una colección de memoria local de máquinas; el único camino para que un nodo A se comunique con un nodo B es a través de la red. Se construyó un software para esta arquitectura "paso de

mensajes"entre nodos. Existen dos populares librerias de paso de mensajes: **PVM y MPI**.
PVM se desarrolló basándose en el concepto de máquina virtual, esto es, una colección de recursos computacionales manejados como una computadora paralela. Por su parte, **MPI** se centró en el paso de mensajes, y aunque no tiene el concepto de máquina virtual, sí hace una abstracción de todos los recursos en términos de topología de paso de mensajes.

El desarrollo de la biblioteca de programación paralela **MPI**(Message Passing Interface) ha causado que muchos programadores se planteen el desarrollar programas usando MPI o bien PVM (Parallel Virtual Machine).

Hasta la fecha, **PVM** ha venido siendo la biblioteca estándar para computación distribuida. Sin embargo, **MPI** está siendo cada vez más utilizada, convirtiéndose en el nuevo estándar para este tipo de programación.

Para paralelizar una aplicación es necesario contar con un *lenguaje o biblioteca* que brinde las herramientas necesarias para esto. Dependiendo de la herramienta con que se cuente, se particionará el código en piezas para que se ejecute en paralelo en varios procesadores.

4.2.1. PVM Parallel Virtual Machine

PVM (Máquina Paralela Virtual) es una herramienta diseñada para solucionar una gran cantidad de problemas asociados con la programación paralela. Sobre todo, el monetario. Para ello, crea una nueva abstracción, que es la máquina paralela virtual, empleando los recursos computacionales libres de todas las máquinas de la red que pongamos a disposición de la biblioteca.

La Máquina Paralela Virtual es una máquina que no existe, pero un API apropiado nos permite programar como si existiese. El modelo abstracto que permite usar el API de la PVM consiste en una máquina multiprocesador completamente es-

calable (es decir, que podemos aumentar y disminuir el número de procesadores rápidamente)[6] . Para ello, se oculta la red que se está empleando para conectar las máquinas, así como las máquinas de la red y sus características específicas. Este planteamiento tiene numerosas ventajas respecto a emplear una supercomputadora, de las cuales, las más destacadas son:

- **Precio**. Así como es más barato una computadora paralela que la computadora tradicional equivalente, un conjunto de computadoras de mediana o baja potencia equivalente. Al igual que ocurrirá como el caso de la computadora paralela, van a existir factores (fundamentalmente, la lentitud de la red frente a la velocidad del bus de la computadora paralela) que van a hacer que sean necesarios más computadoras de pequeña potencia que los teóricos para igualar el rendimiento. Sin embargo, aún teniendo esto en cuenta, la solución es mucho más barata. Además, al no ser la PVM una solución que necesite de máquinas dedicadas (es decir, el daemon de PVM corre como un proceso más), se puede emplear en el proceso los tiempos muertos de los procesadores de todas las máquinas de la red a las que se tenga acceso. Por ello, si ya se tiene una red UNIX montada, el costo de tener una supercomputadora paralela va a ser cero ya se disponen de las máquinas, no se tendrá que comprar nada nuevo, y además la biblioteca PVM es un software libre, por lo que no hay que pagar por usarla.

- **Disponibilidad**. Todo centro de cálculo tiene un mínimo de una docena de máquinas arrumbadas en una esquina, y nadie sabe qué hacer exactamente ya con ellas. Con esa docena que hace seis años que ya no corren ni la última versión del Word para Windows, se puede instalar Linux, la PVM y añadirlo a la supercomputadora paralela virtual que conforma las máquinas que ya se tendrían en red.

- **Tolerancia a fallos**. Si por cualquier razón falla una de las computadoras que conforman nuestras PVM y el programa que la usa está razonablemente bien hecho. La aplicación puede seguir funcionando sin problemas. Siempre hay alguna razón por la que alguna máquina puede fallar, y la aplicación debe

continuar haciendo los cálculos con aquel hardware que continúe disponible.

- **Heterogeneidad**. Se puede crear una máquina paralela virtual a partir de computadoras de cualquier tipo. La PVM abstrae la topología de la red, la tecnología de la red, la cantidad de memoria de cada máquina, el tipo de procesador y la forma de almacenar los datos. Por último incluye en la PVM hasta máquinas paralelas. Una máquina paralela en una PVM se puede comportar tanto como una sola máquina secuencial (por ejemplo, el soporte SMP de Linux) o, como ocurre en muchas máquinas paralelas, presentarse a la PVM como un conjunto de máquinas secuenciales.

- **Disponibilidad**. La disponibilidad de la PVM es completa. Se ha encontrado con facilidad para PowerPC con AIX, Sun con Solaris y PC 80x86 con Linux.

El uso de la PVM tiene muchas ventajas, pero también tiene una gran desventaja: se puede olvidar del paralelismo fuertemente acoplado. Si se dispone de una red Ethernet, simplemente la red va a dejar de funcionar para todas las aplicaciones (incluida la PVM) de la cantidad de colisiones que se van a producir en caso de que se intente el paralelismo fuertemente acoplado. Si se dispone de una red de tecnología más avanzada; es decir, más cara (como ATM) el problema es menor, pero sigue existiendo.

La segunda desventaja es que la abstracción de la máquina virtual, la independencia del hardware y la independencia de la codificación tienen un costo. La PVM no va a ser tan rápida como son los Sockets. Sin embargo, si el grado de acoplamiento se mantiene lo suficientemente bajo, no es observable esta diferencia.

La arquitectura de la PVM se compone de dos partes. La primera es el daemon, llamado *pvmd*. En la versión 3 de pvm, el nombre es *pvmd3*. El daemon ha de estar funcionando en todas las máquinas que vayan a compartir sus recursos computacionales con la máquina paralela virtual. A diferencia de otros daemons y programas del sistema, el daemon de la PVM puede ser instalado por el usuario en su directorio particular. Esto permite hacer supercomputación como usuarios, sin que se tenga

que discutir con el administrador de la red qué programas se pueden ejecutar. Una vez que se instaló PVM, todos los usuarios pueden hacer uso de esa instalación con el requisito de que el directorio donde esté instalada PVM sea de lectura al usuario que quiera hacer uso de ella.

En la mayoría de los casos, el administrador prefiere instalar él mismo PVM; con lo que, además de evitar que un usuario pueda borrarla sin consultar a los demás, va a permitir que todos los usuarios tengan la PVM instalada por defecto; y, lo que es más importante, el administrador puede determinar el valor de *nice* (prioridad del daemon) con el que va a ser lanzado el daemon *pvmd3* y así, si este valor de *nice* es lo suficientemente alto, permite que la máquina ejecute la PVM solamente en los momentos ociosos.

Este daemon *pvmd3* es el responsable de la máquina virtual de por sí, es decir, de que se ejecuten los programas para la PVM y de administrar los mecanismos de comunicación entre máquinas, la conversión automática de datos y de ocultar la red al programador. Por ello, una vez que la PVM esté en marcha, el paralelismo es independiente de la arquitectura de la máquina, y sólo depende de la arquitectura de la máquina virtual creada por la PVM.

Cada usuario, arrancará el daemon como si de un programa normal se tratara, para ejecutar el código de PVM. Este programa se queda residente, realizando las funciones anteriores.

La segunda parte es la biblioteca de desarrollo. Contiene las rutinas para operar con los procesos, transmitir mensajes entre los procesadores y alterar las propiedades de la máquina virtual. Toda aplicación se enlaza a la biblioteca para poderse ejecutar después. Se tendrán tres archivos de bibliotecas, *libpvm3.a* (biblioteca básica en C), *libgpvm3.a* (biblioteca de tratamiento de grupos) y *libfpvm3.a* (biblioteca para Fortran).

Un programa para PVM va a ser un conjunto de tareas que cooperan entre sí. Las tareas se intercambian información empleando paso de mensajes. La PVM, de forma transparente al programador, va a ocultar las transformaciones de tipos asociadas al paso de mensajes entre máquinas heterogéneas. Toda tarea de la PVM puede incluir o eliminar máquinas, arrancar o parar otras tareas, mandar datos a otras tareas o sincronizarse con ellas.

Cada tarea en la PVM tiene un número que la identifica unívocamente, denominado *TID*[35]. Es el número al que se mandan los mensajes habitualmente. Sin embargo, no es el único método de refenciar una tarea en la PVM. Muchas aplicaciones paralelas necesitan hacer el mismo conjunto de acciones sobre un conjunto de acciones sobre un conjunto de tareas. Por ello, la PVM incluye una abstracción nueva, el *grupo*. Un grupo es un conjunto de tareas las cuales se pueden referir con el mismo código, el identificador del grupo. Para que una tarea entre o salga de un grupo, basta con avisar de la salida o entrada al grupo. Esto denota un mecanismo muy cómodo y potente para realizar programas empleando modelos *SIMD*[36], en el que se dividen los datos en muchos datos pequeños que sean fáciles de tratar, y después se va a codificar la operación simple y replicarla tantas veces como datos unitarios se tengan de dividir el problema. Para trabajar con grupos además de enlazar la biblioteca de la PVM (*libpvm3.a* se tiene que enlazar también la de los grupos *libgpvm3.a*).

Habitualmente para arrancar un programa para la PVM, se lanzará manualmente desde una computadora contenida en el conjunto de máquinas una tarea madre. La tarea se lanzará con el comando *spawn* desde un monitor de la máquina virtual, que a su vez se activará con el comando *pvm*. Esta tarea se encargará de iniciar todas las demás tareas, bien desde su función *main*, bien desde alguna subrutina invocada por ella. Para lanzar nuevas tareas se emplea la función *pvm-spawn*, que devolverá un código de error, asociado a si pudo o no crearla, y el TID de la nueva tarea.

[35]Task Identification Number.
[36]Single Instruction, Multiple Data

Para evitar el engorro de andar realizando transformaciones continuas de datos, la PVM define clases de arquitecturas. Antes de mandar un dato a otra máquina comprueba su clase de arquitectura. Si es la misma, no necesita convertir los datos, con lo que se tiene un gran incremento en el rendimiento. En caso que sean distintas las clases de arquitectura se emplea el protocolo *XDR*[37]

Las clases de arquitectura están mapeadas en números de codificación de datos, que son los que realmente se transmiten y, por lo tanto, los que realmente determinan la importancia de la conversión.

El modelo de paso de mensajes es transparente a la arquitectura para el programador, por la comprobación de las clases de arquitectura y la posterior codificación con XDR de no coincidir las arquitecturas. Los mensajes son etiquetados al ser enviados con un número entero definido por el usuario, y pueden ser relacionados por el receptor tanto por dirección de origen como por el valor de la etiqueta.

El envío de mensajes no es bloqueante. Esto quiere decir, que el que envía el mensaje no tiene que esperar a que el mensaje llegue, sino que solamente espera a que el mensaje sea puesto en la cola de mensajes. La cola de mensajes, además, asegura que los mensajes de una misma tarea llegarán en orden entre sí. Esto no es trivial, ya que empleando ***UDP***[38] puede que se envíe dos mensajes y que lleguen fuera de orden.**TCP**, por ser un protocolo orientado a la conexión, realiza una reordenación de los mensajes antes de pasarlos a la capa superior, sin embargo, tiene el inconveniente que establecer las conexiones entre nodos empleando TCP supone, si tenemos n nodos, se tendrá un mínimo de ($n)(n$-1) conexiones TCP activas. Provocando esto que hasta para números ridículos de n se quede sin puertos por este planteamiento. Establecer conexiones TCP entre procesos en lugar de entre nodos es peor todavía, por las mismas razones que en el caso de los nodos.

La comunicación de las tareas con el daemon se hace empleando TCP. Esto se

[37]External Data Representation
[38]UDP es un protocolo no orientado a conexión

debe a que, al ser comunicaciones locales, la carga derivada de la apertura y cierre de un canal es muy pequeño. Además, no se van a tener tantas conexiones como en el caso de la conexión entre daemons, ya que las tareas no se conectan entre sí ni con nada fuera del nodo, por lo que sólo hablan directamente con su daemon. Esto determina que serán n conexiones TCP, que sí es una cifra razonable.

La recepción de los mensajes se pueden hacer mediante primitivas bloqueantes, no bloqueantes o con un tiempo máximo de espera. La PVM ofrece primitivas para realizar estos tres tipos de recepción. En principio son más cómodas las bloqueantes, ya que dan un mecanismo de sincronización bastante cómodo. Las de tiempo máximo de espera son útiles para trabajar con ellas como si fuesen bloqueantes, mas dando soporte al hecho de que puede que el que tiene que mandar el mensaje se haya colgado. Por último, la recepción de mensajes mediante primitivas no bloqueantes hace de la sincronización un dolor de cabeza. De cualquier forma, en los tres casos anteriormente citados la misma PVM se encargará de decir cuándo una tarea se acabó. Para informar de lo que pasa, emplea un mecanismo de eventos asíncronos.

La PVM puede ser empleada de forma nativa como funciones en C y en C++, y como procedimientos en Fortran. Basta para ello con tomar las cabeceras necesarias (si se trabaja con C o C++); y, para los tres, enlazar con la biblioteca adecuada, que viene con la distribución estándar. En el caso de C es *libpvm3.a* y en el de Fortran *libfpvm3.a*.

Si se desea trabajar con otros lenguajes puede ser un poco más complejo. Si el lenguaje permite incorporar funciones nativas en lenguaje C (como es el caso, por ejemplo, de Java) no hay ningún problema; ya que se puede invocar la función; bien directamente si el lenguaje lo permite, haciendo una pequeña rutina para adaptar el tipo de los datos, el formato de llamada a función o cualquiera de las restricciones que imponga el lenguaje que se emplee para invocar funciones en C.

Cabe señalar que toda función en C *pvm_algunacosa* tiene como equivalente en

Fortran *pvmf-algunacosa*, y viceversa.

El programa PVM corresponde al intérprete de comandos de la máquina virtual. Algunos de los comandos más importantes son:

- **add**: máquina: Incorpora la máquina indicada a la máquina paralela virtual.
- **delete**: máquina: Elimina la máquina indicada del conjunto de máquinas asociadas a la máquina paralela virtual. Como es lógico, no se puede eliminar la máquina desde la que se está ejecutando el intérprete de comandos.
- **conf**: Configuración actual de la máquina paralela virtual.
- **ps**: listado de procesos de la máquina paralela virtual. *ps -a* lista todos los procesos.
- **halt**: Apaga la máquina paralela virtual. Esto significa que mata todas las tareas de la PVM, elimina el daemon de forma ordenada y sale del programa pvm.
- **help**: Lista los comandos del programa.
- **id**: imprime el TID de la consola.
- **jobs**: Genera un listado de los trabajos de ejecución.
- **kill**: Mata un proceso de la PVM.
- **mstat**: Muestra el estado de una máquina de las pertenecientes a la PVM.
- **pstat**: Muestra el estado de un proceso de los pertenecientes a la PVM.
- **quit**: Sale de la máquina paralela virtual sin apagarla.
- **reset**: Inicializa la máquina. Eso supone matar todos los procesos de la PVM salvo los programas monitores en ejecución, limpiar las colas de mensajes y las tablas internas y pasar a modo de espera todos los servidores.

- **setenv**: Lista todas las variables de entorno del sistema.
- **sig** señal tarea: Manda una señal a una tarea.
- **spawn**: Arranca una aplicación bajo PVM. Es un comando bastante complejo.
- **alias**: Define un alias predefinido, es decir, un atajo para teclear un comando.
- **unalias**: Elimina un alias predefinido.
- **version**: Imprime la versión usada de la PVM.

Configuración PVM 3.4.3-4

En el directorio del usuario se tendrá que crear la siguiente estructura de directorios
$HOME/pvm3/bin/LINUX. Se modificará el archivo *.bashrc* quedando de la siguiente forma:

```
# .bashrc
# Source global definitions
if [ -f /etc/bashrc ]; then
. /etc/bashrc
fi
# append this file to your .profile to set path according to machine
# type. you may wish to use this for your own programs (edit the last part to point
to a different directory f.e.  /bin/_$PVM_ARCH.
export PVM_ROOT=/usr/share/pvm3
export XPVM_ROOT=$PVM_ROOT/xpvm

if [ -z $PVM_ROOT ]; then
if [ -d ~/pvm3 ]; then
export PVM_ROOT= /pvm3
else print “Warning - PVM_ROOT not defined”
```

print “To use PVM, define PVM_ROOT and rerun your .profile”fi fi

if [-n $PVM_ROOT]; then
export PVM_ARCH= $PVM_ROOT/lib/pvmgetarch
uncomment one of the following lines if you want the PVM commands
directory to be added to your shell path.
export PATH=$PATH:$PVM_ROOT/lib # generic
export PATH=$PATH:$PVM_ROOT/lib/$PVM_ARCH # arch-specific
uncomment the following line if you want the PVM executable directory
to be added to your shell path.
export PATH=$PATH:$PVM_ROOT/bin/$PVM_ARCH
export PATH=$PATH:$HOME/pvm3/bin/$PVM_ARCH
fi

A continuación se creará el archivo *.pvmrc* , en el cual se incluirán el nombre de los nodos que van a formar la PVM. Dicho archivo tendrá la siguiente estructura:
example PVM console startup script
copy this file to $HOME/.pvmrc
command aliases
alias ? help
alias print_environment spawn -¿/bin/env
alias h help
alias j jobs
alias t ps
alias tm trace
alias v version
important for debugging
setenv PVM_EXPORT DISPLAY
want to see these trace events by default
tm addhosts delhosts halt
tm pvm_mytid pvm_exit pvm_parent

tm send recv nrecv probe mcast trecv sendsig recvf
inscripción de los nodos que forman parte del cluster
add pc1
add pc2
version # print PVM release version
id # print console TID
conf

Seguidamente se modificará el fichero *.rhosts* incluyendo el nombre de los nodos que van a trabajar con la PVM.

pc0 =¿front end
pc1
pc2
Antes de compilar se tendrá que comprobar que la PVM está activa de la siguiente forma: $pvm Una vez activada la PVM se utilizará el comando quit para salir de esta.
Seguidamente se creará un archivo llamado *Makefile.aimk* , que tendrá la siguiente estructura:

```
DEBUG =
SDIR = ..
BDIR = $(HOME)/pvm3/bin
#BDIR = $(SDIR)/../bin
XDIR = $(BDIR)/$(PVM_ARCH)
CC = gcc
OPTIONS = -g
CFLAGS= $(OPTIONS) -I$(PVM_ROOT)/include $(ARCHCFLAGS)

LIBS = -lpvm3 $(ARCHLIB)
GLIBS = -lgpvm3
LFLAGS= $(LOPT) -L$(PVM_ROOT)/lib/$(PVM_ARCH)
```

default: nombre_programa -master nombre_programa-slave

nombre_programa-master : $(SDIR)/ejer5-master.c $(XDIR)newli $(CC) $(DEBUG)
$(CFLAGS) -o $ $(SDIR)/ejer5-master.c $(LFLAGS) $(LIBS) -lm
cp $ $(XDIR)

nombre_programa-slave : $(SDIR)/nombre_programa-slave.c $(XDIR)
$(CC) $(DEBUG) $(CFLAGS) -o $ $(SDIR)/nombre_programa-slave.c $(LFLAGS)
$(LIBS) -lm
cp $ $(XDIR)
$(XDIR): - mkdir $(BDIR) - mkdir $(XDIR)
clean: rm -f *.o nombre_programa-master nombre_programa-slave $(XDIR)/nombre_programa-master $(XDIR)/ nombre_programa -slave

Para compilar los programas fuentes únicamente se tendrá que hacer:
$¿aimk
En el caso de que se quiera borrar los código objeto:
$¿aimk clean
Una vez que tenemos los programas ya compilados para ejecutarlos se realizará lo siguiente:

$¿programa-master Numero de procesos

XPVM

En muchas ocasiones, es muy útil tener una representación gráfica de la configuración de la máquina virtual que se está utilizando, así como una codificación visual de la actividad llevada a cabo en cada host de la máquina virtual, qué mensajes se están enviando, quién los envía y a dónde. La interfaz gráfica de usuario de PVM (XPVM) permite realizar todas estas funciones. XPVM (A Graphical Console and

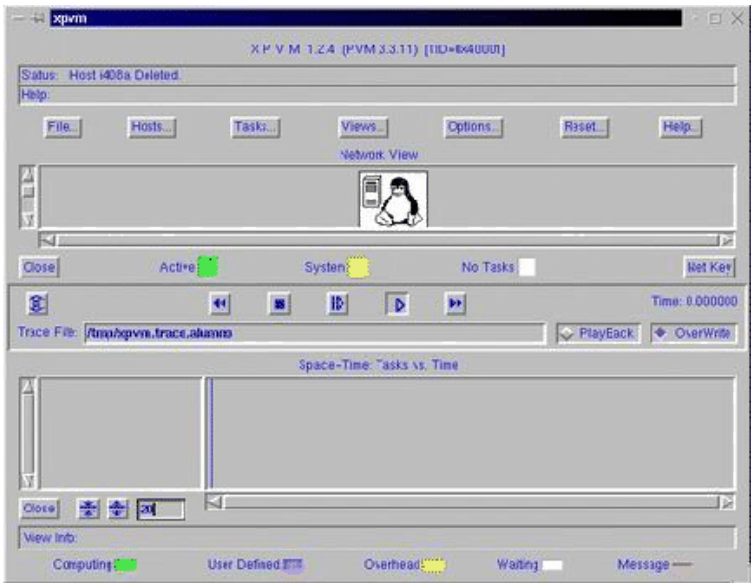

Figura 4.1: Apariencia XPVM.

Monitor for PVM) combina las funciones de la consola básica PVM con un monitor de seguimiento de actividades y un debugger en una interfaz tipo X-Windows. XPVM está escrito en C, usando el toolkit TCL/TK.

Para ejecutar XPVM, hay que asegurarse de que el daemon no está ya corriendo y que no haya archivos temporales relacionados con PVM.

La consola se compone de varias vistas de tamaño reconfigurable y una serie de ventanas que son utilizados por XPVM para mostrar mensajes de estado o de ayuda (Status y Help). Por defecto, la consola inicialmente muestra la vista de red (Network View) y la vista de representación temporal de tareas (Space-Time).

El menú Hosts permite añadir un nuevo host a la máquina virtual, seleccionado de entre todos los hosts listados en el archivo *.xpvm-hosts*.

Se pueden añadir varios hosts. Cada vez que se añade uno aparece un nuevo símbolo de hosts conectado a los símbolos existentes figura 4.2.

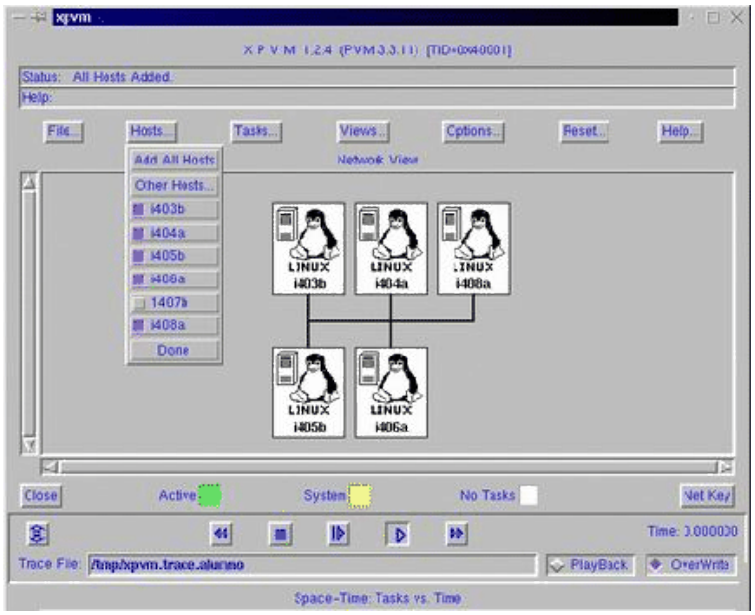

Figura 4.2: Conexión entre nodos.

A través del menú Tasks-SPAWN pueden lanzarse tareas en cualquiera de los hosts que compone la máquina.

La lista de Representación de Tareas muestra el estado de todas las tareas que se están ejecutando en la máquina virtual en un momento dado. Para que las tareas se muestren, el botón PLAY que se ve en la parte superior de la ventana de tareas figura 4.1. La visualización puede ser interrumpida o terminada en cualquier momento, utilizando los botones PAUSE y STOP. Una vez detenida la visualización se puede mover hacia atrás o adelante de las tareas utilizando los botones REWIND y FORWARD.

La vista de Representación de Tareas se compone de dos ventanas figura 4.3. Una ventana contiene el nombre del hosts y el de la tarea ejecutada en el mismo. Las tareas aparecen ordenadas alfabéticamente. El número de tareas mostradas en una ventana puede aumentarse utilizando el botón de compresión de tareas.

La otra ventana muestra, para cada proceso, el estado de dicha tarea en cada momento.

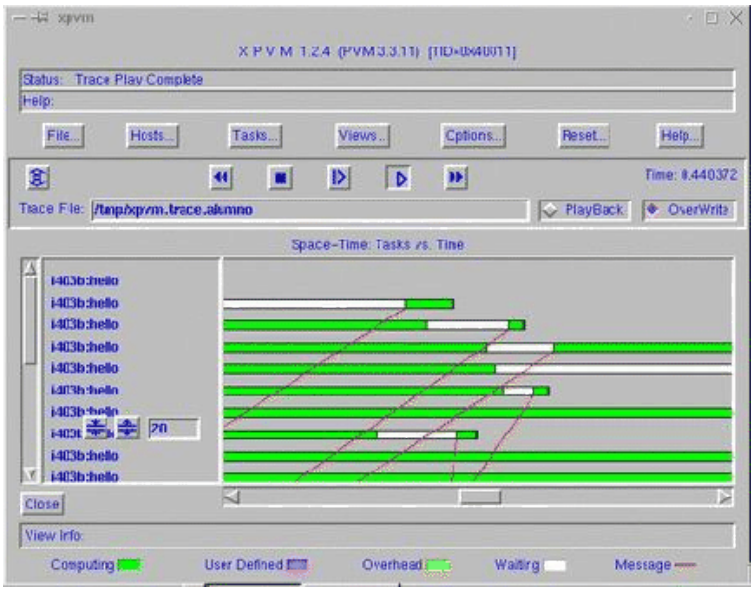

Figura 4.3: Representación de las tareas.

El usuario puede recabar información detallada sobre un estado determinado o un mensaje, seleccionando con el botón izquierdo un estado o mensaje. Si se selecciona una barra de tarea, se obtiene su estado así como el tiempo de comienzo y fin de la tarea y la última llamada a PVM que se hubiera generado. Si se selecciona una línea de mensaje, la ventana que aparece mostrará el tiempo de envío y recepción, así como el número de bytes enviados y el identificador de mensaje.

Existe una ventana de salida de tareas, accesible a través del menú VIEWS, que actúa como salida estándar para los procesos.

Finalmente, existe una ventana de utilización de recursos, accesible también a través del menú VIEWS. Esta ventana, que está sincronizada con la ventana de representación de tareas, muestra el número de tareas que están ejecutándose, ejecutando funciones PVM o en espera en cada momento.

4.2.2. MPI Message Passing Interface

MPI (Interfaz de paso de mensajes) es un estándar creado por un amplio comité de expertos y usuarios con el objetivo de definir una infraestructura común y una semántica específica de interfaz de comunicación. De esta forma los productores de software de procesamiento paralelo pueden implementar su propia versión de MPI siguiendo las especificaciones de este estándar, con lo cual múltiples implementaciones de MPI cambiarían solamente en factores tales como la eficiencia de su implementación y herramientas de desarrollos, pero no en sus principios básicos.

El paso de mensajes es una tarea ampliamente usada en ciertas clases de máquinas paralelas, especialmente aquellas que cuentan con memoria distribuida. Aunque existen muchas variaciones. En los últimos 10 años, se ha logrado un proceso substancial en convertir aplicaciones significativas hacia este tipo de tareas. Más recientemente diferentes sistemas han demostrado que un sistema de paso de mensajes puede ser implementado eficientemente y con un alto grado de portabilidad[29].

La meta de MPI, es el desarrollar un estándar para escribir programas que implementen el paso de mensajes. Por lo cual la interfaz intenta establecer para esto un estándar práctico, portable, eficiente y flexible.

El esfuerzo para utilizar MPI involucró a cerca de 60 personas de 40 organizaciones diferentes principalmente de U.S.A y Europa. La mayoría de los vendedores de computadoras concurrentes estaban involucradas con MPI, así como con investigadores de diferentes universidades, laboratorios del gobierno e industrias. El proceso de estandarización comenzó en el taller de estándares para el paso de mensajes en un ambiente con memoria distribuida, patrocinado por el Centro de Investigación en Computación Paralela en Williamsbur Virginia (Abril 29-30 de 1992). Se llegó a una propuesta preliminar conocida como MPI1, enfocada principalmente en comunicaciones punto a punto sin incluir rutinas para comunicación colectiva y no presentaba tareas seguras. El estándar final por MPI fue presentado en la conferencia de Supercomputación.

En un ambiente de comunicación con memoria distribuida en el cual las rutinas de paso de mensajes de nivel bajo, los beneficios de la estandarización son muy notorios. La principal ventaja al establecer un estándar para el paso de mensajes es la portabilidad y el ser fácil de utilizar.

En el modelo de programación MPI, un cómputo se comprende de uno o más procesos comunicados a través de llamadas a rutinas de librerías para mandar (send) y recibir (receive) mensajes a otros procesos. En la mayoría de las implementaciones de MPI, se crea un conjunto fijo de procesos al inicializar el programa, y un proceso es creado por cada tarea. Sin embargo, estos procesos pueden ejecutar diferentes programas. De ahí que, el modelo de programación MPI es algunas veces referido como MPMD (multiple program multiple data) para distinguirlo del modelo SPMD, en el cual cada procesador ejecuta el mismo programa.

Debido a que el número de procesos en un cómputo de MPI es normalmente fijo, se puede enfatizar en el uso de los mecanismos para comunicar datos entre procesos. Los procesos pueden utilizar operaciones de comunicación punto a punto para mandar mensajes de un proceso a otro, estas operaciones pueden ser usadas para implementar comunicaciones locales y no estructuradas. Un grupo de procesos puede llamar colectivamente operaciones de comunicación para realizar tareas globales tales como *broadcast*, etc. La habilidad de MPI para probar mensajes da como resultado el soportar comunicaciones asíncronas. Probablemente una de las características más importantes de MPI es el soporte para la programación modular. Un mecanismo llamado comunicador permite al programador de MPI definir módulos que encapsulan estructuras internas de comunicación (estos módulos pueden ser combinados secuencialmente y paralelamente).

El paso de mensajes en módulos de programación son por defecto no determinísticos; el orden de llegadas de los mensajes enviados desde dos procesos A y B hacia un tercer C, no está definido. Pero, MPI garantiza que dos mensajes enviados desde

un proceso A, hacia otro proceso B, llegarán en el orden en que fueron enviados.

En el modelo de programación Tarea/Canal, el determinismo es garantizado al definir canales separados para diferentes comunicaciones y al asegurar que cada canal tiene un sólo escritor y un sólo lector. Por lo cual, un proceso C puede distinguir mensajes recibidos de A o B tal y como llegan en diferentes canales. MPI no soporta canales directos, pero provee mecanismos similares; en particular, permite una operación de recibimiento para especificar una fuente, tag y/o contexto.

Los algoritmos paralelos ejecutan llamadas a operaciones para coordinar la comunicación en múltiples procesos. Por ejemplo, todos los procesos pueden necesitar cooperar para intervenir una matriz distribuida o para sumar un conjunto de números distribuidos en cada proceso. Estas operaciones globales pueden ser implementadas por un programador usando las funciones *send* y *receive*. Por conveniencia y para permitir la optimización, MPI provee un conjunto especializado de funciones colectivas de comunicación que obtienen operaciones de este tipo.

Una implementación alternativa con MPI, es el distribuir las estructuras de datos compartidas entre los procesos existentes, los cuales deben solicitar periódicamente las solicitudes pendientes de lectura y escritura. Para esto MPI presenta tres funciones **MPI-IPROBE, MPI-PROBE, MPI-GET-COUNT**.

- **MPI_IPROBE**: revisa existencia de mensajes pendientes sin recibirlos, permitiendo escribir programas que generan cómputos locales con el procesamiento de mensajes sin previo aviso.
- **MPI_PROBE**: es utilizado para recibir mensajes de los cuales se tiene información incompleta.
- **MPI_GET_COUNT**: se utiliza para conocer el tamaño del mensaje.

El mensaje puede ser recibido usando MPI_RECV. Primero detecta la llegada del mensaje utilizando MPI_PROBE. Después, determina la fuente del mensaje y utiliza

MPI_GET_COUNT para conocer el tamaño del mensaje. Finalmente, direcciona un buffer para recibir el mensaje.

MPI soporta la programación modular a través de su mecanismo de comunicador (comm, el cual provee la información oculta necesaria al construir un programa modular), al permitir la especificación de componentes de un programa, los cuales encapsulan las operaciones internas de comunicación y proveen un espacio para el nombre local de los procesos.

Una operación de comunicación MPI siempre especifica un comunicador. Éste identifica el grupo de procesos que están comprometidos en el proceso de comunicación y el contexto en el cual la comunicación ocurre. El grupo de procesos permite a un subconjunto de procesos el comunicarse entre ellos mismos usando identificadores locales de proceso y el ejecutar operaciones de comunicación colectivas sin meter a otros procesos. El contexto forma parte del paquete asociado con el mensaje. Una operación *receive* puede recibir un mensaje sólo si éste fue enviado en el mismo contexto. Si dos rutinas usan diferentes contextos para su comunicación interna, no puede existir peligro alguno en confundir sus comunicaciones.

MPI es un estándar creado por un amplio comité de expertos y usuarios con el objetivo de definir una infraestructura común y una semántica específica de interfaz de comunicación. De esta forma los productores de software de procesamiento paralelo pueden implementar su propia versión de MPI siguiendo las especificaciones de este estándar, con lo cual múltiples implementaciones de MPI cambiarían solamente en factores tales como la eficiencia de su implementación y herramientas de desarrollos, pero no en sus principios básicos.

Existen actualmente diferentes implementaciones de MPI, algunas son comerciales y otras son gratis y portables a múltiples arquitecturas. En el cluster se trabajará con **LAM** (Local Area Multicomputer) que es un ambiente de desarrollo y programación de MPI orientado a arquitecturas heterogéneas de computadoras dis-

tribuidas en una red local. Con LAM tanto un cluster dedicado como un una simple red local de computadoras puede actuar como una computadora multiprocesador y **MPICH** (MPI/Chameleon) producida por el laboratorio Nacional de Argonne y la Universidad del Estado de Mississippi.

Características principales de LAM:

- Implemetación completa del estándar MPI-1
- Contiene además muchas de las especificaciones de MPI-2 (MPI client/server, dynamic process spawning, one-sided communication, C++ bindings, MPI I-O, etc.)
- Soporta redes con estaciones de trabajo heterogéneas.
- Contiene herramientas para el debugging de procesos y mensajes.
- Nodos LAM dinámicos.
- Tolerancia a fallo.
- Comunicación rápida cliente-a-cliente.
- Está disponible gratuitamente.

LAM es portable a la mayoría de las máquinas UNIX : SUN (SunOS and Solaris), SGI IRIX, IBM AIX, DEC OSF/1, HP-UX, y Linux.

Para ejecutar aplicaciones paralelas en la máquina local o remota, el ejecutable de dicha aplicación puede estar almacenado en cualquier directorio de la máquina desde donde se ejecutan los programas maestros. Una manera sencilla de trabajar con *MPI* es crear estos mismos directorios en las máquinas remotas, de manera que no sea necesario decirle a MPI el directorio donde debe buscar las aplicaciones esclavas en las máquinas remotas, pues asume que es el mismo que utiliza la máquina

maestra. Adicionalmente *MPI* permite también transmitir por red el programa ejecutable a las máquinas remotas, la ventaja de hacer esto es que no es estrictamente necesario tener el programa compilado en todas las máquinas, aunque tiene un costo de tiempo de transmisión del mismo por la red[30].

Algunos de los comandos más importantes son:

- **lambot**: Inicializa los demonios de *LAM* en los nodos especificados.
- **mpirun**: Ejecuta el programa paralelo compilado en todos los nodos.
- **mpitask**: Brinda información de los procesos origen y destino, los identificadores de mensaje, tipos de datos transmitidos en los mensajes y las funciones invocadas.
- **lamclean**: Elimina todos los procesos y mensajes de *MPI*, sin abortar *MPI*. Es recomendable que este comando siempre se utilice después de terminar la sesión de trabajo con MPI.
- **wipe**: Elimina todos los procesos y demonios de *LAM* en las máquinas que se están utilizando. Esto solo debe ejecutarse cuando se termine de trabajar con *MPI*.

Instalación

- **Fuente**: http://www.mpi.nd.edu/lam
- **Version**:lam 6.3.1-4

Existen dos formas de instalar *LAM*: El paquete rpm de *LAM/MPI* que se va a instalar es el que provee el cd de Red Hat Linux, para llevar a cabo su instalación se realizará lo siguiente:
$¿mount /mnt/cdrom

$¿cd /mnt/cdrom/RedHat/RPMS/
$¿rpm -ivh lam-6.2.1-4.i386.rpm

Configuración

En el directorio **/usr/boot** se crean los siguiente archivos con sus correspondientes configuraciones.

- Archivo *conf.lam* , se recogerá la topología de red utilizada.
- Archivo *bhost.def* , se añadirá al final los nombres de todos los nodos que forman parte del cluster. localhost nodo10 nodo12 ...
- Para saber si la configuración es correcta se deberá de realizar lo siguiente:
  $¿lamboot

- Si la salida muestra algún error entonces es que hay algún problema con la instalación, comunicación entre nodos, etc.

Compilación y ejecución de programas

Se genera el siguiente fichero llamado **makefile** , que tendrá la siguiente estructura:

.SILENT:
CFLAGS=-I/usr/include/lam -L/usr/lib/lam
CC=mpicc
nombre_programa : nombre_programa.c $(CC) $(CFLAGS) nombre_programa.c -o nombreprograma

Para compilar los programas LAM/MPI se utilizará el siguiente comando:

$¿make -f makefile

Una vez compilado el programa y antes de ejecutarlo se necesita primero arrancar

el sistema Multicomputadora de Area Local, esto se hace a través del comando lamboot.

Para ejecutar el programa se utiliza el siguiente comando:

$¿mpirun -np n_de_procesos nombre_programa argumentos

Características principales de MPICH

La principal característica de la librería *MPICH* es su adaptabilidad en gran número de plataformas, incluyendo clusters de workstations y procesadores masivamente paralelos (MPP).

Las características más importantes de esta librería son:

- Soporta paralelismo del tipo SPMD (Single Program Multiple Data) y MPMD (Multiple Program Multiple Data)
- La transferencia de mensaje se realiza de forma cooperativa, tanto el proceso emisor como el receptor participan en el traspaso.
- Permite establecer comunicación punto a punto (participan exactamente dos procesos) o comunicación colectiva (participan un número elevado de procesos simultaneamente).
- Proporciona cuatro modos de comunicación: *standar, synchronous, buffered y ready.*

Instalación y configuración

- **Fuente**: http://www-unix.mcs.anl.gov/mpi/mpich
- **Version**: mpich 1.2.5

Una vez que se obtenga la ultima version de *mpich* continuan los siguientes pasos:

- Se desempaca el archivo de instalación de *mpich* en el directorio compartido
- Se va al directorio **mpich** y se teclea *./configure -arch=LINUX device=ch_p4 -rsh=ssh*
- Se teclea *make*
- Se teclea *util/tstmachines*
- Se va al directorio **mpich/util/machines**
- Se edita *machines.LINUX* y se borran todas las líneas del servidor y luego se teclean los nombres de todos los nodos
- Se va a **mpich/util**
- Se teclea *./tstmachines*

Compilación y ejecución de programas

Para compilar los programas con *MPICH*, se teclea lo siguiente en la línea de comandos:

$mpicc -lm -o nombre_binario nombre_fuente.c

Los tipos de ejecución del binario se describe a continuación:
$mpirun na-b nombre_binario parametros_del_binario
$mpirun N nombre_binario parametros_del_binario
$mpirun -c NUM nombre_binario parametros_del_binario
donde,
n indica que se va a utilizar un intervalo de nodos.
a indica el primer nodo.
b indica el último nodo.
N indica quese utilizarán todos los nodos.
c indica que se ejecutará copias del programa.
NUM indica el numero de copias del programa que se van a ejecutar.

nombre_binario es el archivo ejecutable en cuestión.

parametros_del_binario son los parámetros del programa.

Capítulo 5
Algunos aspectos de seguridad

Hay algunos pasos que se pueden tomar para hacer el sistema un poco más resistente a los ataques. Incluso si, varios defectos fundamentales en el modelo UNIX aseguran que nunca se alcanzará el máximo de seguridad:

- UNIX está optimizado por conveniencia y no hace la seguridad fácil y natural. UNIX fue diseñado por investigadores, para investigadores y, su filisofía enfatiza la fácil manipulación de la información en un ambiente multiusuario en una red.

- La seguridad de UNIX es efectivamente binaria: existe un usuario sin privilegios y un usuario llamado root. Las facilidades de UNIX tales como la ejecución de *setuid* tienden a conferir todo un poder total una sóla vez. Lapsos insuficientes en seguridad pueden comprometer a sistemas completos.

- La mayoría de las funciones administrativas están implemetadas fuera del kernel, donde pueden ser inspeccionadas y alteradas. Los *hackers* plenamente pueden acceder al sistema.

Podría parecer que la seguridad de UNIX debería mejorar gradualmente al mismo tiempo como los problemas de seguridad son descubiertos y corregidos, pero desafortunadamente éste no parece ser el caso. El software del sistema está creciendo siempre más complicado, los *hackers* están llegando a ser mucho mejores y mucho mejor organizados y, las computadoras están conectadas más y más intimamente en Internet.
La seguridad está en una batalla constante que realmente nunca puede ser ganada.
Recordar esto muy bien:

$$Seguridad = \frac{1}{(1,072)\,(Conveniencia)}$$

A más seguridad del sistema, más miserablemente el administrador y los usuarios tenderán a ser.

5.1. Siete reglas de sentido común de seguridad

La seguridad efectiva de un sistema tiene sus raíces en el sentido común:

- No poner archivos en el sistema que probablemnte sean de interés para los *hackers* o para usuarios molestos. Documentos secretos, archivos personales, programas secretos, etc., deben ser guardados cuidadosamente si éstos están en línea. Seguridad como información criptográfica proveerá un grado más alto de seguridad, que simplemente tratar de prevenirse de usuarios sin autorización que quieran tener acceso a estos archivos.
- Cerrar agujeros que los *hacker* puedan usar para ganar acceso al sistema. Monitorear los boletines de seguridad del vendedor y las listas de correo de seguridad. Parar servicios innecesarios.
- No proveer lugares para que los *hackers* construyan sus nidos en el sistema. Los hacker a veces entran en un sistema y después lo usan como una base de operaciones para conseguir otros sistemas. Los directorios de FTP anónimos de escritura para todos, cuentas de grupos, y cuentas con una contraseña fácil de adivinar, todos éstos fomentan actividad de anidamiento.
- Poner trampas para detectar intrusiones o intentos de intrusiones. Herramientas tales como *tripwire*, *tcpd* y *crack* ayudarán a mantener el sitema nivelado de problemas potenciales.
- Monitorear los reportes generados por estas herramientas de seguridad. Un problema menor que es ignorado en un reporte, puede llegar a ser una catastrofe en el tiempo que el siguiente reporte es mandado.
- El administrador debe aprender por sí mismo sobre la seguridad de UNIX.

Los consultores sobre seguridad cobran muy caro[39]. Desafortunadamente, sus soluciones no siempre resuelven del todo el problema.

- Inspeccionar para encontrar actividades inusuales. Investigar todo lo que parezca inusual, tal como mensajes *logs* antiguos o cambios en la actividad de una cuenta (más actividad, actividad en horas inusuales o, tal vez actividad mientras el propietario está de vacaciones).

5.2. Agunos problemas de seguridad

5.2.1. TFTPD

tftpd es un servidor para el Protocolo de Transferencia de Archivos Trivial (Trivial File Transfer Protocol), un protocolo fácil de implantar que es a veces usado para bajar *fireware* o código boot en dispositivos de red. Porque esto permite a la máquinas sobre la red pedir archivos del disco duro, esto es un agujero potencial de seguridad. Es mejor dejarlo deshabilitado si no se está usando.

En este trabajo de investigación es obligatorio usar el demonio tftpd, sin embargo, como se utiliza sobre una red interna, el problema de seguridad es nulo.

5.2.2. Seguridad y NIS

Estas palabras nunca deberían ser usadas juntas. Como ya se vio en el capítulo anterior, NIS es una herramienta de distribución de base de datos de Sun, que muchos sistemas usan para mantener y distribuir archivos tales como */etc/group*, */etc/passwd* y */etc/hosts*. Desafortunadamente, su muy natural "acceso fácil a la información" la hace tentadora a los *hackers*. Un reemplazo más actual de NIS es llamado NIS+, éste hace un intento ineficaz para dirigir los problemas de seguridad de NIS. Se recomienda no correr ninguno para mayor seguridad.

Revisar la siguiente tabla:

*El Cluster **SCLUSTER** utiliza NIS, pero como ya se mencionó, se utiliza en una red interna, por lo tanto no hay problemas de seguridad.*

[39] Aproximadamente por 250 dolares, ellos pueden hacer el sistema seguro

5.2.3. Seguridad y NFS

NFS provee una manera conveniente de acceder archivos en una red y, por lo tanto esto tiene un gran potencial de causar problemas de seguridad.

El Protocolo *NFS* originalemnte fue diseñado con aspectos muy simples de seguridad. Más tarde el protocolo fundamental de *RCP* fue revisado para permitir el uso de módulos de autenticación conectables. Sin embargo, la nueva especificación se detuvo y se recomendó algún mecanismo de seguridad particular.

SUN, que le da un énfasis enorme a programas basados en una llave pública que otros vendedores generalmente ignoran y, *Kerberos* el cual extiende su autenticación estándar a *RCP*. Los dos esquemas ayudan a asegurar qué usuarios remotos realmente son quienes dicen ser. Sin embargo, actualmente ninguno de los dos encriptan la información en la red, así que, todavía se está a merced de cualquiera que corra un husmeador de paquetes *(packet sniffer)*.

Si el sistema ya está corriendo un sistema de llave pública de Sun o *Kerberos*, usarlos con *NFS*. Si no es así, *NFS* no proveerá mucha seguridad. Aunque en un ataque a la seguridad de un sistema, estos dos esquemas pueden ser dignos de investigar, en realidad ambos sistemas o esquemas que sólo ofrecen un incremento mínimo de seguridad comparado con un buen sentido común.

Si el sistema ha instalado un *firewall*, es buena idea bloquear el acceso a los puertos 2049 de TCP y UDP, los cuales son usados por NFS[41]. Se debería también bloquear el acceso al demonio **portmarp** SunRPC, el cual normalmente escucha en los puertos 111 de TCP y UDP. Es implícito en estas precauciones, pero, es mejor decir que el sistema de archivos NFS no debería ser exportado a máquinas que no son locales (excepto WebNFS).

[41]Se tendrá que desbloquear el puerto 2049 TCP, si el sistema provee el servicio *WebNFS*, sin embargo, no desbloquear este puerto para alguna máquina que contenga información sensitiva.

Sistema	**Soporta NIS**	**Soporta NIS+**
Solaris	Si	Si
HP-UX	Si	Si
Red Hat[40]	Si	No
FreeBSD	Si	No

Cuadro 5.1: Tabla de soporte de NIS y NIS+

5.3. Seguridad en cuentas de usuarios

Sumar y remover usuarios es un trabajo rutinario en la mayoría de los sistemas. Una cuenta bien configurada es una clave determinante de *seguridad del sistema*. Cuentas que no se usan frecuentemente son el primer objetivo para los *hackers*, como también lo son las cuentas con contraseña fácil de imaginar. Incluso si se usa la herramienta por default del sistema para sumar y remover usuarios, es importante comprender los cambios importantes que la herramienta está haciendo.

5.3.1. El archivo /etc/passwd

El archivo */etc/passwd* es una lista de usuarios reconocidos por el sistema. El sistema consulta el archivo cuando alguien accede a éste, para determinar el UID de usuario y para verificar su contraseña. Cada línea en el archivo representa un usuario y contiene siete campos separados por dos puntos:

- Nombre de acceso.
- Contraseña encriptada (A menos de que un archivo de contraseña sombra "shadow password"sea usado).
- Número de UID.
- Número GID por defecto.
- Información "GECOS": Nombre completo, oficina, etc.
- Directorio hogar (home).
- shell de acceso.

Un ejemplo:

root:jsg8Y.1p6uWMo:0:0The System,,x6096,:/:/bin/csh
laura:oy7tes4wq6j7yrt.ljhi:101:20::/home/laura:/bin/csh

Los contenidos de /etc/passwd son a veces compartidos entre sistemas con sistema de base de datos tales como *NIS o NIS+*.

En este trabajo de investigación se utilizó el sistema de base de datos NIS.

5.3.1.1. Nombre de acceso

Los nombres de acceso son caso sensitivo, sin embargo, la mayoría de los sistemas de correo (incluyendo *sendmail*) esperan nombres de acceso con minúsculas. Por esta razón, se sugiere evitar caracteres con mayúsculas, a menos que el usuario no esté esperando recibir algún correo. Los nombres con minúsculas son tradicionales y también son fáciles de teclear.

Si se tiene más de una máquina, los nombres de acceso deberían ser únicos en dos sentidos.
Primero, un usuario debería tener el mismo nombre de acceso en cada máquina. Esta regla es principalmente por conveniencia, para ambos, el administrador y el usuario.

Segundo, un nombre de acceso particular siempre debería referirse a la misma persona. Algunos comandos Unix (ejemplo, *rlogin* and *ssh*) pueden ser configurados para validar usuarios remotos basados en sus nombres de acceso. Incluso, si laura@scluster y laura@gulfi son dos personas diferentes, uno podría acceder dentro de la cuenta del otro sin proveer una contraseña, si el sistema no estuviera propiamente configurado.

5.3.1.2. Contraseña encriptada

/etc/passwd guarda contraseñas en una forma encriptada. A menos de que alguien pueda ejecutar encriptación DES en la cabeza, se debe colocar el contenido de este campo usando el comando **passwd** (**yppaswd** si se usa NIS) o copiando la cadena de la contraseña encriptada de otra cuenta [42]

[42]La mayoría, pero no todos, los sistemas usan DES para contraseñas encriptadas. Se puede sólo copiar las contraseñas encriptadas entre máquinas que usan el mismo algoritmo de encriptación.

Cuando se edita */etc/passwd* para crear una nueva cuenta, poner un asterisco (*) en el campo de la contraseña encriptada. El asterisco previene uso no autorizado de la cuenta hasta que se ha configurado una contaseña real.
Nunca se deje este campo vacío, esto introduce un agujero gigantesco de seguridad porque ninguna contraseña es requerida para acceder a la cuenta.

Red Hat Linux y *FreeBSD* incluyen soporte para contraseñas basadas en *MD5*, las cuales también pueden ser de alguna longitud. Las contraseñas encriptadas *MD5* son fáciles de notar porque son de 31 caracteres de longitud y los 3 primeros caracteres son siempre "1".

El hardware en cómputo ha llegado a ser más rápido y ha llegado a ser incrementalmente peligroso dejar contraseñas encriptadas en vista plana. En la actualidad, la mayoría de los sistemas permiten ocultar las contraseñas encriptadas colocándolas en un archivo separado (*/etc/shadow*) que no puede ser leído por los usuarios, excepto el superusuario (root). Éste es conocido como un mecanismo de contraseña sombra *(shadow password)*.

5.3.1.3. Número UID

Por definición, *root* tiene UID 0. La mayoría de los sistemas también definen seudo-usuarios *bin*, demonio y tal vez algunos otros. Es usual poner estas seudo-cuentas al inicio del archivo */etc/passwd* y darles un número de UID pequeño. Para permitir tener suficientes números para asignarles a estas seudo-cuentas que se podría querer sumar en el futuro, se recomienda asignar UIDs a usuarios reales comenzando con el 100.

Evitar reciclar UIDs tanto como sea posible, incluso los UIDs de gente que ha dejado la organización y tuvieron sus cuentas permanentemente removidas (en esta investigación de las personas que dejen el servico social o la tesis). Esta precaución previene confusión si los archivos son más tarde restaurados de un respaldo, en los cuales los usuarios son identificados por UID en lugar de su nombre de acceso.

5.3.1.4. Número GID por defecto

El GID 0 está reservado para el grupo llamado 'root' o 'whell'. El GID 1 es usualmente el grupo "demonio".

Los grupos son definidos en el archivo */etc/group*, con el campo GID en */etc/passwd* provee el GID efectivo durante el acceso. Versiones modernas de UNIX permiten a usuarios estar en más de 16 grupos a la vez, así que el GID efectivo nunca es usado para determinar el acceso. El campo GID en */etc/passwd* es por lo tanto algo de una herencia histórica, aunque su valor es todavía incluido en la lista de grupo del usuario.

5.3.1.5. Campo GECOS

El campo GECOS[43] es comunmente usado para registrar información personal de algun usuario.

El comando **chfn** (**passwd -g en Solaris**) le permite al usuario cambiar su propia información GECOS. **chfn** es útil para guardar datos como números telefónicos, pero este puede ser usado erroneamente: Un usuario puede cambiar la información que sea ambas, incorrecta y obscena. Algunas instituciones tienen deshabilitado el comando **chfn**.

5.3.1.6. Directorio hogar (home)

Los usuarios son colocados en su directorio *home* cuando acceden al sistema. Si un directorio *home* del usuario es omitido al acceder, el sistema imprime un mensaje tal como "ningun directorio *home*". Algunos sistemas permiten el acceso para proceder y poner al usuario en el directorio root. Otros sistemas no permiten acceder sin un directorio *home* válido.

Ser consciente que si los directorios *home* son montados sobre *NFS*, ellos pueden no estar disponibles en el caso del servidor o problemas de red.

[43]Cuando Honeywewll tomó el mando de la división de cómputo de GE, GECOS fue cambiada a GCOS; ambas abreviaturas sobreviven hoy.

5.3.1.7. Shell de acceso

El shell de acceso es normalmente un intérprete de comandos tal como Bourne shell o el C shell (*/bin/sh* o */bincsh*), pero puede estar en algunos programas. **sh** es el shell por omisión en la mayoría de los sistemas y, es usado si */etc/passwd* no especifica un shell de acceso. Otros shells comunes incluyen *ksh* (the Korn shell), *bash* (the GNU Bourne again shell), y *tcsh* (un supuesto C shell con edición de comandos).

En la mayoría de los sistemas, los usuarios pueden cambiar sus shells con el comando **chsh**. El archivo */etc/shells* contiene una lista de los shells que **chsh** permitirá que los usuarios seleccionen el más conveniente a sus necesidades, root puede usar **chsh** sin restricciones. Si se suman entradas a */etc/shells*, asegurarse de usar rutas absolutas ya que **chsh** y otros programas las esperan.

5.3.2. El archivo /etc/group

El archivo */etc/group* contiene los nombres de los grupos de UNIX y una lista de cada miembro del grupo. Por ejemplo:

root:x:0:root
tesis03:x:1000:laura,martin,jeanette,roy

Cada línea representa un grupo y contiene cuatro campos:

- Nombe del grupo
- Contraseña encriptada
- Número de GID
- Lista de miembros, separados por comas

La mayoría de los sitios ponen un asterisco (o una 'x') en el campo de la contraseña, pero es seguro dejarlo en blanco si se desea. El comando **newgrp** no cambiará a una grupo sin una contraseña, a menos que el usuario ya esté listado como perteneciente a ese grupo.

Tener cuidado de no sumar espacios entre los nombres de los miembros del grupo. La mayoría de los sistemas ignoraran cualquier cosa después del primer espacio.

5.3.3. Sumar usuarios

Mecánicamente, el proceso de sumar un nuevo usuario consiste de tres pasos requeridos por el sistema, dos pasos que establecen un ambiente útil para el nuevo usuario y, varios pasos extras para el administrador.

Requerido:

- Editar los archivos *passwd* y *shadow* para definir la cuenta del usuario.
- Configurar una contraseña inicial.
- Crear el directorio hogar *(home)* del usuario.

Por el usuario:

- Copiar los archivos de inicio por default al directorio home del usuario.
- Configurar el home del correo del usuario y establecerle un alias.

Para el administrador:

- Sumar el usuario al archivo */etc/group*.
- Configurar cuotas del disco (disk quotas).
- Verificar que la cuenta esté correctamente configurada.

5.3.3.1. Editar los archvivos passwd y shadow

Para editar de forma segura el archivo *passwd*, correr **vipw** para invocar un editor de texto en una copia de éste. El editor por omisión es **vi**, pero se puede especificar uno diferente configurando el valor de la variable de ambiente *editor*. La existencia del archivo temporalmente editado sirve como una cerradura; **vipw** permite que sólo una persona edite el archivo *passwd* a la vez. Cuando el editor termina, **vi** reemplaza el archivo *passwd* original con la copia editada.

5.3.3.2. Configurar una contraseña inicial

Root puede cambiar alguna contraseña de los usuarios con el comando **passwd**:

passwd user

El programa **passwd** que viene con Red Hat checa contraseñas esperadas para asegurarse que no estén en el directorio del sistema. Esta precaución no es completamente como a través del chequeo ejecutado por **npasswd**[44], pero es útil.

Nunca dejar una cuenta nueva o una cuenta que tiene acceso a un shell, sin una contraseña.

5.3.3.3. Crear el directorio home del usuario

Algún directorio que se cree es inicialmente propiedad de root, así que se debe cambiar su propietario y su grupo con los comandos **shown** y **chgrp**. La siguiente secuencia de comados crearía un directorio home apropiado para un usuario llamado laura:

mkdir /home/staff/laura
chown laura /home/staff/laura
chgrp staff /home/staff/laura
chmod 700 /home/staff/laura

5.3.3.4. Copiar los archivos de inicio por default

Se puede personalizar algunos comandos y utilerías colocando los archivos de configuración en un directorio home del usuario. Los archivos de inicio tradicionalmente comienzan con un punto y terminan con las letras *rc*.

Si todavía no se tiene un conjunto de archivos de inicio por default, */usr/local/lib/skel* es un lugar razonable para ponerlos. Copiar algunos archivos para usar como punto

[44]Se recomienda actualizar el **passwd**. Se sugiere el **npasswd**, que puede ser encontrado en http://www.utexas.edu/cc/unix/software/npasswd

de partida y modificarlos con un editor de textos. Se podría desear comenzar con los archivos que vienen con el sistema del directorio */etc/skel*, si el sistema los provee.

Dependiendo del shell del usuario, */etc* puede contener archivos de inicio del sistema amplios que son procesados antes que los archivos de inicio del propio usuario. Por ejemplo, el Bourne shell (**sh**) lee el archivo */etc/profile* antes de procesar */.profile*.

La secuencia de comandos para instalar archivos de inicio para un nuevo usuario llamado laura podría verse como esto:

cp /usr/local/lib/skel/.[a-zA-Z]* /laura
chmod 644 /laura/.[a-zA-Z]*
chown laura /laura/.[a-zA-Z]*
chgrp staff /laura/.[a-zA-Z]*

Notar que se puede usar:

chown laura /laura/.*

porque laura reconocería entonces no sólo sus propios archivos, si no también los del directorio padre ".."(*/home/staff*).

Éste es un error del administrador del sistema (sysadmin) muy común y peligroso.

5.3.3.5. Configurar el home del correo del usuario

Es conveniente para cada usuario recibir correo únicamente en una máquina. Este esquema es a veces implantado con una entrada en el archivo de alias global */etc/mail/aliases* o en el *sendmail* userDB. En Red hat 9.0 el archivo de alias global es */etc/aliases*.

5.3.3.6. Editar el archivo /etc/group

Continuando con el nuevo usuario laura, se debería sumar el nombre de acceso a la lista de usuarios en el grupo 100, ya que éste fue el grupo por default que se le asignó en el archivo */etc/passwd*. Estrictamente hablando, laura estará en el grupo 100 sin importar que esté en lista o no en el archivo */etc/group*, porque su entrada *passwd* ya le ha dado su membresía. Sin embargo, esta información debería ser

inscrita en */etc/group* para que el administrador conozca exactamente qué usuario pertenece a que grupo.[45]

Suponer que se quiere poner a laura en el grupo "wheel". En la mayoría de los sistemas, los usuarios deben estar en este grupo para usar el comando **su**. Simplemente se harían los siguientes cambios a */etc/group*.

wheel:*:0:root,laura
csstaff::100:laura

5.3.3.7. Configurar cuotas del disco

Si el sitio usa cuotas del disco (disk quotas), se debería configurar límites de cuotas para cada una de las nuevas cuentas con el comando **edquota**. Éste puede ser usado interactivamente, pero es más comunmente usado en modo "prototipo"para el modelo de cuotas del nuevo usuario, después de que se usó en alguien más. Por ejemplo, el comando

#**edquota -p** proto-user new-user

configura la cuota de new-user para ser la misma a la de proto-user. Esta manera de usar **edquota** es especialmente útil en el script **adduser**.

A pesar de que el espacio en el disco es barato en la actualidad, no se está a favor de grandes cuotas de disco. Ellas a veces parecen causar más problemas de los que resuelven e imponen un soporte obligatorio adicional en los administradores.

5.3.3.8. Verificar el nuevo login

Para verificar que una nueva cuenta ha sido adecuadamente configurada, primero salir de la sesión actual, entonces acceder como el nuevo usuario y ejecutar los siguientes comandos:

[45]El kernel actualmente no se preocupa de lo que haya en */etc/passwd* o */etc/group*, éste sólo se preocupa por los números UID y GID crudos (raw). *passwd* y *group* guardan información de la cuenta del usuario por software de alto nivel tal como **login**.

% **pwd** /* Para verificar el directorio home */
% **ls -la** /* Checa propietario/grupo de los archivos de inicio */
Se necesitará notificar a los usuarios nuevos de su nombre de acceso (login) y su contraseña inicial. Es buen tiempo para sumar documetación adicional de los usuarios, si se dispone.
Asegurarse de recordar a los nuevos usuarios de cambiar su contraseña inmediatamente.

5.3.4. Remover usuarios

Este procedimiento involucra remover todas las referencias al nombre de acceso que fue sumado por el administrador o con el programa **adduser**. Se podría querer usar la siguiente lista de chequeo:

- Configurar la cuota de disco a cero del usuario, si la cuota está en uso.
- Remover el usuario de alguna base de datos de usuario local o listas telefónicas.
- Remover el usuario del archivo *aliases* o sumar una dirección de reenvío.
- Remover el archivo *crontab* del usuario y algunos trabajos pendientes.
- Matar todos los procesos del usuario que estén todavía corriendo.
- Remover algunos archivos temporales propiedad del usuario en */var/tmp* o */tmp*.
- Remover el usuario de los archivos *passwd* y *group*.
- Remover el directorio home del usuario.
- Remover el 'spool mail' del usuario.

Antes de remover el directorio *home* del usuario, asegurarse de relocalizar algunos archivos que son necesarios para otros usuarios. Ya que a veces no se puede estar seguro cuáles archivos de éstos podrían ser, siempre es una buena idea hacer

una cinta extra de respaldo del directorio *home* y del 'spool mail' del usuario antes de borrarlos.

Una vez que se removió el usuario, se debería verificar que el UID viejo del usuario no contiene más archivos de su propiedad en el sistema. Una manera rápida de ejecutar este chequeo es usar el comando **quot**. En Linux Red Hat 9.0 el comando es **quotacheck.**

5.3.5. Deshabitar nombres de acceso

En ocasiones el login del usuario debe ser deshabilitado. Antes, para no dejar acceder a un usuario a la red en el mundo UNIX, se pondría sólo un asterisco enseguida de la contraseña encriptada, hacer esto imposibilita el acceso al usuario. Sin embargo, los usuarios todavía pueden acceder a través de la red sin teclear una contraseña, así que esta técnica no trabaja muy bien.

Actualmente, se reemplaza el shell del usuario con un programa que imprime un mensaje explicando porqué el login ha sido deshabilitado y provee instrucciones para rectificar la situación. El pseudo-shell no debería estar listado en */etc/shells*; muchos demonios que proveen acceso al sistema sin ningún login (ejemplo, **ftpd**) checan para ver si el shell de acceso del usuario está listado en */etc/shells* y niega el acceso si éste no está (que es lo se requiere).

Este es un problema con este método de deshabilitar logins. Sin embargo, por default **sendmail** no deliverará correos a los usuario cuyo shell no se encuentre en */etc/shells*. Generalmente, es mala idea interferir con el flujo del correo, incluso si el recipiente no está posibilitado para leerlo inmediatamente. Se puede anular el comportamiento de default de **sendmail** sumando un shell simulado llamado *SENDMAIL/ANY/SHELL* a el archivo */etc/shells*.

5.3.6. Configuración de una cuenta para usar el cluster

Pasos esenciales para generar una cuenta:

Como root:

- Crear un grupo para el usuario, si el grupo ya existe, sumarlo al grupo elegido después de crear la cuenta. Ejemplo
  #**groupadd** -g 1001 tesis04

  donde -*g* es GID (1001) y el último parametro es grupo (tesis04).

- Elegir un nombre de acceso(login). Se recomienda usar la primer letra del nombre y el apellido. Ejemplo
  #**adduser** -g 1001 -c Lissette lgonzalez

  donde -g es UID, -c es GECOS y, el último parámetro es login (lgonzalez).

- Elegir una contraseña válida, es decir, que no esté dentro del diccionario del sistema, no sea fácil de imaginar, recordable; usar letras, números y caracteres alfanúmericos. Ejemplo
  #**passwd** lgonzalez
  Utilizar una contraseña como: mi35fd/$pw.

- Editar el archivo /etc/group. Sumar el usuario que se acaba de crear. Ejemplo
  #**vi** /etc/group.
  Y sumar a la línea donde se encuetra el grupo creado o seleccionado, el login del usuario.
  tesis04:x:1001:usuario1,usuario2,usuario3,...,lgonzalez

- Recompilar el Servidor NIS para actualizar la cuenta creada. Ejemplo
  Ir a directorio */var/yp*.
  Ejecutar el comando **make**.

Como usuario:

- En el home del usuario crear un archivo llamado */home/lgonzalez/.rhosts* con cualquier editor de texto (vi, emacs, etc, o con un simple cat) y agregar la siguiente línea: "servidor-de-acceso login". Ejemplo
  $**vi** .rhost (enter)
  Sumar la siguiente línea. "scluster lgonzalez"

5.3.7. Utilidades para la administración de cuentas

Red Hat provee un conjunto de utilidades para ayudar a automatizar la creación, borrado y modificación de usuarios y grupos.

- **useradd**. Comando para sumar usuarios. Ejemplos:

  # **useradd** laura. /* Sin parametros */
  # **useradd** -c "Laura Sandoval" -d /home/telematica/laura -g sinodal -G academico -m -s /bin/tcsh laura /* Con parametros */

- **adduser**. Comando para sumar usuarios.

- **usermod**. Comando para cambiar las entradas de *passwd* de usuarios existentes. Ejemplo:

  # **usermod** -e "diciembre 31, 2003"laura /* Fecha de expiración */

- **userdel**. Comando para borrar usuarios.

- **groupadd**. Comando para sumar un grupo.

- **groupmod**. Comando para modificar un grupo.

- **groupdel**. Comando para borrar un grupo.

Aunque estos comandos son convenientes, ellos son raramente suficientes para implantar todas las políticas de la organización o institución. Se recomienda escribir scripts propios para ***adduser*** *y* ***rmuser*** *(que pueden o no ser llamados como los del sistema). Perl es adecuado para esto.*

5.4. Herramientas de seguridad

5.4.1. nmap: Explora puertos de red

nmap es un explorador de puertos de red. Su principal función es checar un conjunto de *hosts* específicos para ver qué puertos de TCP y UDP tienen servicios escuchando en ellos[46]. Puesto que la mayoría de los servicios de red están asociados con números de puertos "conocidos", esta información le dice al administrador completamente todo sobre el software que la máquina está corriendo.

Correr **nmap** es una excelente forma de averiguar cómo se ve un sistema cuando alguien está tratando de entrar.

Ejemplo de una máquina insegura:

% **nmap** -sT host1.uexample.com
Starting nmap V. 2.12 by Fyodor (fyodor@dhp.com, www.insecure.org/nmap/)
interesting ports on host1.uexample.com (10.10.2.1):

[46]Un puerto es un canal de comunicación numerado. Una dirección IP identifica a una máquina completa y, una dirección IP + un número de puerto identifica a un servidor específico o a una conversación de red en esa máquina.

Port	State	Protocol	Service
7	open	tcp	echo
9	open	tcp	discard
13	open	tcp	daytime
19	open	tcp	chargen
21	open	tcp	ftp
23	open	tcp	telnet
25	open	tcp	smtp
...			echo
513	open	tcp	login

Nmap run completed – 1 IP address (1 host up) scanned in 1 second

El argumento **-sT** pregunta a **nmap** para tratar y conetar a cada uno de los puertos TCP sobre el host especificado en la forma normal[47]. Una vez que una conexión ha sido establecida, **nmap** inmediatamente se desconecta, lo cual es desagradable pero no es perjudicial para un servidor de red escrito apropiadamente.

Del ejemplo de arriba se puede ver que está corriendo varios servicios que han sido históricamente asociados con problemas de seguridad: **ftpd** (ftp), **rlogind** (login) y, probablemente **sendmail** (smtp). Se puede ver claramente varias líneas potenciales de ataque.

La columna *state* en la salida de **nmap** muestra "open" para puertos con servicios, "unfiltered" para puertos sin servicios y "filtered" para puertos que no pueden ser probados porque interviene un *firewall*. Puertos no filtrados (unfiltered) son el caso típico y normalmente no son mostrados a menos de que haya relativamente pocos de ellos.

Ejemplo de un servidor web comercial más seguro:

% **nmap** -sT www.aexample.com

[47] Actualmente, sólo los puertos privilegiados (estos con números de puerto abajo de 1,024) y los puertos bien conocidos son checados por default. Usar **-p** para especificar explícitamente qué rango de puertos explorar.

Starting nmap V. 2.12 by Fyodor (fyodor@dhp.com, www.insecure.org/nmap/)
(Not showing ports in state: filtered)

Port	State	Protocol	Service
53	unfiltered	tcp	domain
80	open	tcp	http
179	unfiltered	tcp	bgp
443	open	tcp	https

Nmap run completed – 1 IP address (1 host up) scanned in 122 seconds

En este caso, es claro que el servidor está configurado para manejar sólo tráfico de la web. Un firewall bloquea el acceso a otros puertos. El tráfico DNS y BGP está permitido a través de éste, pero ningún servidor está corriendo para recibirlo. Idealmente, el *firewall* en este sitio debería bloquear el tráfico para todos los servicios sin usar (tal como DNS y BGP, en este caso), para que estos puertos no puedan ser forzados para otros propósitos.

nmap tiene una habilidad útil y mágica para imaginar que sistema operativo (OS) un sistema remoto está corriendo, buscando en sus particularidades de su implantación de TCP/IP. Con la opción **-O** se activa este comportamiento. Ejemplo:

% **nmap** -O disaster mrhat lollipop
Starting nmap V. 2.12 by Fyodor (fyodor@dhp.com, www.insecure.org/nmap/)
Interesting ports on disaster.xor.com (192.108.21.99):
...
Remote operating system guess: HP-UX 11.0
Interesting ports on disaster.xor.com (192.108.21.2):
...
Remote operating system guess: BSD1 4.0
Interesting ports on disaster.xor.com (192.108.21.48):
...
Remote operating system guess: Solaris 2.6 - 2.7

Nmap run completed – 3 IP address (3 host up) scanned in 5 seconds

Esta característica puede ser útil para tomar un inventario de una red local. Desafortunadamente, ésta también es muy útil para los *hackers*, quienes pueden basar sus ataques sobre las debilidades conocidas del sistema operativo escpecificado.

5.4.2. SAINT: Examina sistemas en red por vulnerabilidades

Como **nmap**, SAINT prueba computadoras sobre una red para averiguar qué servidores están corriendo, pero, a diferencia de **nmap**, SAINT conoce completamente todo sobre los programas actuales del servidor de UNIX y sus vulnerabilidades históricas. Éste busca las configuraciones erróneas que degradan a la seguridad y, también checa la presencia de *bugs* conocidos.

Porque un reporte de SAINT esencialmente provee instrucciones para entrar en un sistema, se recomienda que los administradores deberían correr SAINT en el sistema antes de que lo hagan los *hackers*.

La interfaz del usuario de SAINT está basada completamente en la web y, requiere que un browser de web esté instalado en la máquina, en la cual se está corriendo SAINT.

Más información:

SAINT
www.wwdsi.com

Nessus: Nueva generación de exploradores de red
www.nessus.org

5.4.3. crack: Encuentra contraseñas inseguras

Puesto que algunos vendedores todavía distribuyen sistemas que dejan contraseñas encriptadas en una vista plana, los hackers traviesos pueden fácilmente compararlas con un diccionario encriptado. Una manera de adelantarse a ese ataque

es que el administrador haga una comparación y force a los usuarios a cambiar contraseñas que él ha roto. **crack** es una herramienta sofisticada por Alec D. E. Muffett, que implanta varias técnicas comunes de contraseñas imaginables.

Incluso si se usa un archivo de contraseña sombra *(shadow password)* para ocultar contraseñas encriptadas de la vista pública. Es todavía deseable verificar qué contraseñas de los usuarios son resistentes a **crack**. El conocimiento de la contraseña del usuario puede ser útil porque la gente tiende a usar la misma contraseña una y otra vez. Una contraseña simple puede proveer acceso a otro sistema, destruir archivos guardados en el directorio *home* del usuario y, permitir acceso a cuentas financieras en la red.

Ya que la salida de **crack** contiene las contraseñas que han sido rotas, se debería cuidadosamente protegerlas y borrarlas tan pronto como se termine de correr el programa.

Se puede bajar de:
ftp.cert.org

Más información:

COPS: Auditoria del sitema de seguridad
www.cerias.purdue.edu

tripwire: Monitor de cambios de los archivos del sistema
www.tripwiresecurity.com

Herramientas de forense
www.fish.com/security

5.5. Herramientas de seguridad criptográficas

La criptografía provee una solución a varios problemas de seguridad. Ésta ha sido posible desde hace mucho tiempo para mezclar mensajes, para que de esta

manera, un escuchador furtivo (eavesdropper) no pueda decifrarlos, pero esto es sólo el principio de las maravillas de la criptografía. Desarrollos tales como criptografía de llave pública y la mala seguridad han permitido el diseño de 'criptosistemas' (cryptosystems) que encuentran casi cualquier necesidad concebible[48].

5.5.1. Kerberos: Una aproximación unificada a la seguridad de red

El sistema Kerberos, diseñado en MIT, intenta dirigir algunos de los aspectos de seguridad de red en una manera consistente y extendible. Kerberso es un sistema de autenticación, una facildad que "garantiza" que los usuarios y servicios son de hecho quienes dicen ser. Esto no provee alguna seguridad adicional o encriptación más allá de esto.

Kerberos usa DES para construir conjuntos anidados de credenciales llamados "tickets". Los tickets son pasados alrededor de la red para certificar la autenticidad del usuario y para proveerle los servicios de red. Cada sitio Kerberos debe mantener al menos una máquina segura fisicamente (llamado el servidor de autenticación) para correr el demonnio Kerberos. Éste demonio entrega tickets a los usuarios o servicios que solicitan autenticación basada en credenciales que ellos proveen, tales como contraseñas.

En esencia, Kerberos mejora a la seguridad de contraseña de UNIX tradicional sólo en dos maneras: Éste nunca transmite contraseñas sin encriptar sobre la red y, éste ayuda a los usuarios de tener que teclear contraseñas repetidamente, haciendo la protección de contraseñas de los servicios de red un poco más aceptable.

Más información sobre Kerberos:

web.mit.edu/kerberos
http://web.mit.edu/kerberos/www/dialogue.html
http://www.nrl.navy.mil/CSS/people/kenh/kerberos-faq.html

[48]Dos excelentes recursos para aquellos interesados en la criptografía son "Preguntas contestadas frecuentemente (FAQ) de RSA Labs, sobre la criptografía en nuestros días en *www.rsasecurity.com/rsalabs/faq* y, el FAQ de sci.crypt disponible por FTP de *rtfm.mit.edu*

Más información:

PGP: Pretty Good Privacy
www.pgpi.org

5.5.2. SSH: El shell seguro

Escrito por Tatu Ylonen, remplazo a rcp, rlogin y telnet; ssh encripta la auntentificación para confirmar la identidad del usuario, así como también encripta la comunicación entre los dos *hosts*. El protocolo ha sido estudiado y está siendo estandarizado por el IETF.

Se puede obtener una versión no comercial de SSH2, pero se recomienda trabajar con SSH1 porque es más sencillo de instalar.SSH1 se puede obtener de ftp.es.hut.fi/pub/ssh. Sin embargo ésta versión ya no está siendo desarrollada, pero el grupo OpenBSD tomó el código fuente y lo está reestructurando así como también le da mantenimiento, a esta versión se conoce como OpenSSH y se puede obtener en www.openssh.com.

SSH es un programa, que permite hacer un login remoto y ejecutar comandos en una máquina remota. Con *ssh*, se pretende reemplazar a *rlogin* y *rsh*, proporcionando comunicaciones encriptadas, entre dos *hosts* no confiables en una red insegura. Las conexiones X11 y los puertos arbitrarios TCP/IP, también pueden ser reenviados a través de un canal seguro.

Los componentes principales de **SSH** son:

- El demonio del servidor **sshd** y,
- Dos comandos de nivel usuario **ssh** para la conexión remota y **scp** para copiar los archivos.

La conexión remota con **sshd** se puede hacer de diferentes formas; el administrador es quien decide qué método usar.

- **Método A:** Si el nombre del host remoto está en alguno de estos archivos **/.rhosts, /.shosts, /etc/hosts.equiv o /etc/shots.equivsshd**, entonces el usuario es logeado automáticamente sin pedir contraseña. No es muy confiable este método.

- **Método B:** *sshd* también puede usar una llave pública encriptada para verificar la identidad del host remoto. Para que esto suceda la llave del host remoto debe estar listada en el archivo del localhost **/etc/ssh.known_hosts** o en el archivo del usuario **/.ssh/known_host.** Si el host remoto prueba que la llave es conocida entonces el usuario es *logeado* sin haber preguntado por el *password*. Este método es más seguro que el método A.

- **Método C:** *sshd* puede usar una llave encriptada para establecer la identidad del usuario. En tiempo real, el usuario debe tener acceso para copiar el archivo de su llave privada y debe reemplazar el *password* encriptado. Este método es el más seguro, pero el usuario no se puede *logear* cuando no tenga una copia del archivo.

- **Método D:** Finalmente, *sshd* puede permitir al usuario entrar con su login y contraseña. Este proceso es parecido al telnet, excepto porque el login y la contraseña están encriptados. La principal desventaja de este método es que el login y la contraseña son débiles (están delimitados por 8 caracteres) y que existen algunas herramientas (como crack) diseñadas para encontrarlos.

La política de autenticación está en el archivo **/etc/sshd_config**. Las opciones relevantes de la autenticación son mostradas en la siguiente tabla:

SSH permite el acceso a un hostname específico. El usuario debe probar su identidad en la máquina remota, usando uno de los distintos métodos según la versión del protocolo utilizado.

Protocolo SSH Versión 1

Si los archivos *etc/hosts.equiv ó etc/ssh/shosts.equiv*, tienen registrado el nombre de la máquina a la que el usuario desea tener acceso, y los nombres de usuario son el mismo en ambos lados, se le permite al usuario acceder a ésta. Por otro lado, si el directorio *home* del usuario ($HOME) en la máquina remota, contiene los archivos *.rhosts ó .shosts*, y éstos, a su vez tienen registrados el nombre de la máquina cliente y el nombre del usuario, se le permite el acceso al usuario. Esta forma de auntentificación, normalmente no es permitida por el servidor, por razones de seguridad.

El método *rhosts ó hosts.equiv*, en combinación con la autentificación de *hosts* basado en RSA (Ravest, Shamir y Adleman), forman el segundo método de autentificación. Es decir, el login es validado por *$HOME/.rhosts*, *$HOME/.shosts*, */etc/ssh/shosts.equiv*, además, la llave host del cliente debe ser verificada por el servidor, y si ésta es válida, el login será permitido. Este método de autentificación, cierra hoyos de seguridad de *IP spoofing*, *DNS spoofing* y *routing spoofing*.

NOTA: Los programas */etc/hosts.equiv y $HOME/.rhosts*, así como los protocolos *rlogin/rsh* en general, son inherentemente inseguros y deben ser deshabilitados.

Como un tercer método, SSh soporta autentificación basada en la encriptación RSA. Este esquema es basado en la criptología de llave pública, lo cual significa, que existen criptosistemas donde la encriptación y desencriptación, son efectuadas usan-

Opción	**Método**	**Significado cuando se activa**
Autenticación de rhosts	A	Permite el login vía /.shosts, /etc/shosts.equiv, etc.
Autenticación rhostsRSA	B	Permite /.shosts et al., requiere de la llave del host.
Ignora Rhosts	A,B	Ignora los archivos /..rhosts y hosts.equiv.
Ignora root rhosts	A,B	Previa auntentificación por root en rhosts/shosts.
Autenticación RSA	C	Permite publicar por usuario la llave encriptada.
Autenticación password	D	Permite usar el login y password.

Cuadro 5.2: Tabla de Autenticación en el archivo /etc/sshd_config

do llaves separadas, por lo que no es posible derivar la llave de desencriptación con la encriptación. Lo ideal, es que cada usuario cree una llave par pública/privada, con propósitos de autentificación. El servidor conoce la llave pública, y sólo el usuario conoce la llave privada. El archivo *$HOME/.ssh/authorized_keys*, contiene las llaves públicas permitidas para el acceso. Cuando el usuario obtiene acceso el programa *ssh* le dice al servidor que la llave par debe usar para la autentificación. El servidor verifica si esta llave es permitida, y si es así, envía al usuario un reto, es decir un número aleatorio encriptado por la llave pública del usuario. Después de que el usuario desencripta el reto usando la llave privada, demuestra que conoce la llave privada, sin revelarlo al servidor.

El *ssh*, implementa automáticamente el protocolo de autentificación RSA. El usuario, crea su llave par RSA corriendo *ssh-keygen*. Esto almacena la llave privada en *$HOME/.ssh/identity*, y la llave pública en *$HOME/.ssh/identuty.pub* en el directorio home del usuario. El usuario debe copiar el archivo *identity.pub* en *$HOME/.ssh/authorized_keys* en la máquina remota, (el archivo *authorized_keys*, corresponde al archivo convencional *$HOME/.rhosts*, y tiene una llave por línea, aunque las líneas pueden ser muy largas). Después de esto, el usuario puede tener acceso sin dar su password. Cabe señalar que la autentificación RSA, es más segura que la autentificación *rhosts*.

La manera más conveniente de usar la autentificación RSA, puede ser con un agente de autentificación *(man ssh-agent)*.

Si algún método de autentificación falla, el programa *ssh* solicita el password al usuario, enviándolo al *host* remoto para su verificación. Debido a que la comunicación está encriptada, el *password* no puede ser visto por alguien que esté escuchando la red.

Protocolo SSH Versión 2

Cuando un usuario se conecta usando el protocolo ssh versión 2, están disponibles distintos métodos de autentificación. El cliente se autentificará usando el método de

la llave pública. Si esto falla, se intenta la autentificación de password.

El protocolo 2 provee macanismos adicionales de confidenciabilidad (el tráfico es encriptado usando 3DES, Blowfish, CAST128 o Arcfour), e integridad (hmac-md5, hmac-shal). Como puede observarse, el protocolo 1 carece de un mecanismo fuerte para asegurarse de la integridad de la conexión.

Acceso y ejecución remota

Cuando la identidad del usuario es aceptada por el servidor, éste ejecuta el comando dado, y asigna al usuario un shell normal en la máquina remota. La comunicación con el comando remoto o shell, será encriptada automáticamente.

Si una pseudo-terminal ha sido asignada (sesión normal de login), el usuario podrá desconectarse con ~ y suspender ssh con **crtl-z**. Todas las conexiones reenviadas pueden ser listadas con ~#, y si la sesión espera por conexiones TCP/IP ó conexiones X11 reenviadas, ésta será enviada a *background* con ~&, el cual no deberá ser usado mientras el shell del usuario se encuentre activo, ya que puede causar que el shell se bloquee. Todas las secuencias de escape, pueden ser listadas con ~?.

Un carácter tilde sencillo, puede enviarse como ~~. El carácter de escape, siempre debe seguir una nueva línea para ser interpretada como especial y puede ser cambiado en archivos de configuración, o en la línea de comandos.

Si una terminal (no pseudoterminal) ha sido establecida, la sesión es transparente y puede ser utilizada para transferencias confiables de información binaria. En la mayoría de los sistemas, puede establecerse el carácter de escape a **none**, para obtener una sesión transparente aún cuando se esté empleando una pseudoterminal.

5.6. Firewall

La seguridad resulta cada vez más importante tanto para las compañías como para los individuos. Internet les ha proporcionado una poderosa herramienta para distribuir información entre ellos y para obtener información de otros, pero también les ha expuesto a peligros de los que habían estado exentos hasta entonces. La criminalidad informática, el robo de información y el daño mal intencionado constituyen peligros potenciales.

Una persona no autorizada y sin escrúpulos que consiga el acceso al sistema de una computadora puede que averigüe contraseñas del sistema o que se aproveche de los errores y del comportamiento particular de ciertos programas para obtener una cuenta funcional en dicha máquina. Una vez que sea capaz de entrar en la máquina, puede que tenga acceso a información que podría resultar dañina, información tan sensible comercialmente como los planes de negocio, detalles de nuevos proyectos o las bases de datos con información de los clientes.

Una de las formas principales de implementar un sistema de seguridad en una red de datos es a través de un **firewall**.

5.6.1. Definición de firewall

Un firewall o cortafuego en el mundo de las redes de computadoras es un dispositivo físico o lógico que permite a una red, tener conexión a Internet con cierto grado de seguridad. Existe una gran variedad de ataques o formas de violar la seguridad de un sistema o red. Un firewall permite combatir diferentes tipos de ataques que son conocidos en Internet:

- Intrusión. Es el más común, con una intrusión, las personas pueden utilizar una computadora sin tener autorización para ello. Las formas de realizar una intrusión en un sistema son muy variadas, desde robarse una contraseña de una cuenta, hasta valerse de errores de algún componente de software para obtener acceso a la computadora sin poseer una cuenta en la misma.

- Negación de Servicios (Denial of Service). Éste es un tipo de ataque que busca inutilizar los servicios que presta una computadora. La forma más común de realizar este tipo de ataques es inundar un sistema o red con mensajes, procesos o requerimientos de servicio, de tal manera que el sistema o red es incapaz de satisfacer las solicitudes de los usuarios legítimos.

- Robo de Información. Este ataque permite al intruso obtener información sin haber utilizado directamente la computadora. Generalmente estos ataques explotan servicios de Internet que son utilizados para proveer información, induciendo a estos servicios a dar más información a personas equivocadas.

El núcleo de **GNU/Linux** proporciona un rango de características internas que permiten funcionar bastante bien como un *firewall* de IP. La implementación de red incluye código para realizar filtros a nivel de IP en numerosas formas, y proporciona un mecanismo para configurar con precisión qué tipos de reglas se impondrían.

El software de *firewall* de Linux proporciona otras dos características muy útiles: **auditoría de IP** y **enmascaramiento de IP**.

Filtrado de IP

El filtrado de IP es simplemente un mecanismo que decide qué tipos de datagramas de IP serán procesados normalmente y cuáles serán descartados. Por descartados se entiende que el datagrama se elimina y se ignora completamente, como si nunca se hubiera recibido. Se pueden aplicar muchos criterios, y en diferentes ordenamientos, para determinar qué datagramas se desea filtrar; algunos ejemplos de esto son:

- Tipo de protocolo: TCP, UDP, ICMP, etc.
- Número de socket (para TCP/UDP)
- Tipo de datagrama: SYN/ACK, datos, petición de eco de ICMP, etc.
- Dirección de origen del datagrama: de dónde proviene

- Dirección de destino del datagrama: a dónde se dirige

El filtrado de IP es una utilidad en la capa de red. Esto significa que este mecanismo no entiende nada acerca de la aplicación que utiliza las conexiones de red, sólo sabe acerca de las conexiones mismas. Por ejemplo, se puede denegar el acceso a usuarios a la red interna por el puerto predeterminado de telnet, pero si se apoya únicamente en el filtrado de IP, no podrá evitar que se utilice el programa de telnet en un puerto por el que se permite el paso a través del *firewall*. Se puede evitar este tipo de problemas haciendo uso de servidores intermediarios para cada servicio que permita que cruce el *firewall*. Los servidores intermediarios comprenden la aplicación para la que fueron diseñados y por tanto evitan los abusos, tales como utilizar el programa de telnet para pasar a través de un *firewall* utilizando el puerto de World Wide Web. Si el firewall soporta un servidor intermediario de World Wide Web, aquella conexión de telnet será siempre respondida por el servidor intermediario que sólo permitirá que pasen peticiones HTTP. Existe un gran número de programas servidores intermediarios. Algunos son software libre y muchos otros son productos comerciales.

Configuración de Linux como firewall

Para poder construir un firewall IP con Linux, es necesario disponer de un núcleo compilado con soporte de *firewall* de IP y de la utilidad de configuración adecuada. En todos los núcleos anteriores a la serie 2.2 se usaba la utilidad *ipfwadm*. Los núcleos 2.2.x supusieron el lanzamiento de la tercera generación de firewall de IP para Linux que se denominó **IP chains** utiliza un programa similar a *ipfwadm* que se llama *ipchains*. Los núcleos de Linux 2.3.15 y siguientes soportan la cuarta generación de *firewall* de IP de Linux que se denomina *netfilter*. El código de netfilter es el resultado de un gran rediseño del flujo en el manejo de paquetes en Linux. **Netfilter** es una criatura con múltiples caras, pues proporciona un soporte compatible hacia atrás tanto con ipfwadm como con *ipchains* además de una nueva orden alternativa que se llama *iptables*.

5.6.2. Linux ipchains

En Linux, el filtrado de paquetes es manejado en el kernel. *ipchains* permite ver el encabezado de los paquetes que pasan por el sistema y decide qué hacer basándose en las políticas de seguridad. *ipchains* puede tomar cualquiera de las siguientes decisiones:

- Denegar los paquetes, es decir, descartarlos como si nunca los hubiera recibido.
- Aceptar los paquetes, es decir, dejar que pasen a través del firewall.
- Rechazar los paquetes, es como denegarlos pero se le informa al origen del paquete lo que ocurrió.

El soporte del *firewall ipchains* fue desarrollado por Paul Russell y Michael Neuling. Paul es el autor del documento sobre IP Chains IPCHAINS-HOWTO.

Cadenas para filtrado

El kernel de Linux contiene tres reglas de filtrado, estas listas son llamadas cadenas (chains). Estas listas tienen el nombre de *input, forward y output*. Cuando un paquete llega a la máquina que funciona como firewall por una de sus interfaces, el kernel utiliza la cadena *input* para decidir su destino. Si el paquete supera este paso, el kernel decide a dónde enviar el paquete basándose en la tabla de enrutamiento. Si el paquete está destinado a otra máquina, el kernel consulta la cadena *forward*. Por último antes de enviar el paquete, el kernel consulta la cadena *output*.
Una cadena es una lista de reglas. Cada regla dice: si el paquete cumple con esto, entonces haga aquello con el paquete. Si la regla no corresponde al paquete, entonces se consulta la regla siguiente. Por último, si ninguna de las reglas es aplicable al paquete, el kernel revisa la política por omisión de la cadena para decidir qué hacer. Esta política por omisión puede ser cualquiera de las acciones, denegar, aceptar o rechazar.

Las etapas que se muestran en el tráfico son:

- En primer lugar el paquete es comprobado con la cadena *input*. Si la decisión no es denegar o rechazar, el paquete continúa.

- Luego se realiza la decisión de enrutamiento basándose en el destino del paquete. Con esto se decide si el paquete es para un proceso local o es para otra máquina.
- Un proceso local puede recibir o enviar paquetes que pasan a través de la etapa de decisión de enrutamiento.
- Si el paquete no fue creado por un proceso local, el paquete es enviado a la cadena *forward*.
- La cadena *forward* se aplica a cada paquete que trata de pasar a través de una máquina hacia otra.
- Por último a todo el paquete que sale de la máquina se le aplica la cadena *output*.

Uso de ipchains

ipchains es el comando que permite manejar las cadenas de *firewall* en el kernel. Existen opciones para manejar cadenas completas. Siempre se tiene inicialmente tres cadenas que no se pueden eliminar. Las opciones para manejar las cadenas son (siempre en mayúsculas):

- **N**: Crear una nueva cadena.
- **X**: Borrar una cadena vacía.
- **P**: Cambia la política de la cadena de reglas. Ésta puede ser ACCEPT, DENY, REJECT, y MASQ (ésta es sólo válida en forward).
- **L**: Lista las reglas de la cadena de reglas.
- **F**: Borra todas las reglas.
- **Z**: Pone a cero todos los contadores de bytes en todas las reglas de la lista.

Los comandos para manipular las reglas que están dentro de la cadena:

- **A**: Añade una nueva regla a la cadena (la añade al final).
- **I**: Inserta una regla en una posición indicada.
- **R**: Reemplaza una regla.
- **D**: Borra una regla.

Las reglas suelen seguir el formato de ipchains -(ADIR) opciones -j (salto), donde:

- *salto*: Si el paquete coincide con la mencionada regla se saltará a donde indique -j que puede ser aceptar, denegar, reejecutar u otra cadena definida por el usuario. De todas formas no es imprescindible el -j.
- *opciones*:
  - **-s** fuente del paquete. Se puede expresar redes o IP, se sigue el formato:[red–ip]/mask. Si no se pone mask se usa por defecto la 32. Con 0/0 referencias a todo el mundo, también se pueden especificar los puertos, éstos se especifican al final, tras ls ip/mask y puede ser un puerto solo o un rango que se expresa separando los puertos límites por un " : ", para expresarlos se pueden usar números o el nombre (que se pueden encontrar en */etc/services*).
  - **-d** igual que **-s** pero para destino.
  - **-p** especifica el protocolo (TCP, UPD, ICMP). Se pueden poner también los números equivalentes. Con el protocolo *ICMP*[49] , se puede especificar el tipo y código, se puede poner en *-s* y *-d* el nombre del paquete ICMP o si se prefiere el tipo en *-s* y el código en *-d*.
  - **-i** especifica la interfaz por el que entra el paquete (en input) o sale (en output y forward). Se pueden especificar interfaces inexistentes. Se permite el uso del comodín " + "para designar un conjunto de interfaces.

[49]Internet Control Message Protocol, se encarga de manejar y controlar los mensajes de error producidos durante el intercambio de información entre computadoras

- **-y** referencia a los paquetes SYN que son los que usan para iniciar una conexión. Con ! se hace referencia a los que no son para iniciar una conexión.
- **-f** la regla sólo se aplicará al segundo y demás fragmentos de un paquete, no se permite especificar puertos.
- **-j** especifica el objetivo de la regla puede ser: ACCEPT, REJECT y DENY (deniegan), MASQ (sólo válida en forward), REDIRECT (redirecciona a otro puerto o máquina), RETURN (aplica la política por defecto). Se puede especificar otra cadena de reglas definidas por el usuario con lo que se aplicarán las reglas de esa cadena y luego se volverá el original. Si no se especifica -j, la regla sólo realizará una actualización de la cuenta, esto es, se pueden contar el número de paquetes que cumplen la regla sin tomar acción sobre ellos (también se cuentan cuando se usa -j).
- **-l** si un paquete coincide con la regla se registra en el syslog.

Activación del firewall en el arranque

El *firewall* se activará al iniciarse el sistema justo después de activarse o levantarse todos los servicios en el arranque. Para llevar a cabo esto se editan los archivos */etc/rc.d/rc.local* y al final de éste se escribirá la siguiente línea:

 # activación de las reglas del firewall
/etc/rc.d/init.d/firewall start

5.7. Software de Monitoreo

Se tiene la posibilidad de monitorear la carga de CPU, memoria RAM, etc., de cada uno de los nodos que componen el cluster. De este modo podemos tener un control total del funcionamiento del conjunto, y evitar sobrecarga en los nodos, detectar mal funcionamiento, derrivar procesos de manera optima, entre otros.

Otro aspecto muy importante en el cluster es la monitorización de los servicios; si alguno de los nodos falla, se tendrá que advertirlo de alguna forma y desencadenar las acciones pertinentes (eliminarlo de la lista de nodos activo).

5.7.1. bWatch Beowulf Watch

La herramienta *bWatch 1.0.3* es un pequeño programa TCL/TK, el cual muestra en una ventana un conjunto de información acerca de cada uno de los nodos que conforma el cluster, es decir, es un sistema de monitorización del cluster.

La información que nos provee dicho software es:

- Host name.
- Nº de usuarios conectados.
- Hora en la cual se hizo la toma de los datos.
- Carga hace 1 min.
- Carga hace 5 min.
- Carga hace 15 min.
- Número de procesos.
- Memoria total.
- Memoria libre.
- Memoria compartida.
- Buffers.
- Cache.
- Total de swap.
- Swap libre.

Las últimas versiones de este software están disponibles en:
ftp://www.sci.usq.edu.au/pub/jacek/bWatch/.

bWatch puede ser ejecutado por cualquier usuario mientras éste pueda ejecutar RSH (Remote Shell) en todas las máquinas que conforman el cluster; fue desarrollado para plataformas LINUX.

bWatch es abreviación de Beowulf Watch. Es un script Tcl/Tk diseñado para supervisar la carga y el uso de memoria en todos los nodos de un cluster Beowulf; no requiere permisos de *root* y puede ser ejecutado por cualquier usuario del sistema, mientras pueda ejecutar rsh en todas las demás máquinas.
Puede hallarse un archivo de ejemplo de configuración en:
/usr/share/doc/packages/bwatch/bWatchrc.tcl.sample.

http://g.unsa.edu.ar/cgi-bin/pacshow.pl?es+bwatch

5.7.2. SCMS Smile Cluster Management System

SCMS es un sistema que facilita el manejo de un cluster al proveer un grupo de módulos que permiten realizar algunas de las tareas de administración de un conjunto de nodos de manera centralizada. Es decir, desde un nodo que podría llamarse la consola del cluster. La palabra *Smile* indica que el sistema de administración de clusters fue desarrollado originalmente para el cluster con tal nombre ubicado en Kasetsart University, en Tailandia. *SCMS* es un ambiente desarrollado para ser utilizado a través de una interfaz gráfica, pero también provee algunos comandos que pueden ser ejecutados desde modo texto.

El conjunto de módulos que conforman SCMS son:

Módulo de Administración del Sistema. Este módulo proporciona un conjunto de comandos básicos para la operación del sistema tales como *shutdown, reboot*, ejecución de comandos de control remotamente, etc. El administrador puede seleccionar si el comando debe aplicarse sobre un nodo, un conjunto de nodos o sobre todo el cluster.

Módulo de Configuración del Sistema. Este módulo permite revisar y mostrar

la configuración de hardware y software que está presente en el cluster o en un nodo en particular.

Módulo de Monitoreo del Sistema. Es un sistema de monitoreo análogo al bWatch en donde se registra en tiempo real la carga y el uso que está recibiendo el sistema.

Módulo de Comandos Paralelos. Consiste en un grupo de comandos basados en la especificación de herramientas UNIX escalables del consorcio de herramientas paralelas. Estos comandos son extensiones de algunos comandos de UNIX que tienen la particularidad que cuando son invocados, entonces son ejecutados simultáneamente sobre todos los nodos del cluster (o sobre un subconjunto de ellos). Algunos de estos comandos paralelos son:

- **Copia Paralela**: Copia archivos en los nodos del cluster.
- **Borrado Paralelo**: Elimina archivos en los nodos del cluster.
- **Búsqueda Paralela**: Consigue archivos en un cluster.
- **Listado Paralelo**: Lista los archivos almacenados en un cluster.
- **Carga Paralela**: Reporta la carga de los nodos de un cluster.
- **Búsqueda de Procesos Paralela**: Consigue un proceso que se está ejecutando en un cluster a través de su nombre.
- **Eliminación de Procesos Paralela**: Elimina un proceso que se está ejecutando en un cluster a través de su nombre.

Como puede observarse, estos comandos paralelos son muy útiles, especialmente en ambientes donde los nodos de un cluster tienen instalado el sistema operativo y las aplicaciones localmente.

Obtención e Instalación del Software SCMS

Obtener la última versión del paquete de la dirección web
http://smile.cpe.ku.ac.th/software/scms/

Instalar el paquete en el cluster siguiendo los pasos indicados en la guía de instalación localizada en la dirección web

http://smile.cpe.ku.ac.th/software/scms/scms_howto.html

5.7.3. SCE Scalable Cluster Environment

El software *SCE* es un conjunto de herramientas, que permiten la construcción y administración de un cluster, facilitando con ello las tareas del administrador.

Fue desarrollado por el grupo de investigación paralela, en el Computer and Network System Research Laboratory, Departamento de Ingeniería en Computación, de la Facultad de Ingeniería, de la Universidad de Kasetsart Tailandia.

Uno de los problemas con la adopción de clusters para cálculo de alto desempeño es la dificultad en la construcción y manejo del sistema por lo que su objetivo es satisfacer estas necesidades a través de un conjunto de herramientas, compuesta por:

- **Beowulf Builder.** Para construir clusters.
- **SCMS.** Para la administración de sistema.
- **KCAP.** Monitoreo en tiempo real escalable a través de Internet.
- **SQMS.** Comandos paralelos de UNIX y calendarizador.

Este software corre sobre una capa denominada middleware que proporciona al cluster amplio control de los procesos y muchos servicios.

Incluye las bibliotecas:

- *MPICH*
- *MPI*
- *PVM*

Todas las herramientas en SCE se diseñaron para estar verdaderamente integradas, además de proporcionar más de 30 APIs para acceder a la información de los recursos de sistema, control remoto de la ejecución de procesos, administración del cluster y otras. SCE se diseñó para ser muy fácil de usar. La mayor parte de la instalación y la configuración están completamente automatizadas por una interfaz gráfica de usuario.

Requerimientos:

- *dhcp*
- *nfs-utils*
- *tk*
- *python*
- *ncurses*
- *perl*
- *chkconfig*

Distribución:

SCE 1.0 se presenta en 2 formatos:

- *sce-1.0.iso*
- *sce-1.0.tar.gz*

Instalación:

- *Descargar SCE 1.0 del*http://pgr.cpe.ku.ac.th/
- *Extraer los archivos*
  $su -
  #tar zxvf sce-1.0.tar.gz
- *Correr ./setup*
  #cd sce
  #./setup
- *El instalador despliega un mensaje de bienvenida*
- *Leer la licencia y aceptar las condiciones de uso*
- *Escoger los paquetes que desea instalar en su sistema*
- *Confirmación de los paquetes a instalar*
- *Si la instalación no tiene errores muestra una caja de diálogo final*
- *El siguiente paso es construir el cluster*:
  - Permitir que Beowulf builder construya el cluster
  - Usar la configuración actual

BB Builder Cluster Es una herramienta que ayuda a la contrucción de un cluster beowulf diskless fácilmente.

Características:

- *Soporta booteo remoto usando el paquete Etherboot*
- *Interfaz de usuario fácil de usar*
- *Configuración de los servicios del sistema para todos los nodos*
- *Cargar y salvar configuración*
- *Generación automática de todos los archivos de configuración*

- *Soporta más de 254 nodos*

SCMS Smile Cluster Management System

Herramienta de administración de clusters interactiva. Su objetivo es permitir al administrador realizar sus tareas fácilmente. *SCMS* posee una poderosa interfaz gráfica, gran cantidad de comandos, monitoreo de subsistemas en tiempo real, interfaz basada en WEB, entre otras cosas. Con SCMS, las tareas de administración del sistema de un gran cluster son mucho más simples.

Características:

- *Herramientas de administración del sistema*
- *Monitoreo del desempeño en tiempo real*
- *Comandos paralelos UNIX*
- *Configuración a través de un navegador*

KCAP Kasetsart Cluster Administration Program

Es un software basado en WEB que ayuda monitorear y administrar un Cluster Beowulf, puede visualizar y desplegar los recursos del sistema interactivamente. Por otra parte, *KCAP* provee muchas características que ayudan tanto al administrador como a los usuarios para usar el sistema eficientemente. *KCAP* es una parte adicional de *SCMS* (éste necesita ser instalado antes de KCAP).

Características:

La configuración del cluster puede ser mediante un navegador de Web estándar. Despliega información útil de algún nodo o de todos, esta información incluye:

- *Mensajes del Kernel*
- *Módulos del Kernel*
- *Configuración de red*

- *Espacio en disco duro*
- *Espacio de memoria*
- *Estado de los usuarios*
- *Estado de los procesos*

Usando applets Java despliega información en tiempo real como:

- *CPU*
- *Temperatura de la tarjeta*
- *Carga promedio*
- *Memoria*
- *Paginación*
- *Swap*

SQMS Simple Queuing Management System

Es un administrador de colas del sistema desarrollado para simplificar las tareas de administración, provee un grupo de comandos para soportar desarrollos en ambientes de Clusters Beowulf para enviar, consultar estados, borrar sus programas de los nodos de cálculo, etc.

Características:

- *Soporta cualquier tipo de Clusters*
- *Soporta tareas secuenciales y paralelas, actualmente sólo programas en MPI*
- *El desarrollador puede crear fácilmente una nueva Política de Balanceo de Carga usando APIs*

IX Kasetsart Advanced System Interconect Executive

Es una parte muy importante del software *SCE*, es un ambiente paralelo dentro del Cluster Beowulf, provee una máquina virtual con un conjunto de APIs para muchos servicios como: control de procesos globales, servicio de nombres, administración conjunta, distribución de servicios. Las APIs permiten a los programadores construir aplicaciones fácilmente para aprovechar el poder del cluster. No se requiere modificar el Kernel para correrlo, sus características permiten una fácil instalación y es altamente portable.

5.7.4. OSCAR Open Source Application Resources

OSCAR es una herramienta de software que consta de un grupo de componentes altamente integrados (RPMS, scripts de Perl, bibliotecas, herramientas, etc.), diseñados para facilitar la contrucción, el mantenimmiento y la operación de un cluster Linux. Una de las características más importantes de esta herramienta es que se trata de un proyecto de software libre. Oscar fue desarrollado por el Open Cluster Group, una colaboración de centros de investigación y compañías de alta tecnología encabezados por el Oak Ridge National Laboratory (ORNL), el National Center of Supercomputing Applications (NCSA), IBM e Intel. Otros colaboradores son Dell, SGI, MSC Software, Veridian y Ericsson.

Los componentes software se encargan de automatizar las tareas más importantes para crear un cluster:

- *Instalación de Linux en los NODOS*
- *Construcción de una base de datos con información de cada uno de los nodos*
- *Seguridad*
- *Administración del sistema*
- *Instalación de bibliotecas y herramientas necesarias para desarrollar programas paralelos*
- *Manejo de la carga de trabajo en clusters multiusuario*

- *Documentación y manejo de paquetes*

Las herramientas que hacen posible realizar todas estas tareas de manera semi-automática vienen totalmente integradas con OSCAR con lo cual lo único que se necesitaría para construir el cluster es descargar de la página de internet http://www.csm.ornl.gov/oscar/software.html.

Los componentes de OSCAR que implementan cada una de las funcionalidades mencionadas son:

- *LUI Linux Utility for cluster Install.* Herramienta soportada por IBM realizada bajo licencia GPL. Es utilizada para instalar clusters Linux heterogéneos a través de la red. LUI también construye y mantiene una base de datos del cluster que incluye los nombres de los nodos, información de red, datos de configuración del cluster, y cualquier otra información requerida para instalar los otros componentes de OSCAR.

- *OpenSSH/OpenSSL.* Versiones Open Source de los protocolos de red que nos permiten soportar comunicaciones seguras entre computadoras generando claves para manejar autentificación y transferencia de archivos de manera cifrada.

- *C3 Cluster Command & Control Suite.* Dentro de un cluster OSCAR, cada computadora corre su propia copia del sistema operativo, sin embargo, en ciertas operaciones es necesario ver el cluster como un solo elemento, la herramienta que nos permite hacer esto es C3, la cual está formada por un conjunto de scripts de Perl que nos posibilitan administrar eficientemente nuestro equipo.

- *Ambientes de programación: MPI, PVM.* OSCAR dispone de las herramientas necesarias para escribir software que corra sobre el cluster, por tal motivo incluye las bibliotecas estándar de paso de mensaje: *MPI (Message Passing Interface) y PVM (Parallel Virtual Machine).*

- *PBS Portable Batch System.* Para asegurar que todos los recursos sean utilizados eficientemente dentro de clusters multiusuario es necesario que una herramienta administre la gran mezcla de trabajos. *PBS* permite optimizar

los recursos del cluster a través de la administración y la calendarización de trabajos. *PBS* monitoriza el estado del cluster, y es capaz de arrancar (o detener) tareas.

En cada caso el Open Cluster Group ha incluido los paquetes completos así como su documentación. *OSCAR 1.1* ha sido desarrollado y probado para Red Hat 7.1, pero en un futuro será capaz de soportar otras distribuciones como TurboLinux, Mandrake, Debian o SuSE.

Capítulo 6
Conclusiones

La necesidad de computadoras con grandes capacidades de procesamiento y de bajo costo (en hardware y software) han motivado la creación y posteriormente el uso de los clusters tipo Beowulf. La administración de clusters se ha vuelto cada vez más especializada y la documentación al respecto aumenta día con día. El presente trabajo de investigación es una guía de referencia a través de un cluster tipo Beowulf en particular, cuya utilización dentro de la Facultad de Ingeniería es de índole académica.

Desde un principio una de las metas de este proyecto fue comprobar experimentalmente que el cluster de la Facultad cumpliera con las características de un sistema distribuido (apertura, conectividad y compartición, capacidad de acelerar el cómputo, transparencia, escalabilidad, tolerancia a fallas y fiabilidad. Y cómo al llevar a cabo una eficiente administración mejora el rendimiento del sistema, sin perder de vista que el objetivo principal es acelerar el cómputo con el uso de hardware prácticamente obsoleto.

El administrador de un cluster de este tipo debe tener en mente que la elección de una plataforma y software abierto es de suma importancia. Esto garantiza que la parte principal del sistema mediante el cual se implemente sea abierto, esto es que el cluster implemente los mecanismos necesarios para ser funcional en pequeños componentes estándares y adaptables a las necesidades del cluster (no monolíticos). Como ejemplo de esto baste decir que en nuestro caso las labores cotidianas como la actualización del kernel y/o los módulos adecuados a cada tipo de nodo no sería posible si el código usado no fuese libre y gratuito.

Una idea primaria que es además una motivación de los sistemas distribuidos es conectar y compartir recursos del sistema, aplicaciones de software (como PVM y

MPI), servicios (TFTP, DHCP, NIS, NFS, etc.) y recursos (sistemas de archivos, memoria y procesador). A este respecto podemos concluir que el trabajo del administrador influye determinantemente en el aseguramiento de la conectividad, la correcta configuración de los servicios y por último en la implementación de políticas y mecanismos de seguridad del cluster.

La elección de nodos sin disco (diskless) reduce la complejidad de la administración (cero administración en los nodos) y el costo por cada nodo ya que no son necesarias instalaciones y/o actualizaciones de software en el cliente, elimina el costo por discos duros, baterías (no breaks), etc; y por todo esto prácticamente toda la administración del cluster se realiza en el servidor.

Como se describió el método de dividir una tarea que requiere de una gran cantidad de cómputo, en muchas tareas pequeñas sobresale como una de las principales tecnologías en el cómputo actual. En el procesamiento en paralelo los datos deben intercambiarse entre éstas tareas que cooperan entre sí. Algunas soluciones se han propuesto para realizar estas labores: memoria compartida, compiladores paralelizadores y el paso de mensajes.

La elección adecuada de una de ellas es vital para la aceleración en el cómputo que se lleva a cabo en el cluster. El paso de mensajes fue el paradigma de nuestra elección; basada en el número, la gran variedad de microprocesadores que soporta y la cantidad de aplicaciones, lenguajes y software de propósito general que lo usan. Existen dos implementaciones (bibliotecas) de paso de mensajes PVM y MPI. En el ámbito académico y para este tipo de cluster ambas implementaciones son funcionales.

El administrador del sistema a través de la adecuada instalación y configuración del hardware y software puede asegurar los siguientes aspectos relacionados con la transparencia: acceso a los recursos del cluster y su ubicación/reubicación, migración, respaldo, concurrencia, fallas y persistencia de los recursos del cluster. Un

ejemplo de lo anterior es la instalación y configuración de NFS, que permitió compartir los sistemas de archivos de tal manera que el usuario no percibe que en los nodos no existe disco alguno (transparencia en la ubicación); tampoco puede notar que el mismo recurso se accede concurrentemente por el resto de los nodos (transparencia en la concurrencia).

Aunque el cluster es tolerante a fallas, ya que el mal funcionamiento de uno de sus nodos no detiene la operación del cluster, si afecta el desempeño de éste; la labor de monitoreo previene y cuantifica las fallas y la disminución del desempeño en los nodos con la finalidad de anticipar y ejecutar las tareas convenientes para la restauración y/o corrección de éstas cuando y donde se presenten. La operación correcta del servidor en cambio es indispensable para que el cluster ejecute cualquiera de sus tareas, ya que sin éste no puede ser iniciada la operación de los nodos, ni paralelizados los procesos, tampoco se comparten los sistemas de archivos ni se autentifica a los usuarios, por tanto la mayor parte del trabajo del administrador es la correcta configuración del servidor y el monitoreo permanente de su desempeño y rendimiento. La realización de estas tareas y la implementación de mecanismos y políticas de seguridad adecuadas permitirán contar con un sistema confiable y de buen desempeño.

La escalabilidad también juega un papel muy importante en el rendimiento de cualquier sistema de cómputo. En el cluster de la Facultad la escalabilidad del sistema representó un reto muy grande debido a la escasez de recursos de hardware con que se contó; por otra parte la escalabilidad en términos del software se encuentra limitada por la misma razón ya que muchas de las aplicaciones de monitoreo y análisis de desempeño, balanceo de carga, seguridad y herramientas de programación, requieren de un hardware con mejores características y que además estén completamente soportados.

En lo referente a la escalabilidad el administrador es el responsable de buscar alternativas para que el cluster tenga un mejor desempeño, y por lo tanto que cuente

con la capacidad de atender a más usuarios ofreciendo a todos ellos más y mejores recursos y servicios.

Es importante no perder de vista que la escalabilidad del sistema no puede darse como una simple sustitución de dispositivos de hardware y sí debe procurarse que ésta se realice en todas las características importantes para el rendimiento del sistema, es decir el mejoramiento de microprocesadores, memoria, red de interconexión y almacenamiento secundario; y en forma obligada se debe tomar en cuenta el diseño original para explorar las debilidades o defectos de funcionalidad existentes y sacar el mayor provecho de los recursos que se agregarán. Debe considerar y cuantificar cómo afectará al sistema y prever las fallas derivadas de su instalación y configuración.

Respecto al software, podemos decir que la elección de aplicaciones de carácter abierto evita el problema de limitarse a la adquisición de hardware propietario y por tanto agrega flexibilidad al diseño o rediseño.

Bibliografía

[1] Michael J. Flynn. *Some Computer Organization and their Effectiveness.* IEE Trans, on Computer, vol. C-21, pp. 948-960, Sept. 1972.

[2] Chance Reschke, Thomas Sterling, Daniel Ridge, Daniel Savarese, Donald Becker, and Phillip Merkey. *A Design Study of Alternative Network Topologies for the Beowulf Parallel Workstation.* Proceedings Fifth IEEE International Symposium on High Performance Distributed Computing, 1996. http://www.beowulf.org/papers/HPDC96/hpdc96.html

[3] The Beowulf Project. *http://www.beowulf.org*

[4] Legends - Beowulf. *http://legends.dm.net/beowulf/index.html*

[5] Thomas L. Sterling, John Salmon, Donald J. Becker, Daniel F. Savarese. *How to Build a Beowulf A Guide to the Implementation and Application of PC Clusters* First Edition 1999 The MIT Press Cambridge, Massachusetts London, England P11 Figure 2.1 Classification Scheme for Parallel Computing

[6] PVM: Parallel Virtual Machine A Users Guide and Tutorial for Networked Parallel Computing Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, Vaidy Sunderam The MIT Press Cambridge, Massachusetts, London, England 1994 Massachusetts Institute of Technology Available on the Internet at: http://www.netlib.org/pvm3/book/pvm-book.html

[7] Andrew S. Tanenbaum. *Modern Operating Systems.* Prentice Hall. First Edition 1992, pp. 416-419.

[8] Dan I. Moldovan. *Parallel Processing rom Applications to Systems.* Morgan Kaufmann First Edition 1993, P26 1.4 Performance of Parallel Computations.

[9] Jacek Radajewski and Douglas Eadline, *Beowulf HOWTO*, Versión 1.1.1 22 November ('98).

[10] Sosa, García, y Casillas, *Análisis del desempeño de un Cluster de tipo Beowulf en diversos algoritmos de tipo concurrente*, año de realización: 2001.

[11] Daniel Manrique, *Construcción y evaluación de desempeño de un Cluster tipo Beowulf para cómputo de alto rendimiento*, Año de realización: Octubre 2001.

[12] Burk Robin, *UNIX: System Administrator's edition*, Edición ('97).

[13] Nemeth, Evi Conut, *Unix System administrator*, Edición ('95).

[14] Fiedler, David, *Unix System V release 4 administration*, Edición ('91).

[15] Ram Samudrala, *Linux Cluster HOWTO, http://www.ram.org/computing/linux/linux_cluster.html*, v0.3, August 21, 2001

[16] Kris Buytaert, *The openMosix HOWTO, http://howto.ipng.be/Mosix-HOWTO*

[17] Hank Dietz, *Linux Parallel Processing HOWTO*, v980105, 5 January 1998

[18] *Paper. High Performance Computing Cluster (HPCC) Benefits*, 2002.

[19] Joel C. Adams, W. David Laverell, Mark A. Ryken. *Paper. MBH'99: A Beowulf Cluster Capstone Project*

[20] Warren, M., Loki. *commodity Parallel Processing, http//:loki-www.lanl.gov/*

[21] *Paper. Cluster Strategy: High Availability and Scalability with Industry-Standard Hardware*

[22] B. Brooks and E. Billings, *LoBoS and LoBoS2's Home Page,http://www.lobos.nih.gov/*, August 1997.

[23] Grossfiel, A., *The Wulfpack Home Page, http://wulfpack.med.jhmi.edu/*

[24] Hoffman, F., Hargrove, W., Schultz, A., *The Stone SouperComputer, ORNL's First Beowulf, http://www.esd.ornl.gov/facilities/beowulf/*, May 1999.

[25] *Dell Power Solution, Publicación 3, 2002, Sección: High Performance Computing, Artículo: Design Choices for a Cost-Effective High-Performance Beowulf Cluster, by By Jenwei Hsieh, Ph.D. www.dell.com/powersolutions.*

[26] Jacek Radajewski and Douglas Eadline, *Beowulf Installation and Administration HOWTO*, Versión 0.1.2.2, Junio 1999.

[27] Thomas Sterling y Donald Becker. *Beowulf HOWTO* November 1998.

[28] Thorsten Kukuk. *NIS HOWTO* November 18, 2000.

[29] MPI *www-unix.mcs.anl.gov/mpi/mpich.html*

[30] LAM *www.lam-mpi.org*