Cluster Heterogéneo De Computadoras

EMILIO JOSÉ PLAZA NIETO

30 de diciembre de 2003

Índice general

Índice de figuras

Capítulo 1

INTRODUCCIÓN.

Contents

1.1. Resumen proyecto.

Debido al espectacular auge de la tecnología y de la informática en la actualidad los equipos informáticos se quedan anticuados en un plazo corto de tiempo. Para aprovechar estos "viejos" equipos se puede construir un cluster[1], de forma que la capacidad de computación que obtengamos pueda llegar a compensarnos ante la inversión a realizar en equipos más potentes. Programas con gran carga de computación tardarían bastante tiempo en realizarse en uno de estos "viejos" ordenadores. Se pretende usar estos ordenadores para construir un cluster y con él reducir este tiempo de computación mediante el reparto de la carga entre sus nodos.

Este proyecto se basa en el desarrollo de un cluster de computadoras heterogéneo formado por un front-end[2] y veinte y tres nodos.

La realización de este cluster de computadoras heterogéneo estará compuesta de dos fases, una de configuración hardware y otra de desarrollo software. La configuración hardware estará a su vez dividida en tres partes, configuración del equipo central o front-end, configuración de los respectivos nodos que formarán parte del cluster y configuración de la red de interconexión.

La configuración software consistirá en la realización de programas para control de latencia de red y ancho de banda de la misma, además de un programa para obtener el incremento de rendimiento o ganancia de velocidad que se consigue añadiendo nuevos nodos a la resolución del problema.

[1] conjunto de computadoras interconectadas con dispositivos de alta velocidad que actúan en conjunto usando el poder cómputo de varios CPU en combinación para resolver ciertos problemas dados

[2] ordenador central o servidor encargado de gestionar los nodos y operaciones que se realizan en el cluster

Además, se instalarán programas que nos permitirán observar los detalles de utilización de cada nodo: números de usuarios conectados al nodo, memoria libre, memoria swap disponible, etc.

1.2. Memoria descriptiva.

Instalación mínima del sistema operativo en cada uno de los nodos, es decir, se instalará los servicios mínimos para realizar un procesamiento paralelo y servicios de acceso para que el sistema funcione correctamente. Además se recompilará el núcleo del sistema operativo.

Configuración de la red de interconexión entre los nodos y el ordenador central o front-end a través del protocolo DHCP (Dinamic Host Configure Protocol) de forma que el ordenador central 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.

Configuración de un firewall en la máquina central o front-end que permita únicamente el acceso desde el exterior a éste a través del protocolo SSH Secure SHell.

Los nodos añadidos se configurarán de forma que puedan arrancar vía NFS (Network File System) a través del protocolo DHCP, obteniendo su dirección IP mediante el protocolo DHCP cuando el nodo arranque de forma local.

Instalación del software PVM (Parallel Virutal 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 mensajes entre los distintos nodos del cluster.

Instalación y configuración del software LAM/MPI (Local Area Multicomputer/Message Passing Interface) con el mismo propósito que PVM. Se realizarán comparativas entre ambos paquetes a través de la implementación de programas.

Instalación de herramientas software que nos permitan observar el estado del cluster.

1.3. Planificación.

Se realizará un estudio sobre qué distribución Linux se ajusta mejor a nuestras necesidades en función de los ordenadores que disponemos para realizar el cluster.

Instalación mínima distribución Linux en el front-end y recompilación del núcleo.

Configuración protocolo DHCP.

Configuración de un firewall en la máquina central o front-end.

Instalación y configuración de PVM y XPVM. Desarrollo de programas en PVM.

Instalación y configuración de LAM/MPI. Desarrollo de programas en LAM/MPI.

Instalación de herramientas software para conocer el estado del ordenador central y los restantes nodos.

1.4. Introducción a los cluster de computadoras.

1.4.1. ¿Que es un cluster de computadoras?

Un cluster es un grupo de equipos independientes que ejecutan una serie de aplicaciones de forma conjunta y aparecen ante clientes y aplicaciones como un solo sistema. Los clusters permiten aumentar la escalabilidad, disponibilidad y fiabilidad de múltiples niveles de red.

La escalabilidad es la capacidad de un equipo para hacer frente a volúmenes de trabajo cada vez mayores sin, por ello, dejar de prestar un nivel de rendimiento aceptable. Existen dos tipos de escalabilidad:

- Escalabilidad del hardware (también denominada «escalamiento vertical»). Se basa en la utilización de un gran equipo cuya capacidad se aumenta a medida que lo exige la carga de trabajo existente.
- Escalabilidad del software (también denominada «escalamiento horizontal»). Se basa, en cambio, en la utilización de un cluster compuesto de varios equipos de mediana potencia que funcionan en tándem de forma muy parecida a como lo hacen las unidades de un RAID (Redundant Array of Inexpensive Disks o Array redundante de discos de bajo coste). Se utilizan el término RAC (Redundant Array of Computers o Array redundante de equipos) para referirse a los clusters de escalamiento horizontal. Del mismo modo que se añaden discos a un array RAID para aumentar su rendimiento, se pueden añadir nodos a un cluster para aumentar también su rendimiento.

La disponibilidad y la fiabilidad son dos conceptos que, si bien se encuentran íntimamente relacionados, difieren ligeramente. La disponibilidad es la calidad de estar presente, listo para su uso, a mano, accesible; mientras que la fiabilidad es la probabilidad de un funcionamiento correcto.

Pero hasta el más fiable de los equipos acaba fallando. Los fabricantes de hardware intentan anticiparse a los fallos aplicando la redundancia en áreas clave como son las unidades de disco, las fuentes de alimentación, las controladoras de red y los ventiladores, pero dicha redundancia no protege a los usuarios de los fallos de las aplicaciones. De poco servirá, por lo tanto, que un servidor sea fiable si el software de base de datos que se ejecuta en dicho servidor falla, ya que el resultado no será otro que la ausencia de disponibilidad. Ésa es la razón de que un solo equipo no pueda ofrecer los niveles de escalabilidad, disponibilidad y fiabilidad necesarios que sí ofrece un cluster.

Vemos cómo los clusters imitan a los arrays RAID al aumentar el nivel de disponibilidad y fiabilidad. En las configuraciones de discos tolerantes a fallos, como RAID 1 o RAID 5, todos los discos funcionan conjuntamente formando un array redundante de modo que cuando uno de ellos falla, sólo hay que reemplazarlo por otro; el resto del array sigue funcionando sin problemas, sin necesidad de que se efectúen tareas de configuración y, lo que es más importante, sin que se produzcan tiempos muertos. En efecto, el sistema RAID reconstruye automáticamente la unidad nueva para que funcione conjuntamente con las restantes. De igual modo, cuando falla un equipo que forma parte de un cluster, sólo hay que sustituirlo por otro. Algunos programas de cluster incluso configuran e integran el servidor de forma automática en el cluster, y todo ello sin que el cluster deje de estar disponible ni un solo instante.

En definitiva, un cluster es un conjunto de computadoras interconectadas con dispositivos de alta velocidad que actúan en conjunto usando el poder cómputo de varios CPU en combinación para resolver ciertos problemas dados.

Se usa un cluster con varios computadores para crear un supercomputador.

Hoy día los supercomputadores son equipos excesivamente costosos que están fuera del alcance de empresas o instituciones pequeñas. Un cluster, siendo una combinación de equipos microcomputadores (IBM PC Compatibles), puede ser instalado inclusive por particulares y puede ofrecer rendimiento muy cercano a un SuperComputador en cuanto a poder de cómputo.

En pocas palabras imagínate unos 20 PCs Pentium II ó III de 500 Mhz que actúan en conjunto como si fuese un sólo CPU de 10.000 Mhz!!! (Si bien no es tan fácil como eso, sirve para ilustrar algo aproximado a lo que se obtendrá).

El surgimientos de plataformas computacionales de comunicación y procesamiento estándares de bajo costo, les ha brindado la oportunidad a los programadores académicos de crear herramientas computacionales del dominio público o de costo razonable. Estas realidades permiten la implantación de códigos paralelizados sobre este tipo de plataformas obteniendo un rendimiento competitivo en relación a equipos paralelos especializados cuyos costos de operación y mantenimiento son elevados.

Una de las herramientas de más auge en la actualidad son los llamados cluster Beowulf, los cuales presentan diversas capacidades para el cómputo paralelo con un relativo alto rendimiento.

1.4.2. Conceptos generales.

Cluster Beowulf no es un paquete software especial, ni una nueva topología de red, ni un núcleo modificado. Beowulf es una tecnología para agrupar computadores basados en el sistema operativo Linux para formar un supercomputador virtual paralelo. En 1994 bajo el patrocinio del proyecto ESS del Centro de la Excelencia en Ciencias de los Datos y de la Información del Espacio (CESDIS), Thomas Sterling y Don Becker crearon el primer cluster Beowulf con fines de investigación.

Beowulf posee una arquitectura basada en multicomputadores el cual puede ser utilizado para la computación paralela. Este sistema consiste de un nodo maestro y uno o más nodos esclavos conectados a través de una red Ethernet u otra topología de red. Esta construido con componentes hardware comunes en el mercado, similar a cualquier PC capaz de ejecutar Linux, adaptadores de Ethernet y switches estándares. Como no contiene elementos especiales, es totalmente reproducible. Una de las diferencias principales entre Beowulf y un cluster de estaciones de trabajo (COW, Cluster Of Workstations) es el hecho de que Beowulf se comporta más como una sola máquina que como muchas estaciones de trabajo conectadas. En la mayoría de los casos los nodos esclavos no tienen monitores o teclados y son accedidos solamente vía remota o por terminal serie. El nodo maestro controla el cluster entero y presta servicios de sistemas de archivos a los nodos esclavos. Es también la consola del cluster y la conexión hacia el exterior. Las máquinas grandes de Beowulf pueden tener más de un nodo maestro, y otros nodos dedicados a diversas tareas específicas, como por ejemplo, consolas o estaciones de supervisión. En la mayoría de los casos los nodos esclavos de un sistema Beowulf son estaciones simples. Los nodos son configurados y controlados por el nodo maestro, y hacen solamente lo que éste le indique. En una configuración de esclavos sin disco duro, estos incluso no saben su dirección IP hasta que el maestro les dice cuál es.

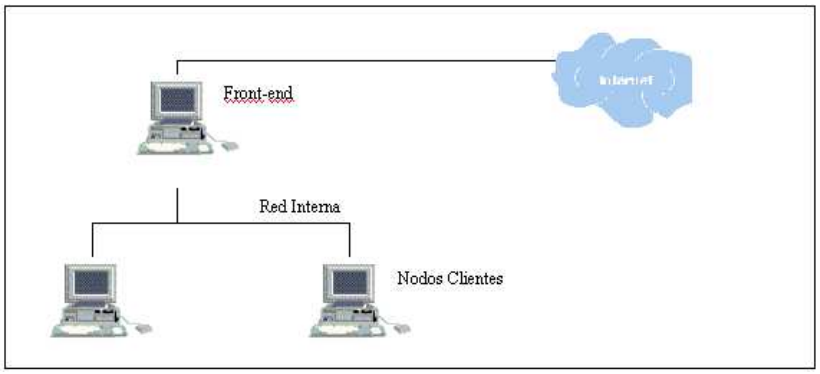

Figura 1.1: Arquitectura genérica de un cluster Beowulf

La topología de red recomendada es un Bus, debido a la facilidad para proporcionar escalabilidad a la hora de agregar nuevos nodos al cluster. Protocolos como Ethernet, Fast Ethernet, GigaEthernet, 10/100 Mbps Switched Ethernet, etc, son tecnologías apropiadas para ser utilizadas en Beowulf.

Beowulf utiliza como sistema operativo cualquier distribución Linux. Además usa bibliotecas de paso de mensajes como PVM y MPI.

Sin lugar a duda los cluster presenta una alternativa importante para varios problemas particulares, no solo por su economía, sino también porque pueden ser diseñados y ajustados para aplicaciones específicas.

1.4.3. Clasificación.

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

- Clase I. Son sistemas compuestos por máquinas cuyos componentes cumplen con la prueba de certificación "Computer Shopper" lo que significa que sus elementos son de uso común, y pueden ser adquiridos muy fácilmente en cualquier tienda distribuidora. De esta manera, estos clusters no están diseñados para ningún uso ni requerimientos en particular.
- Clase II. Son sistemas compuestos por máquinas cuyos componentes no pasan la prueba de certificación "Computer Shopper" lo que significa que sus componentes no son de uso común y por tanto no pueden encontrarse con la misma facilidad que los componentes de sistemas de la clase anterior. De tal manera, pueden estar diseñados para algún uso o requerimiento en particular. Las máquinas ubicadas en esta categoría pueden presentar un nivel de prestaciones superior a las de la clase I.

Capítulo 2

INSTALACIÓN DE LINUX EN EL FRONT-END.

Contents

2.1. Creación de particiones.

Se crearán tres particiones una de 16MB para MS-DOS, una de 128MB para swap y el espacio restante será una partición de Linux native.

Para crear la partición de MS-DOS utilizaremos un disco de arranque de windows98 y con la herramienta **fdisk** crearemos una única partición primaria de 16MB. Posteriormente se formateará dicha partición y copiaremos los archivos del sistema para que esta sea arrancable.

Para crear la partición de swap y Linux native utilizaremos utilizaremos al igual que el caso anterior la herramienta fdisk pero en este caso dicha herramienta será la que ofrece el proceso de instalación. Los comandos más importantes son:

- **m:** muestra la ayuda.
- **q:** sale sin grabar cambios.
- **w:** sale guardando los cambios.
- **p:** muestra las particiones definidas en el disco (si se ha creado una nueva partición, este comando también la mostrará, pero no existe en el disco hasta que no se guarde con el comando w.)
- **v:** te enseña los sectores que no estan asigandos a particiones.
- **d:** borra una partición.

- **n:** crea una partición (habrá que tener espacio para ello). Lo primero que se tiene que decir es si es primaria o extendida. Después solicitará cual va a ser el inicio de la partición enseñándote dos números: se pone el más pequeño, por ultimo solicita el final de la partición. Si se desea ocupar todo el el espacio libre se pone el número que ofrece, también se puede dar un tamaño determinado, p.ej. una partición de 500MB se pondría +500M, con +1500K se crea una de 1.5MB.

- **t:** cambia el tipo de partición. Solicita la introducción del número de partición y después se introduce el código correspondiente al tipo de partición (83 linux native, 82 swap).

- **a:** pone/quita el flag de arrancable.

2.2. Creación disquete para instalación.

Si el computador no puede arrancar directamente desde el CD-ROM se tendrá que crear un disquete para poder comenzar con la instalación del sistema operativo. Esta situación suele pasar con los CD-ROM SCSI.

Para realizar el disquete bajo MS-DOS, usaremos la uitilidad **rawrite** incluida en el CD-ROM de Red Hat Linux en el directorio **dosutils**. Será necesario introducir en la disquetera un disquete en blanco formateado a 1.44MB.

Pasos a seguir para la creación de dicho disquete:

- Iniciar nuestro sistema con un disquete de arranque de windows 98 que disponga de los drivers para dispositivos SCSI.

- Finalizada la fase de arrancado nos desplazaremos a la unidad asignada al CD-ROM durante esta fase (puede ser F, G, H, I, etc, en función de la unidades que tengamos en nuestro sistema).

- Una vez dentro de la unidad asignada al CD-ROM nos desplazaremos al directorio denominado dosutils y ejecutaremos rawrite. Esta utilidad nos preguntará primeramente por el nombre del fichero que contiene la imagen de arranque (p.ej. *..\images\boot.img*). Seguidamente se nos preguntará la unidad en la cual se va a grabar la imagen, generalmente la unidad **a**. Finalmente se nos preguntará si la unidad seleccionada es correcta, pulsaremos **Enter** para confirmar. En este paso tendremos que cerciorarnos de tener introducido un disquete formateado en la unidad seleccionada. Una vez presionada la tecla de enter la uitilidad rawrite copiará la imagen de arranque al disquete.

Ejemplo de los pasos a seguir para realizar un disquete para la instalación una vez que se ha arrancado con el disquete de windows 98 y se han cargado los drivers para CD-ROM SCSI.

*C:\>**d:***
*D:\>**cd \dosutils***
*D:\dosutils>**rawrite**>*
*Enter disk image source file name: **..\images\boot.img***
*Enter target diskette drive: **a:***
*Please insert a formatted diskette into drive A:and press –ENTER–:**Enter***
D:\dosutils>

Una vez realizado los pasos anteriores reiniciaremos el equipo con el disquete creado, introducido en la unidad de disco y el CD-ROM de Red Hat Linux introducido en el CD-ROM

2.3. Selección modo de instalación.

Al iniciarse el disquete de instalación aparecerá el siguiente menú en el cual deberemos seleccionar el método de instalación.

Welcome to Red Hat 6.2

- To instal or upgrade Red Hat Linux in **graphical mode** press the <**ENTER**> key
- To instal or upgrade Red Hat Linux in **text mode**, type: **text** <**ENTER**>
- To enable **low resolution mode**, type: **lowres**<**ENTER**>
  press <**F2**> for more information about low resolution mode
- To disable **framebuffer mode**, type: **nofb** <**ENTER**>
  press <**F2**> for more information about framebuffer mode
- To enable **expert mode**, type: **expert** <**ENTER**>
  press <**F3**> for more information about expert mode
- To enable **rescue mode**, type: **linux rescue** <**ENTER**>
  press <**F5**> for more information about rescue mode
- If you have a **driver disk**, type: **linux dd** <**ENTER**>
  Use the function keys listed below for more information

[**F1 - Main**][**F2 - General**][**F3 - Expert**][**F4 - Kernel**][**F5 - Rescue**]

$>boot:
Se seleccionará el modo de instalación grafico, es decir, se pulsará la tecla **enter**.

2.4. Pasos a seguir para realizar la instalación de Linux en el front-end.

A continuación se muestran los pasos seguidos para realizar dicha instalación. Usando el ratón se seleccionará el lenguaje que deseamos para la instalación, en nuestro caso el lenguaje seleccionado será el español.

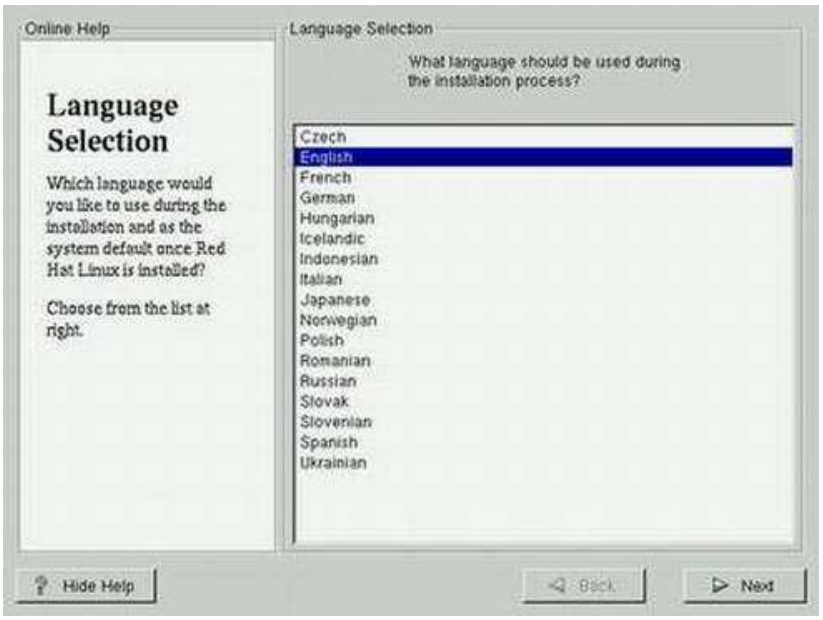

Figura 2.1: Selección del lenguaje

Seguidamente se pasará a configurar el teclado, seleccionando el teclado español genérico de 104 teclas además se activan las teclas muertas. Para asegurarnos del correcto funcionamiento de la configuración elegida la testearemos en el recuadro que existe para ello.

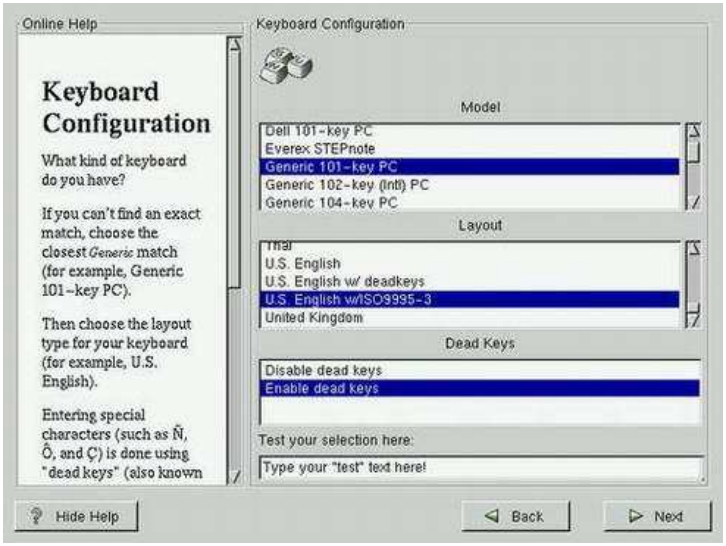

Figura 2.2: Configuración del teclado

El siguiente paso a realizar es la configuración del ratón, si no se encuentra el modelo correcto que se dispone, se puede elegir un tipo de ratón del cual se este seguro que es compatible con nuestro sistema. Si tampoco se encuentra alguno compatible se seleccionará uno genérico, indicando el número de botones y la interfaz.

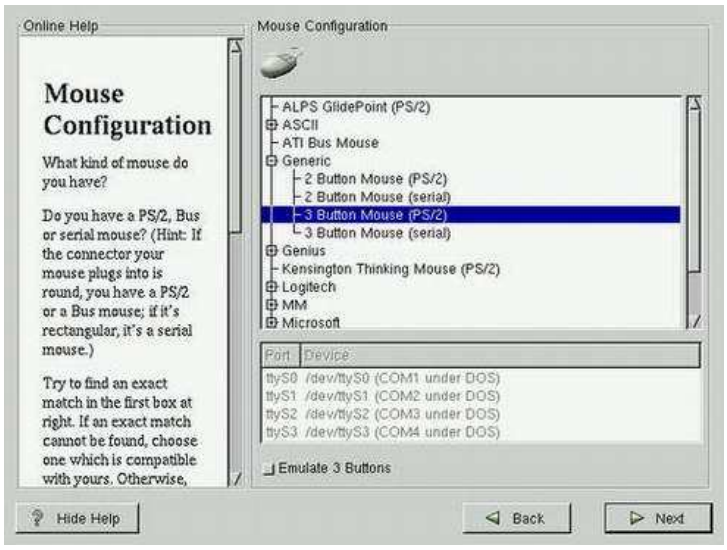

Figura 2.3: Configuración del ratón

Figura 2.4: Bienvenido a Red Hat Linux

A continuación se determinará el tipo de instalación que se va a realizar. Se seleccionará la opción de *custom o personalizada*.

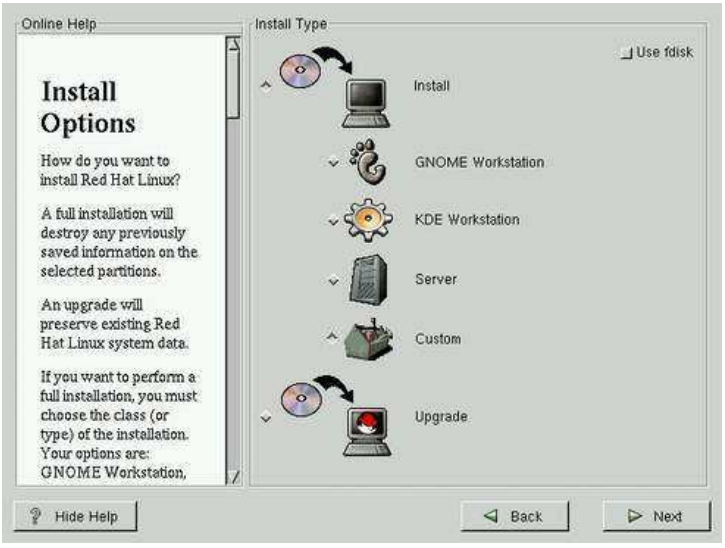

Figura 2.5: Opciones de instalación

El siguiente paso a realizar es la asignación de los puntos de montaje a cada una de las particiones que se han creado. Para llevar a cabo esta operación se marcará la partición *Linux native* creada y se pulsará el botón *Edit*. La partición creada para MS-DOS aparecerá de tipo FAT 32.

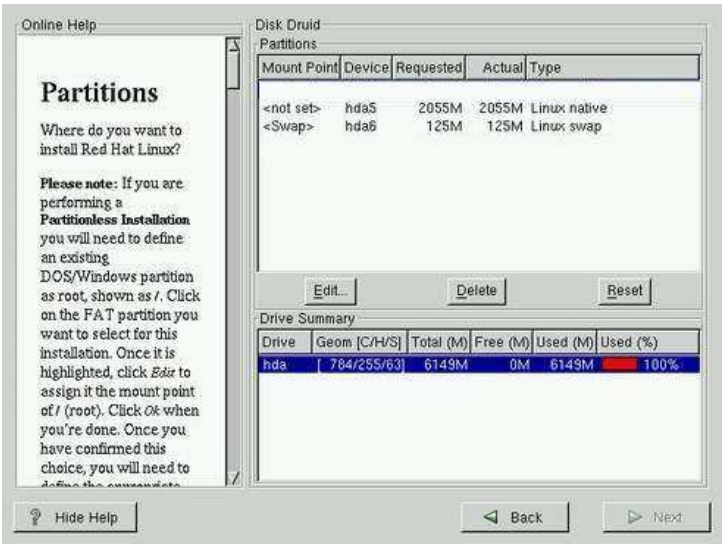

Figura 2.6: Particiones

Al pulsar el botón edit nos aparece el menú que se muestra a continuación, en el cual tendremos que decir que el punto de montaje va ha ser / seguidamente pulsaremos el botón ok.

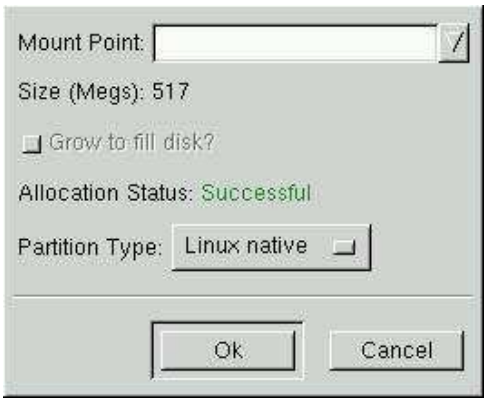

Figura 2.7: Punto de montaje

Seguidamente seleccionaremos la partición o particiones que deseamos formatear. También se nos permite chequear la partición seleccionada para determinar sectores dañados de las particiones.

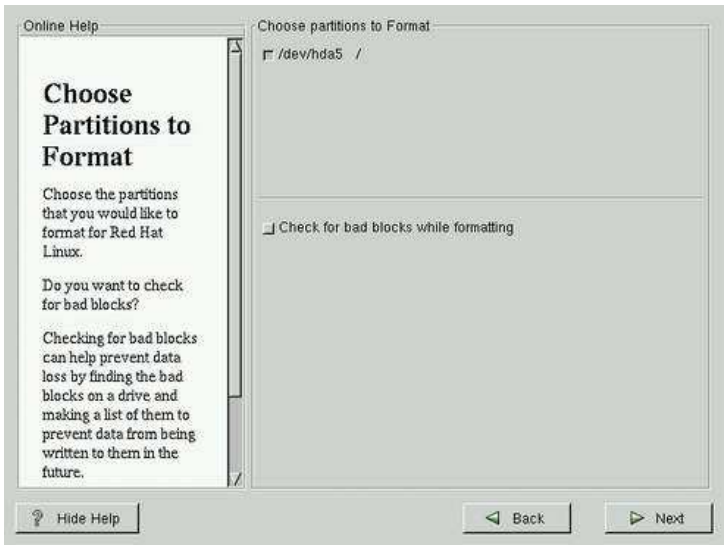

Figura 2.8: Particiones a formatear y chequear

Con el fin de instalar el gestor de arranque que nos permita la selección de los distintos sistemas operativos. Red Hat Linux nos proporciona *LILO* (Linux Loader, el cargado de linux, todas las distribuciones de linux lo posee). Se puede instalar LILO en cualesquiera de los siguientes sitios:

1. El registro maestro de arranque (MBR). Es el sitio recomendado para instalar LILO, a no ser que otro cargador del sistema operativo (System Comander o OS/2 Boot Manager) este ya instalado. Cuando la máquina comienza con el proceso de arrancado, se inicia LILO y se presenta el indicador *boot:*, se puede determinar que sistema operativo se desea iniciar, siempre que este configurado para su arranque ya sea Linux, Windows, etc.

2. El primer sector de la partición raíz. Se recomienda si se tiene instalado un cargador en

tu sistema (como el OS/2 Boot Manager). Se puede configurar el cargador para lanzar a LILO y arrancar Red Hat Linux.

3. El primer sector de un disquete. Es una alternativa a las dos opciones anteriores, para iniciar Red Hat Linux, se inserta el disquete LILO en la disquetera del sistema y se reinicia de nuevo. LILO se inicia desde el disquete y presenta el indicador *boot:*.

El sitio en el cual generalmente se instala LILO es en el MBR, se marcará el sistema operativo por defecto y se indicará si se desea crear un disco de arranque o no.

Si no se desea instalar LILO se deberá marcar la casilla de no instalación de LILO.

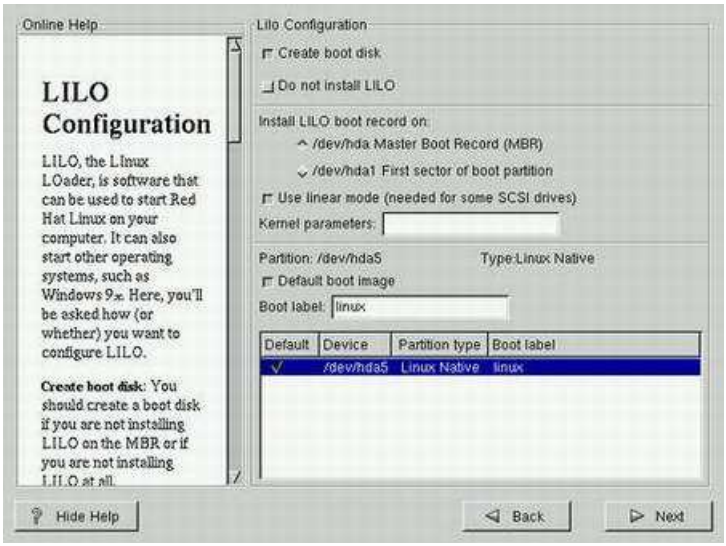

Figura 2.9: Configuracion lilo

En el caso de que no se desee usar LILO para arrancar el sistema Red Hat Linux, se podrá utilizar LOADLIN que puede cargar Linux desde MS-DOS, desafortunadamente, requiere disponer de una copia del núcleo de Linux accesible desde una partición MS-DOS. LOADLIN esta disponible en **ftp://sunsite.unc.edu/system/Linux-boot/**.

Pasos a realizar para la configuración de LOADLIN

- En la partición creada de MS-DOS se deberá crear un directorio llamado loadlin
- Dentro de este directorio se copiará el archivo LOADLIN.EXE descargado de la dirección FTP dada anteriormente. Además copiaremos la imagen del núcleo de Linux. Esta imagen del núcleo la podemos encontrar en nuestro sistema de Red Hat Linux en la siguiente ruta: */usr/src/linux/arch/i386/boot/bzImage*.
- Además se tendrá que modificar los archivos del sistema *Autoexec.bat* y *Config.sys*.

A continuación mostraremos ejemplos de cómo quedaría dichos archivos:

Ejemplo archivo Autoexec.bat

goto
%config%
:Linux
cd loadlin
linux
goto fin

:Win98
SET BLASTER=A220 I7 D1 H7 P330 T6
SET SBPCI=C:\SBPCI
mode con codepage prepare=((850) C:\WINDOWS\COMMAND\ega.cpi)
mode con codepage select=850
keyb sp,,C:\WINDOWS\COMMAND\keyboard.sys
:Fin

Ejemplo archivo Config.sys

[menu] menuitem=Linux,Linux
menuitem=Win98,Win98
menudefault=Linux,25
[Linux] [Win98] DEVICE=C:\WINDOWS\HIMEM.SYS
DEVICE=C:\WINDOWS\EMM386.EXE
device=C:\WINDOWS\COMMAND\display.sys con=(ega,,1)
Country=034,850,C:\WINDOWS\COMMAND\country.sys [commom]

- Se generará el archivo LINUX.BAT con la siguiente estructura:

rem Sample Dos batch file to boot Linux.
rem First, ensure any unwritten disk buffers are flushed smartdrv /C
rem Start the LOADLIN process:
c:\loadlin\loadlin c:\loadlin\bzImage root=/dev/hda4 ro mem=128M
A continuación se realiza una aclaración de la configuración anterior:

- **c:\loadlin\loadlin** ⇒ Ruta del fichero LOADLIN.EXE
- **c:\loadlin\bzImage** ⇒ Imagen del núcleo de nuestro sistema
- **root=/dev/hda4** ⇒ Partición donde se encuentra montada la partición /
- **mem=128M** ⇒ Tamaño de memoria física que dispone nuestro equipo

Una vez configurado el método de arranque de nuestro sistema se pasará a configurar las tarjetas de red. Recordemos que el front-end dispone de dos Ethernet, una para la red externa y otra para la red interna.

La eth0 tendrá la siguiente información: (Dichos datos serán suministrados por el proveedor de internet)

Dirección IP:
Mascara de red:

Direccion de red:
Broadcast:
Gateway:
DNS primario:

La eth1 tendrá la siguiente información:

Dirección IP: 192.168.1.1
Mascara de red: 255.255.255.0
Direccion de red: 192.168.1.0
Broadcast: 192.168.1.255

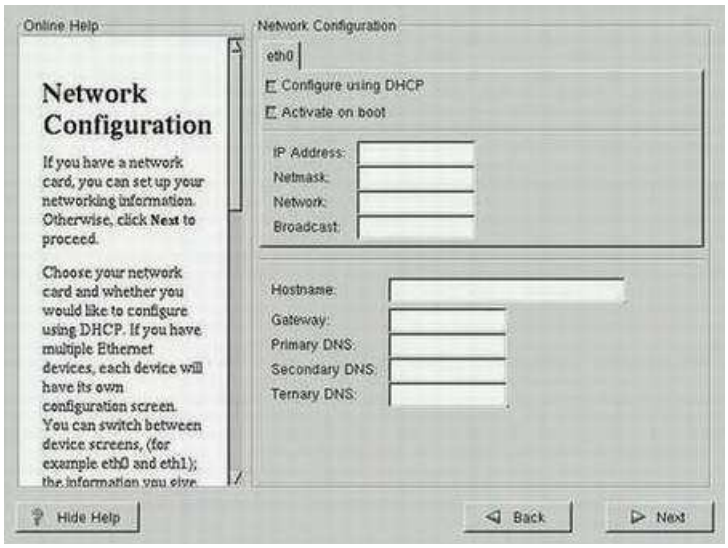

Figura 2.10: Configuracion de la red

El programa de instalación nos presentará un formulario que nos ayudará a configurar la franja horaria de nuestro sistema Red Hat Linux. Se seleccionará para el caso de eoncontrarnos en la península española la opción Europe/Mainland - Madrid.

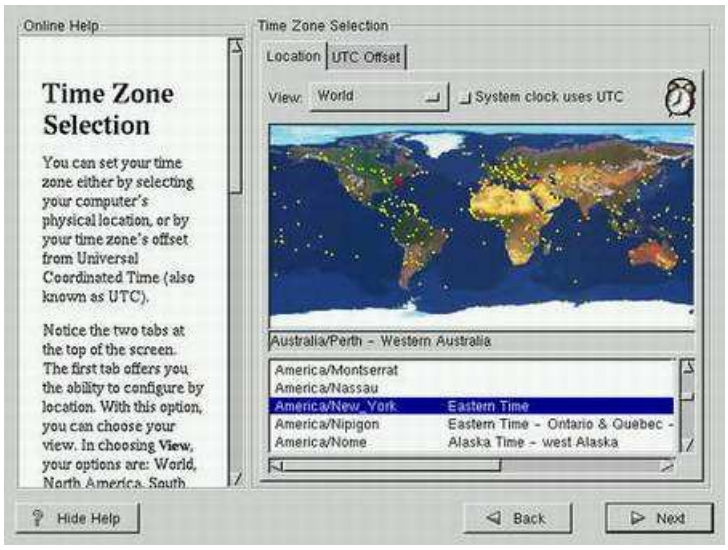

Figura 2.11: Selección franja horaria

Se puede cambiar la configuración de tu zona horaria después de arrancar el sistema usando el comando */usr/sbin/timeconfig*.

Después de haber fijado la granja horaria del sistema, el programa de instalación solicitará la introducción de una clave root al sistema, esta es la clave que se usará para acceder por primera vez al sistema Red Hat Linux.

La clave de superusuario (root) debe tener una longitud de al menos 6 caracteres; cuando se teclea no se muestra por pantalla. Se debe introducir la clave dos veces; si las dos claves no coinciden, el programa de instalación te pedirá que la introduzcas de nuevo. Se deber escoger como clave de superusuario algo que se pueda recordar, pero que no sea nada que alguien pueda adivinar fácilmente. Tu nombre, tu numero de teléfono, qwerty, password, root, 123456, son ejemplos de malas claves. Las claves buenas mezclan números con letras mayúsculas y minúsculas y no contienen palabras de diccionario: Por ejemplo, Aard387vark or 420BMttNT.

Además de introducir la clave de root crearemos una cuenta de usuario del sistema.

Aviso: El usuario root (también conocido como superusuario) tiene acceso completo a todo el sistema; por esta razón, es mejor acceder como usuario root solo para realizar labores de mantenimiento o administración, de forma que no se cambien, muevan, o borren inadvertidamente ficheros críticos del sistema.

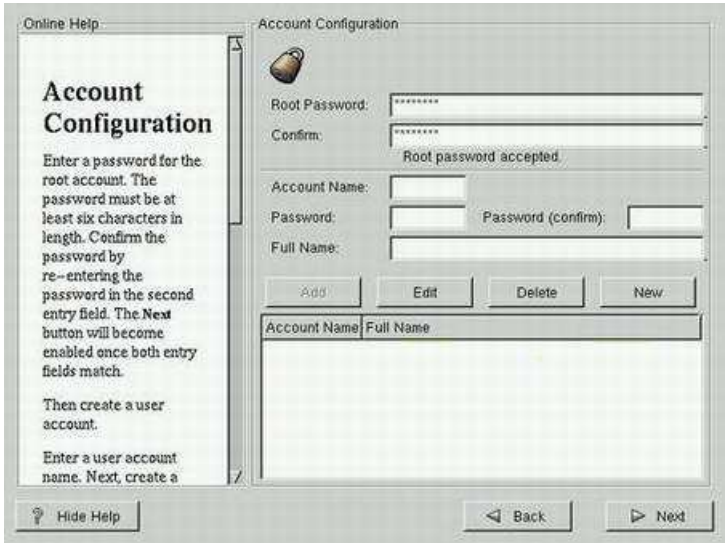

Figura 2.12: Configuración de las cuentas de usuarios

Una vez acabada la configuración de la cuentas de usuario el programa de instalación nos presentará un formulario en el cual sus opciones MD5 y password sombreadas están activadas por defecto, provocando esto que nuestro sistema sea mas seguro.

Esto es debido a que el algoritmo MD5 es una funcion de cifrado tipo hash que acepta una cadena de texto como entrada, y devuelve un número de 128 bits. Las ventajas de este tipo de algoritmos son la imposibilidad(computacional) de reconstruir la cadena original a partir del resultado, y también la imposibilidad de encontrar dos cadenas de texto que generen el mismo resultado.

Esto nos permite usar el algoritmo para transmitir contraseñas a través de un medio inseguro. Simplemente se cifra la contraseña, y se envía de forma cifrada. En el punto de destino, para comprobar si el password es correcto, se cifra de la misma manera y se comparan las formas cifradas.

Además nos permite utilizar password de mayor tamaño que las clásicas de 8 caracteres o menos.

La opción de shadow passwords nos provee un método seguro de almacenaje de password. Las password son guardadas en el fichero /etc/shadow que únicamente puede ser leído por el superusuario.

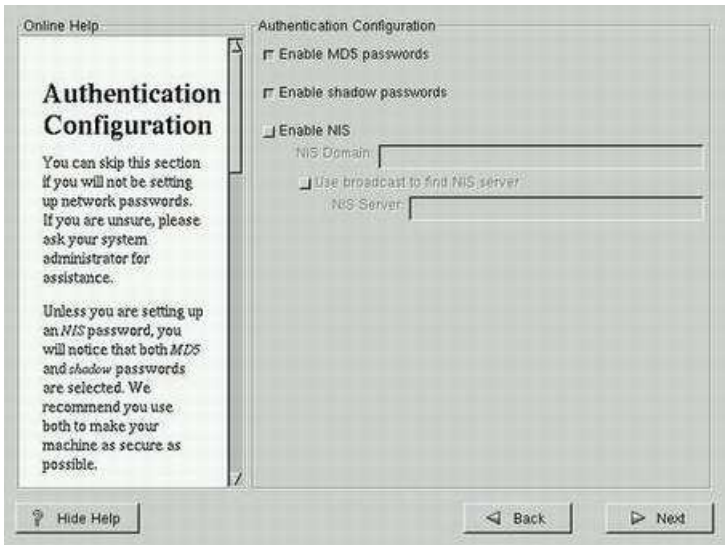

Figura 2.13: Configuración autentificación

Seguidamente se realizará la selección de los paquetes ha instalar. El programa de instalación nos mostrará un menú en el cual se seleccionará la casilla de *kde*, *kernel development* y *utilities*.

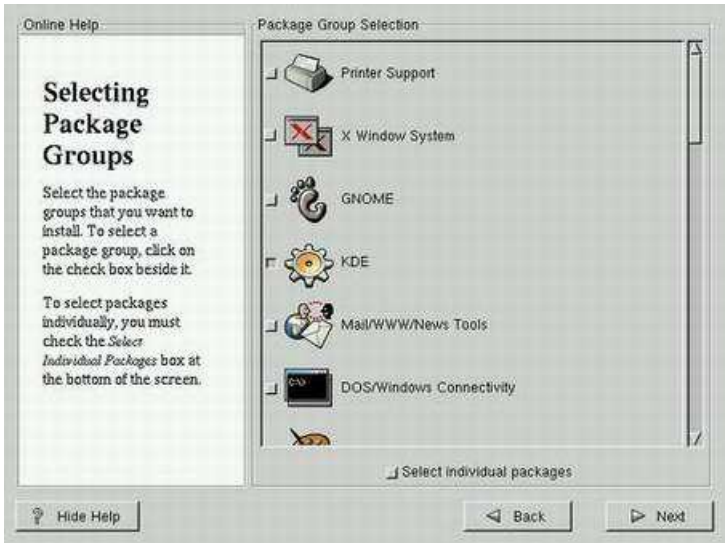

Figura 2.14: Selección grupos de paquetes

A continuación el programa de instalación detectará el tipo de monitor de nuestro sistema. Si el monitor no es detectado o no aparece en la lista se podrá seleccionar uno genérico.

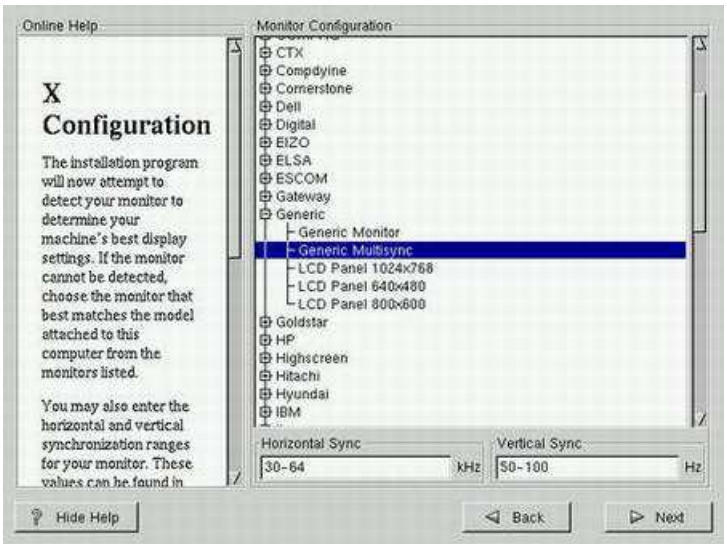

Figura 2.15: Configuración de la X, selección del monitor

Seguidamente configuraremos la tarjeta de video.

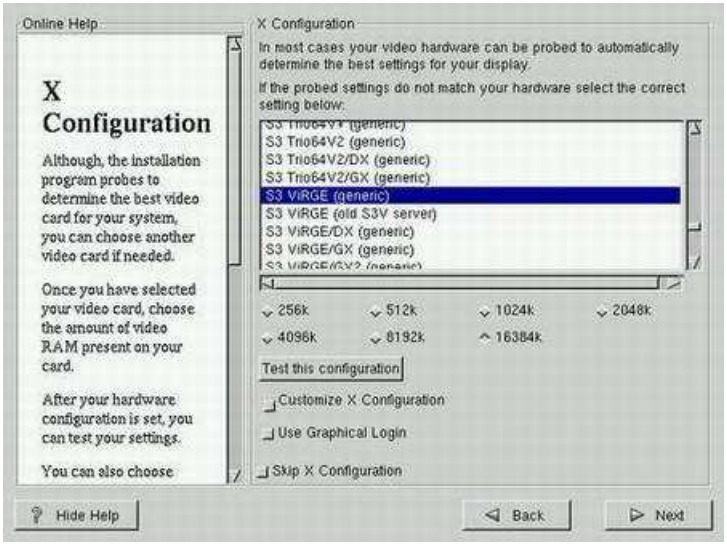

Figura 2.16: Configuración de la X, seleccion de la tarjeta de video

A continuación seleccionaremos las distintas posibilidades de resolución y realizaremos el test para comprobar que nuestra configuración ha sido correcta.

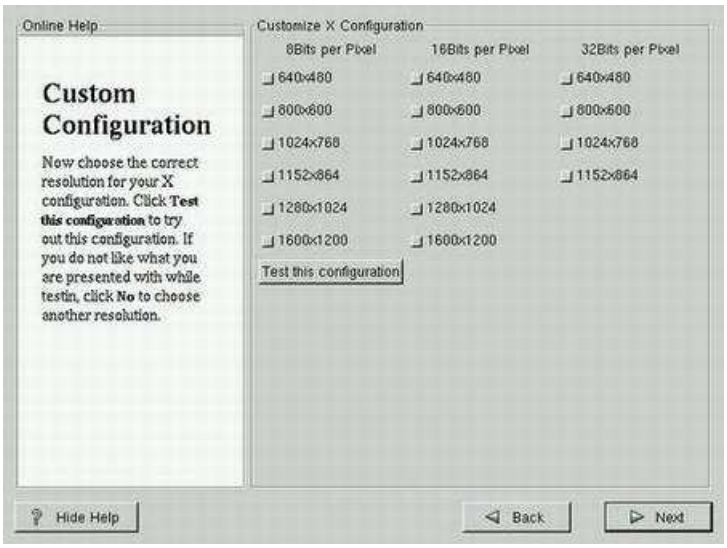

Figura 2.17: Configuración de la resolución de pantalla

El programa de instalación nos mostrará a continuación un menú que nos indicar que se va ha proceder al comienzo de la instalación.

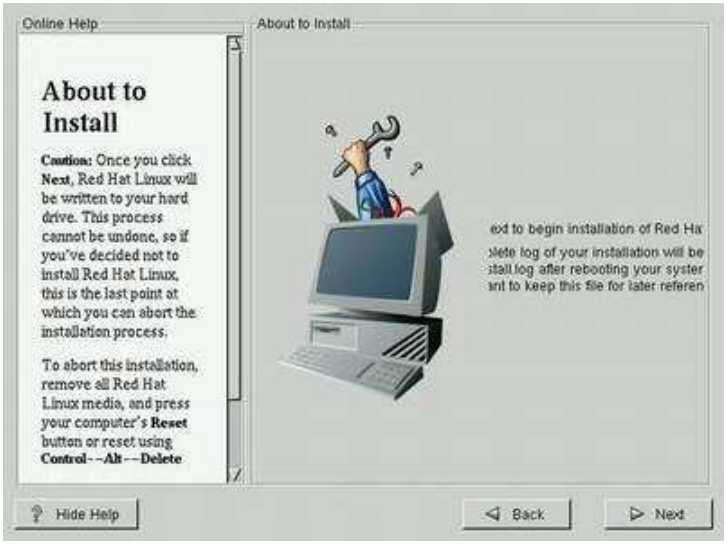

Figura 2.18: Sobre la instalación

Seguidamente nos aparecerá una pantalla en la cual podremos ir observando el estado de la instalación. En este, podremos observar los paquetes que se están instalando, el tiempo total que durará la instalación, el tiempo transcurrido desde que comenzó la instalación y el tiempo restante para su finalización, el tamaño que ocupará la instalación, el tamaño instalado y el restantes.

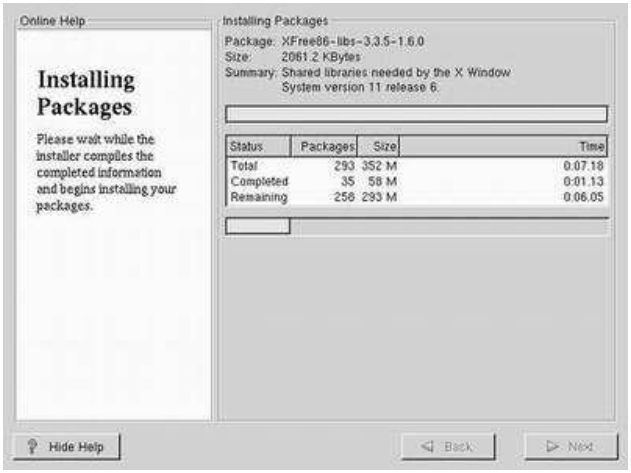

Figura 2.19: Instalación de paquetes

Una vez terminada la instalación, el programa de instalación nos permitir crear un disco de inicio, pudiendo también desechar esta opción pulsando sobre la casilla de *skip boot disk creation*

Figura 2.20: Creación del disco de arranque

Una vez terminada la instalación se nos mostrará una ventana de confirmación de la instalación y que se va a proceder al reiniciado del sistema (para ello pulsaremos el botón de exit) para arrancar por primera vez el sistema instalado.

Figura 2.21: Enhorabuena

2.5. Configuración de los dispositivos de red.

Esta configuración la llevará a cabo el usuario root a través de la herramienta del sistema Network Configurator invocada por el comando:

$>netconf

Una vez ejecutado este comando aparecerá en pantalla el siguiente menú en tcl/tk:

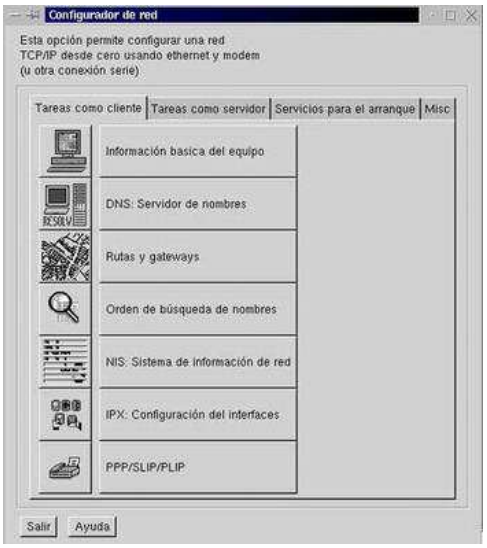

Figura 2.22: Network Configurator

A continuación se muestrán los pasos a seguir para realizar la configuración:

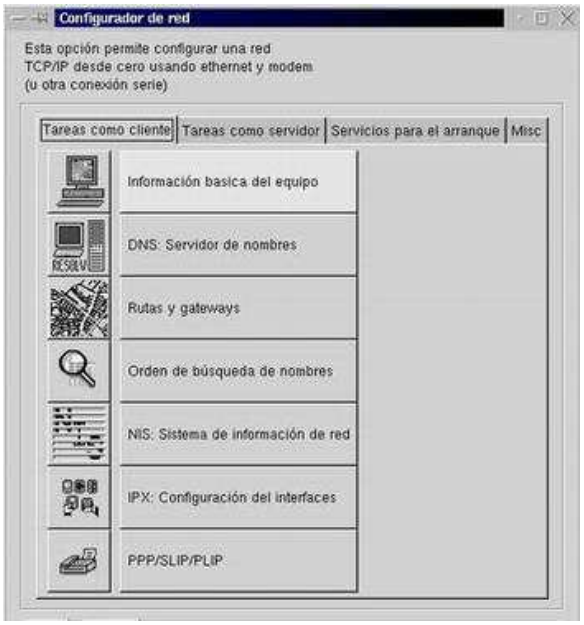

Figura 2.23: Network Configurator. Información básica del equipo

Figura 2.24: Network Configurator. Nombre de la máquina

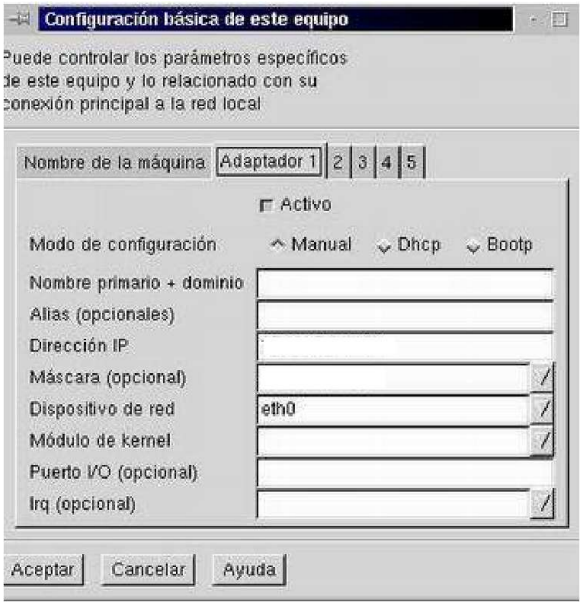

Figura 2.25: Network Configurator. Adaptador Ethernet 1

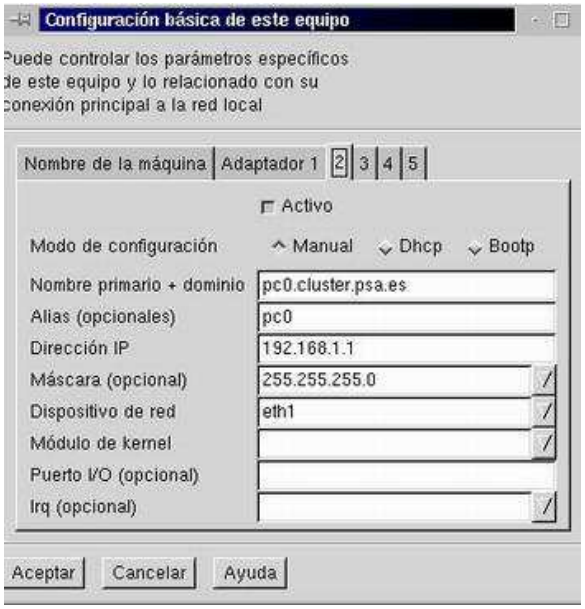

Figura 2.26: Network Configurator. Adaptador Ethernet 2

Figura 2.27: Network Configurator. DNS Servidor de nombres

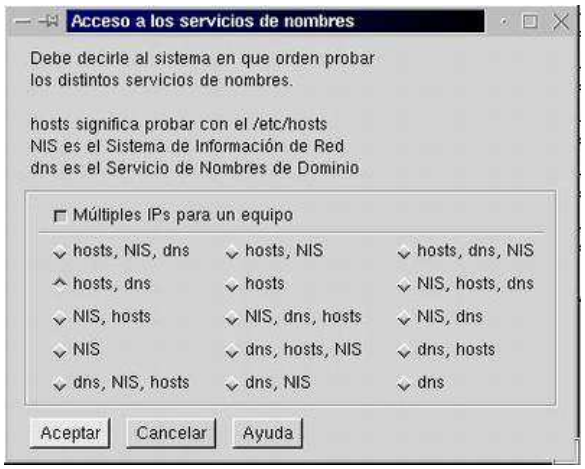

Figura 2.28: Network Configurator. Acceso al servidor de nombres

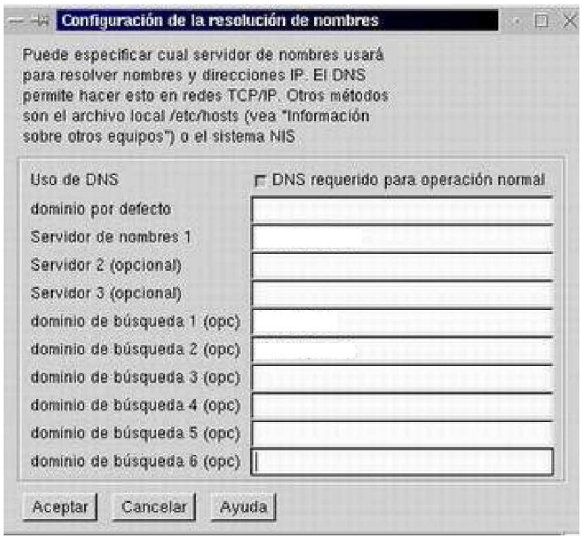

Figura 2.29: Network Configurator. Configuración de la resolución de nombres

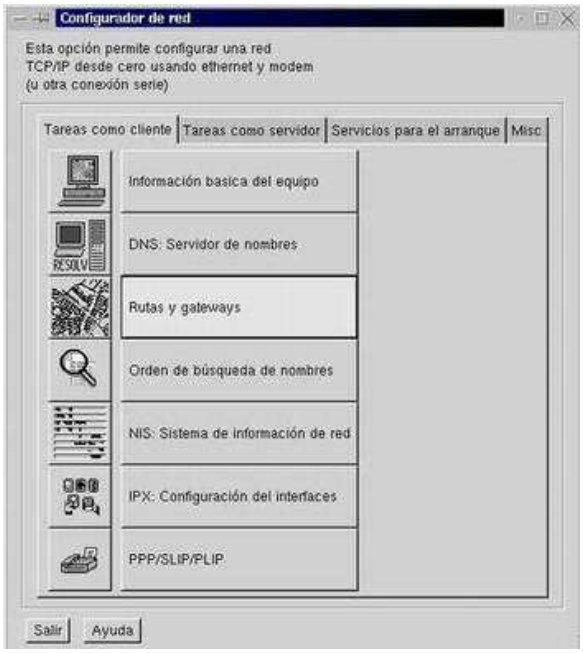

Figura 2.30: Network Configurator. Rutas y gateways

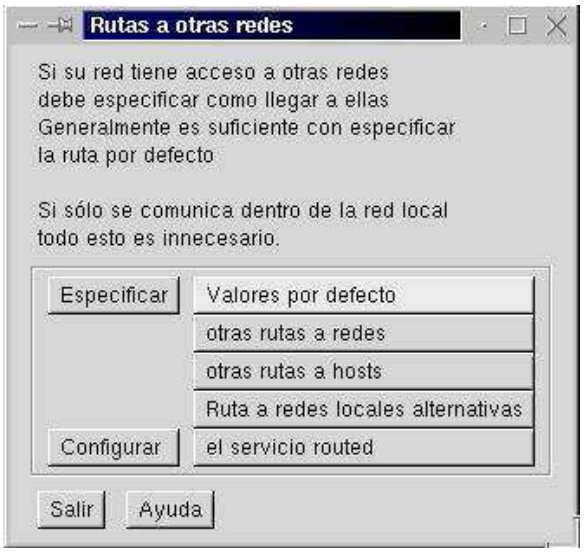

Figura 2.31: Network Configurator. Rutas a otras rutas

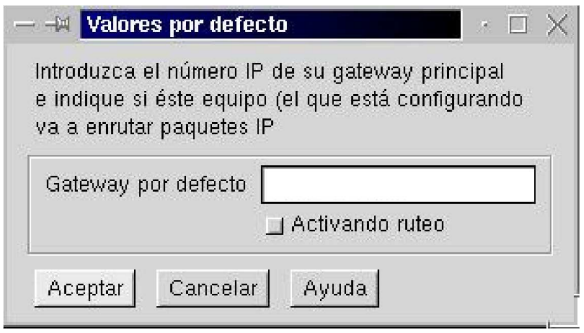

Figura 2.32: Network Configurator. Valores por defecto

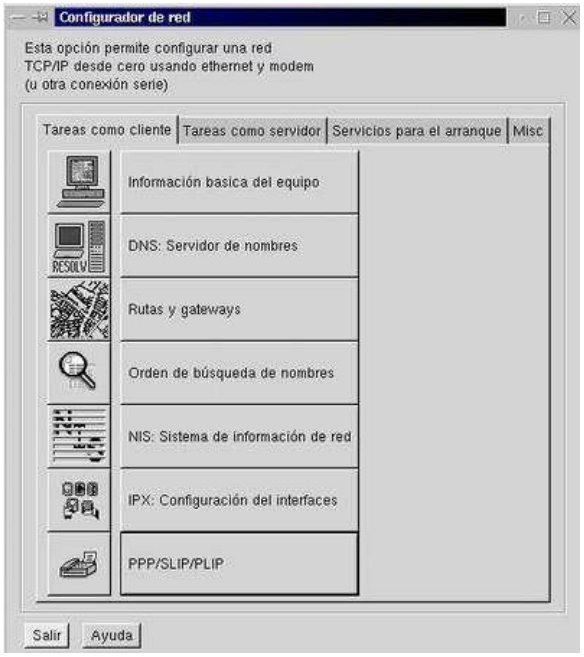

Figura 2.33: Network Configurator. Salir

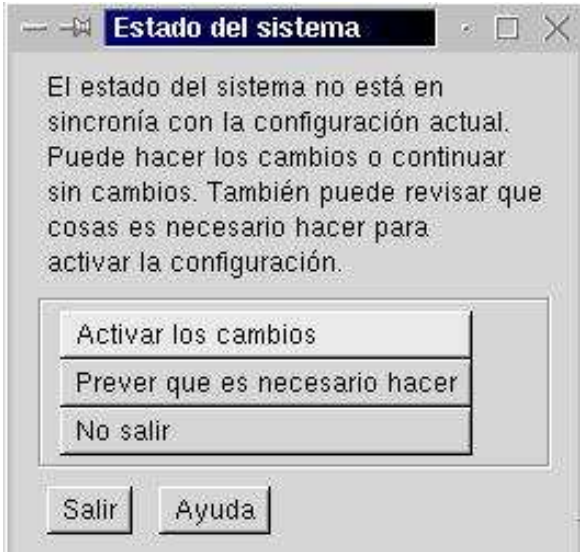

Figura 2.34: Network Configurator. Activar Cambios

Capítulo 3

ARRANQUE SIN DISCO O DISKLESS.

Contents

3.1. Introducción.

Uno de los cambios más destacables del uso de la computadoras en los últimos años ha sido la expansión de la conectividad de red con TCP/IP desde la mesa del 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 más 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.

3.2. Requisitos del protocolo de arranque.

Algunas computadoras sólo necesitan unas cuentas 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 host 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 computador conectado 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 ficheros de configuración, a los que accede el SO en el arranque.

¿Qué ocurre con los computadores sin disco?

Podría guardarse el S.O y el software de red en la ROM de la ethernet, pero esa información no es conocida de antemano por el fabricante ya que define la red a la que se va a conectar el computador.

Disponemos de tres protocolos que nos permitirán transferir esta información por la red:

- **RARP** (Reverse Address Resolution Protocol) sólo proporciona la dirección IP al computador 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 un computador sin disco.

- **DHCP** (Dinamic Host Configuration Protocol) es una extensión de BOOTP, es decir, lo mejora.

3.3. Objetivos del arranque por red.

- Reducir los costes de mantenimiento del software en gran cantidad de máquinas. Con el arranque por red los ficheros son mantenidos en un servidor central, esto conlleva a la ventaja de poder ser actualizados en una sola máquina.
- La posibilidad de conmutar entre sistemas operativos sin tener que cargar el software cada vez que se cambie de un sistema a otro.
- Usar ordenadores en lugares donde los discos duros no son suficientemente resistentes, como podría ser en la planta de una factoría, en la que éstos pueden ser relativamente frágiles.
- Facilita el traspaso de equipos de una persona a otra. Por ejemplo, si en una empresa un equipo pasa de una persona a otra, ésta no tendrá que hacer un traspaso de información sino que, como todo se encuentra en el mismo servidor, bastará con entrar con el nuevo usuario.
- Reducir el coste económico. El tener equipos sin disco duros reduce en gran medida el coste de un equipo.

3.4. Introducción DHCP.

Las siglas DHCP significan Dinamic Host Configuration Protocol. Es utilizado para grandes redes. El daemon actúa dándole información de la red a las estaciones de trabajo, tales como IP Address, Subnet Mask, DNS Server, Gateway, etc.

DHCP ha sido creado por el Grupo de Trabajo Dynamic Host Configuration del IETF (Internet Engineering Task Force, organización de voluntarios que define protocolos para su uso en Internet). Su definición se encuentra en los RFC's 2131, el protocolo DHCP, y el 2132, opciones de DHCP.

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 hosts sobre una red TCP/IP. Este protocolo se encarga de la configuración automática de los parámetros de red, utilizando direcciones.

DHCP es una extensión de BOOTP, es decir, mejora BOOTP, y es compatible con él (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 otros dispositivos.

DHCP puede usarse cuando el número de IPs es menor que el número de computadores y todos no están conectados a la vez, como en un proveedor de servicio de Internet (ISP).

DHCP está formado por dos partes: 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 coge de la pila de direcciones IP disponibles y se la asigna durante un periodo de tiempo negociado.

El servidor admite tres tipos de configuración de direcciones IP:

1. Estática. Se configura en el servidor la dirección de red que se 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 chequea su base de datos estática. Si existe una entrada para esa dirección física, se devuelve la dirección IP estática correspondiente. Si no se 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 67.
  - Aquellos servidores que puedan dar este tipo de servicio responden con 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 de 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.

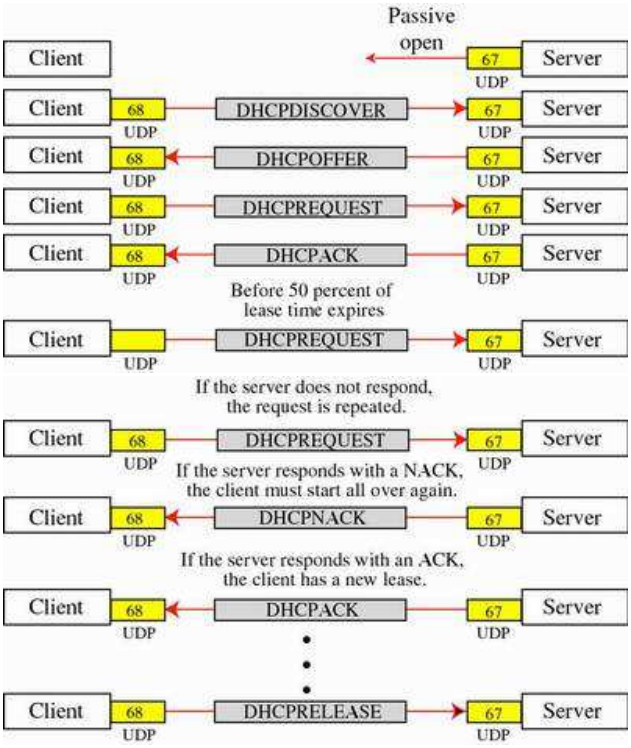

Figura 3.1: Funcionamiento DHCP

- **Formato del paquete**:

  Para hacer DHCP compatible con BOOTP, los diseñadores de DHCP han usado casi el mismo formato de paquete. Solo 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 forzar que la respuesta del servidor sea broadcast en vez de unicast. Si la respuesta es unicast, la dirección destino sera la del cliente y este no la conoce, por lo que puede descartar el mensaje. Al ser broadcast, todos los computadores reciben y procesan el mensaje.

- **Opciones**: Se han añadido varias posibiliades 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 configurará 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 mas 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.

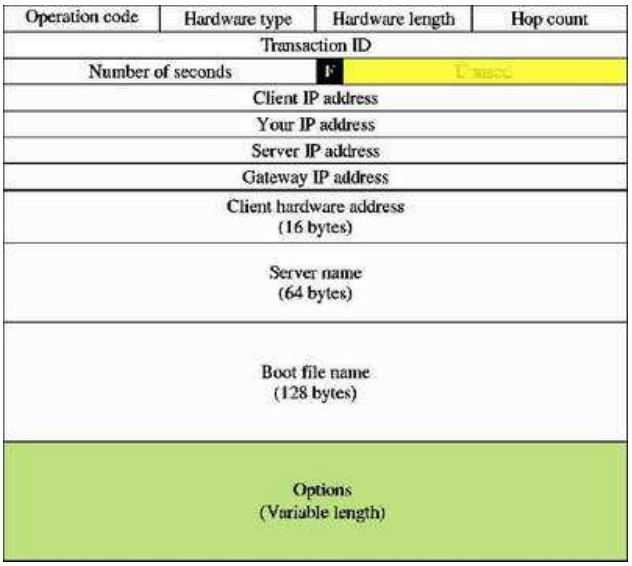

Figura 3.2: Formato opciones

Etiq.	*Long.*	*Valor*
53	1	1 DHCPDISCOVER
		2 DHCPOFFER
		3 DHCPREQUEST
		4 DHCPDECLINE
		5 DHCPACK
		6 DHCPNACK
		7 DHCPRELEASE

Figura 3.3: Opciones

El arrendamiento de direcciones se emplea para racionar las direcciones IP, minimizando el coste 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 sera bajo (ej. 15 minutos); si se conectan/desconectan los clientes con mucha frecuencia e interesa que este 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 esta desconectado en vacaciones para asumir que el cliente ya no esta 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.

3.5. Características de DHCP.

El protocolo de configuración dinámico de host extiende significativamente las posibilidades de BOOTP. Las mejoras mas importantes son:

- Administración más sencilla.
- Configuración automatizada.
- Permite cambios y traslados.
- Posibilidad de que el cliente solicite los valores de ciertos parámetros.
- Nuevos tipos de mensajes de DHCP que soportan interacciones cliente/servidor robustas.

3.6. ¿Qué ventajas me da el DHCPd?

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

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

Para que un ordenador, 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 fichero del que debe arrancar. Estos datos pueden varias dependiendo de las necesidades de cada uno.
- Un sistema de ficheros con el que trabajar.
- La imagen (núcleo del sistema operativo) para realizar el arranque.

Si tenemos una red formada por ordenadores sin disco (DC-Diskeless Computer) teniendo estos una ROM para arrancar por red, se diferenciarán unos de otros gracias a la dirección Ethernet.

Un ejemplo de intercambio sería el siguiente:

> 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 192.168.1.2, tu servidor es 192.168.1.1. En el caso que necesite un fichero 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, el DC debe conseguir la imagen del sistema operativo y lanzarlo a ejecución. En esta fase, se usa otro protocolo TCP/IP, TFTP (Trivial File Transfer Protocol).Éste es una versión reducida del FTP (File Transfer Protocol). El TFTP no contempla autentificación, trabaja a través de UDP (User Datagram Protocol) en vez de TCP (Transmisión Control Protocol).
>
> La implementación de UDP en un ordenador 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 fichero completo para almacenarlo en 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 sido 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 ficheros 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 acabamos de cargar. El sistema operativo debe ser capaz de ejecutarse con un sistema de ficheros raíz NFS, en vez de un disco real.

3.8. Preparación de la máquina servidora para atender peticiones por la red.

3.8.1. Creación del sistema de ficheros para los PC clientes o nodos del cluster.

Los directorios de cada uno de los posibles clientes de nuestro servidor se van a encontrar dentro del directorio **/tftpboot**. La estructura de este directorio tendrá la siguiente forma:

/tftpboot/nombre_máquina_cliente1
/tftpboot/nombre_máquina_cliente2

Dentro de los directorios clientes debemos crear la jerarquía de directorios necesaria para el sistema de ficheros, es decir, tendrá la propia estructura del sistema de ficheros de Linux:

```
/tftpboot/nombre_máquina_cliente/bin
/ "   /dev
/ "   /etc
/ "   /lib
/ "   /sbin
/ "   /var
/ "   /home
/ "   /mnt
/ "   /proc
/ "   /root
/ "   /tmp
/ "   /usr
```

Para crear esta estructura interna a cada directorio cliente tenemos que ejecutar las siguientes órdenes de comandos en el front-end:

$> cd /tftpboot/nombre_máquina_cliente
$> cp -r /bin .
$> cp -ra /dev .
$> cp -ra /etc .
$> cp -r /lib .
$> cp -r /sbin .
$> cp -r /var .
$> mkdir home
$> mkdir mnt
$> mkdir proc
$> mkdir root
$> mkdir tmp; tendrá permiso temporal ⇒ chmod 0177
$> mkdir usr

Los directorios home, mnt, proc, root, tmp, es decir, los que solo creamos, serán montados desde el frontend a través del protocolo NFS, mientras que los directorios que copiamos son "locales" a cada uno de los clientes. El proceso de montaje lo realiza el fichero */etc/fstab* local de cada cliente, que posteriormente será explicado.

3.8.2. Instalación del dhcpd en el servidor.

Para disponer del servidor DHCPD se tendrá que instalar el paquete **dhcpcd-1.3.18pl3-1.i386.rpm** disponible en el CD-ROM de Red Hat Linux de la siguiente forma:

rpm -ivh dhcpcd-1.3.18pl3-1.i386.rpm

A continuación se editará el fichero */etc/rc.d/init.d/dhcpd*. En dicho fichero se tendrá que indicar cual es el dispositivo de red para la red interna, para ello se tendrá que modificar la siguiente linea:

daemon /usr/sbin/dhcpd eth0

Sustituyendo eth0 por eth1 ya que es la ethernet para la red interna en nuestro caso concreto. Se guardará los cambios realizado a dicho fichero y se reiniciará del daemon como se muestra a continuación:

/etc/rc.d/init.d/dhcpd restart

Una vez finalizada la tarea anterior, el siguiente paso a realizar será la activación del servicio TFTP (Trivial File TransFer Protocol), debido que el posteriormente será utilizado por el paquete Etherboot[1] para transferir el núcleo por la red y otro ficheros necesario para el arranque sin disco. La ejecución del demonio tftpd se realiza a través del superdemonio inetd. La configuración de este superdemonio se encuentra en el archivo */etc/inted.conf*. Este demonio es el encargado de arrancar automáticamente el servidor correspondiente a un servicio solicitado, este servidor particular termina una vez que el servicio se ha proporcionado. Por lo tanto, el proceso inetd está a la escucha en los diferentes puertos correspondientes a los servicios disponibles.

El archivo */etc/inetd.conf* es utilizado por el proceso inetd cuando se lanza para conocer el conjunto de puertos sobre los que se tiene que poner a la escucha. Este archivo contiene una línea por servicio, cada línea suministra la siguiente información:

[1] Etherboot es un paquete software, cuya función es la creación de imagenes ROM que puede ser descargables a través de una red Ethernet para ser ejecutadas en computadores x86.

- El nombre del servicio.
- El tipo de socket.
- Una opción wait/nowait que se utiliza para las comunicaciones en modo no orientado a conexión (dgram), las otras se utilizan siempre con la opción nowait. La opción wait evita, en el modo orientado a conexión, la ejecución de varios servidores a la escucha sobre un mismo puerto.
- Un nombre de usuario que será propietario del proceso demonio asociado al servicio cuando se cree.
- La referencia absoluta del archivo que contiene el programa que proporciona el servicio.
- Una lista de parámetros para el programa.

Asi pues, se tendrá que activar la línea correspondiente al tftp eliminando la marca de comentario (#) para poder ejecutar el demonio tftpd:

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

El archivo */etc/services* contiene la lista de servicios de Internet conocidos. Un servicio se caracteriza por su nombre, un número de puerto, un protocolo y una lista de alias. El servicio anterior describe un servicio estándar en Internet basados en el protocolo UDP.

Se deberá modificar el archivo *etc/services* descomentando las siguientes dos líneas:

tftp 69/udp #TFTP server

Una vez que se han modificado los dos archivos anteriores debemos de reiniciar el superdemonio inetd mediante la orden:

kill -HUP PID_de_inetd

3.8.3. Configuración del dhcpd en el servidor.

El demonio dhcpd tiene un archivo de configuración llamado */etc/dhcpd.conf*. Finalizada la realización de los pasos anteriores se procederá a configurar las opciones DHCP creando o editando el fichero anterior. Para este caso concreto dicho fichero presentará la siguiente configuración:

```
#red interna
subnet 192.168.1.0 netmask 255.255.255.0 {
option broadcast-address 192.168.1.255;

    host pc1{
        hardware ethernet 00:50:04:09:DA:EB;
        fixed-address 192.168.1.2;
        option host-name "pc1";
        filename /tftpboot/pc1/vmlinuz.nodos";
    }
    host pc2{
```

hardware ethernet 00:50:DA:3D:2F:1C;
fixed-address 192.168.1.3;
option host-name "pc2";
filename /tftpboot/pc2/vmlinuz.nodos";
}

A continuación se realiza una aclaración de la configuración anterior:

subnet 192.168.1.0 netmask 255.255.255.0: Declaración de la subred interna.
option broadcast-address: Dirección de broadcast.
hosts pc1 : Indica que la configuración corresponde a la máquina cliente pc1.
hardware ethernet: Especifica el tipo de hardware y la dirección de la tarjeta de red de la máquina cliente.
fixed-address: Dirección IP que se asigna a la máquina cliente con la dirección de red arriba indicada.
option host-name "pc1.cluster.psa.es": Indica el nombre que se le asigna a la máquina cliente.

Terminada la configuración del servidor DHCP se reiniciaría el demonio DHCP como anteriormente se ha comentado.

3.8.4. Configuración del fichero /etc/hosts.

Este fichero contiene las informaciones relativas a las diferentes máquinas de la red local a la que pertenece el sistema. Contiene la tabla de correspondencias entre la dirección Internet y un nombre simbólico. Dentro de una red este nombre simbólico contiene una componente única. A cada máquina de la red le corresponde una línea de dicho archivo quedando la siguiente tabla:

127.0.0.1 localhost.localdomain localhost
192.168.1.1 pc0.cluster.psa.es pc0
192.168.1.2 pc1.cluster.psa.es pc1
192.168.1.3 pc2.cluster.psa.es pc2

3.8.5. Configuración del fichero /etc/exports.

Este archivo sirve como la lista de control de acceso para sistemas de ficheros que pueden ser exportados a clientes NFS. Cada línea contiene un punto de montaje y una lista de máquinas o nombres de grupo de red a las que se les permite montar el sistema de ficheros en ese punto. Su estructura se muestra a continuación:

*/tftpboot *.cluster.psa.es(rw,no_root_squash)*
*/bin *.cluster.psa.es(rw,no_root_squash)*
*/usr *.cluster.psa.es(rw,no_root_squash)*
*/sbin *.cluster.psa.es(rw,no_root_squash)*
*/home *.cluster.psa.es(rw,no_root_squash)*
*/lib *.cluster.psa.es(rw,no_root_squash)*
*/boot *.cluster.psa.es(rw,no_root_squash)*
*/root *.cluster.psa.es(rw,no_root_squash)*
*/etc/passwd *.cluster.psa.es(rw,no_root_squash)*
*/etc/group *.cluster.psa.es(rw,no_root_squash)*

donde:

rw: Permitir pedidos de lectura/escritura.
no_root_squash: Deshabilita la opción de root squashing, permitiendo conexión en modo root. Esta opción es particularmente útil para clientes sin disco.

3.8.6. Preparación de la configuración del cliente.

Como ya se ha creado una estructura de directorios donde alojar el sistema de ficheros de las máquinas clientes en los apartados anteriores, y se han copiado los directorios que contienen la configuración de Linux de la máquina servidora, ahora se tiene que adaptar para cada máquina los ficheros de configuración.
Nota: Todos los cambios que se van a realizar a continuación se harán sobre los archivos de configuración que están en la ruta */tftpboot/pc1*, es decir que cuando se diga que se modifica el *etc/hosts* se esta haciendo referencia al archivo que existe en la ruta */tftpboot/pc1/etc/hosts*.

3.8.7. Configuración del ficheros /etc/HOSTNAME.

No se dará ningun nombres, es decir, se deberá dejar el archivo vacío.

3.8.8. Configuración del fichero /etc/hosts.

En este constará unicamente las direcciones IP de los clientes que van a formar la red interna:

127.0.0.1 localhost.localdomain localhost
192.168.1.1 pc0.cluster.psa.es pc0
192.168.1.2 pc1.cluster.psa.es pc1
192.168.1.3 pc2.cluster.psa.es pc2

3.8.9. Configuración del fichero /etc/fstab.

Este archivo contiene la tabla del sistema de ficheros utilizado por el cliente pc1. Hay que indicar los directorios que se van a montar vía NFS. El contenido debe quedar de la siguiente forma:

pc0:/tftpboot/pc1 / nfs defaults 1 1
pc0:/bin /bin nfs defaults 1 1
pc0:/usr /usr nfs defaults 1 1
pc0:/sbin /sbin nfs defaults 1 1
pc0:/home /home nfs defaults 1 1
pc0:/lib /lib nfs defaults 1 1
pc0:/boot /boot nfs defaults 1 1
none /proc proc defaults 0 0
none /dev/pts devpts gid=5,mode=620 0 0

Así pues el directorio raíz de la máquina pc1 va a ser el directorio /tftpboot/pc1 de la máquina pc0 siendo este el servidor o front-end.

3.8.10. Configuración del fichero /etc/sysconfig/network.

Este archivo presenta información de red relativa a la máquina necesaria para la red, para nuestro caso del cliente pc1 queda de la siguiente forma:

NETWORKING=yes
HOSTNAME=
FORWARD_IP=no

3.8.11. Configuración del fichero /etc/sysconfig/network-scripts/ifcfg-eth0.

Este archivo especifica el nombre del dispositivo de la interfaz de red, la dirección IP, la máscara de red, la dirección de red, la dirección de broadcast e información sobre el modo de arranque. Los datos para la máquina pc1 son los siguientes:

DEVICE="eth0"
BOOTPROTO="dhcp"
BROADCAST=
IPADDR=
NETMASK=
NETWORK=
ONBOOT="yes"

En este momento la máquina cliente quedaría bien configurada para que hiciese el arranque, lo que pasa es que cuando el proceso de arranque intenta chequear los discos duros, el sistema se queda bloqueado ya que esta trabajando NFS. Para solucionar este problema se presenta el siguiente apartado.

3.8.12. Configuración del fichero /etc/rc.d/rc.sysinit.

En este archivo tenemos que comentar o eliminar las líneas que hacen referencia al chequeo de las unidades. Estas líneas son las siguientes:

```
if [ -f /fsckoptions ]; then
        fsckoptions='cat /fsckoptions'
else f
        fsckoptions=
fi
if [ -f /forcefsck ]; then
        fsckoptions="-f $fsckoptions"
fi
if [ "$BOOTUP" != "serial" ]; then
        fsckoptions=" -C $fsckoptions"
else
        fsckoptions=" -V $fsckoptions"
```

```
fi
_RUN_QUOTACHECK=0
if [ ! -f /fastboot ]; then

        STRING="Checking root filesystem"
        echo $STRING
        initlog -c "fsck -T -a $fsckoptions /"
        rc=$?

if [ "$rc- "0"]; then

        success "$STRING"echo

elif [ "$rc- "1"]; then

        passed "$STRING"echo

fi
# A return of 2 or higher means there were serious problems.
if [ $rc -gt 1 ]; then

        failure "$STRING"
        echo
        echo
        echo "*** An error occurred during the file system check."
        echo "*** Dropping you to a shell; the system will reboot"
        echo "*** when you leave the shell."
        PS1="(Repair filesystem) # # "; export PS1
        sulogin
        echo "Unmounting file systems"
        umount -a
        mount -n -o remount,ro /
        echo "Automatic reboot in progress."
        reboot -f
        elif [ "$rc" = "1" ]; then

            _RUN_QUOTACHECK=1
        fi

fi
_RUN_QUOTACHECK=0
# Check filesystems
if [ ! -f /fastboot ]; then
STRING=Çhecking filesystems"
echo $STRING
initlog -c "fsck -T -R -A -a $fsckoptions"
rc=$?
if [ "$rc- "0"]; then

        success "$STRING"
        echo
```

```
elif [ "$rc- "1"]; then

    passed "$STRING"
    echo

fi
# A return of 2 or higher means there were serious problems.
if [ $rc -gt 1 ]; then

    failure "$STRING"
    echo
    echo
    echo "*** An error occurred during the file system check."
    echo "*** Dropping you to a shell; the system will reboot"
    echo "*** when you leave the shell."
    PS1="(Repair filesystem) # # "; export PS1
    sulogin
    echo "Unmounting file systems"
    umount -a
    mount -n -o remount,ro /
    echo "Automatic reboot in progress."
    reboot -f
    elif [ "$rc" = "1" -a -x /sbin/quotacheck ]; then
    _RUN_QUOTACHECK=1

fi
fi
```

3.8.13. Comprobación del arranque vía NFS.

Una vez acabada la configuración de los clientes debemos comprobar que podemos montar la estructura de directorios creada para los clientes vía NFS realizando los siguientes pasos:

Exportar los directorios:

```
exportfs -av
```

Montar los directorios en el cliente:

```
mount -t nfs nb_servidor:/tftpboot/directorio_cliente /directorio_destino
```

En este caso quedaría:

```
mount -t nfs pc0:/tftpboot/pc1 /mnt
```

Si no lo monta correctamente se revisará todos los pasos realizados.

3.9. Configuración del núcleo.

El usuario root es el único que tiene permisos para poder compilar el kernel.

A continuación se va ha indicar los módulos que debe tener el kernel para poder realizar la petición de dirección IP y poder montar su sistema de ficheros a través de la red. La opciones que se tendrá que activar son:

- En el apartado LOADABLE MODULE SUPPORT.

  Enable loadable module support: Desactivado

- En el apartado NETWORKING OPTIONS.

  IP kernel level autoconfiguration: Activado
  BOOTP support: Activado
  RARP support: Activado

- En el apartado NETWORK FILE SYSTEM.

  NFS Filesystem support: Activado
  Root Filesystem on NFS: Activado
  NFS server support: Desactivado

Para poder compilar el kernel con las opciones anteriormente indicadas vamos a utilizar una herramienta que nos mostrará un menú en TCL/TK:

$> cd /usr/src/linux
$> make xconfig

Una vez ejecutada esta herramienta se realizará los pasos que muestran las siguientes figuras:

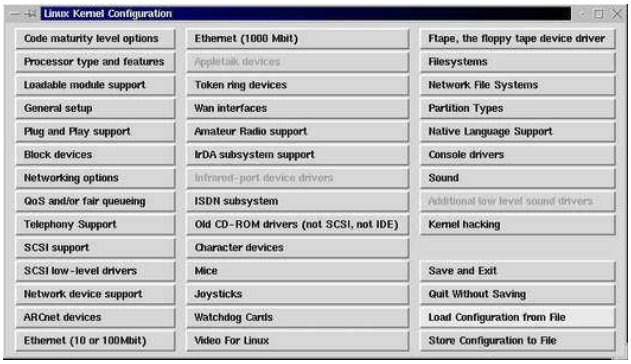

Figura 3.4: Paso 1. Cargar preconfiguración del núcleo

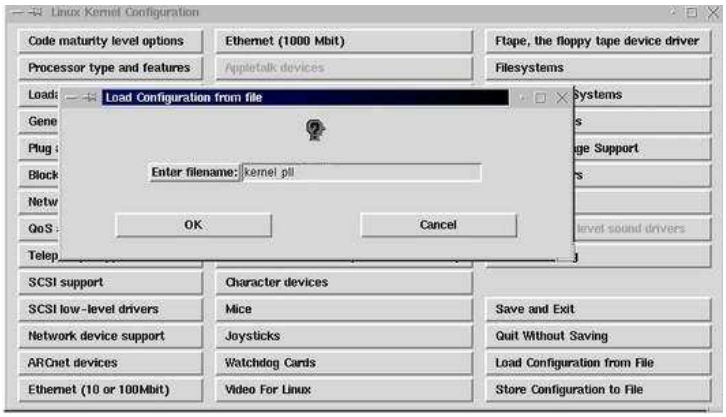

Figura 3.5: Paso 2. Cargar fichero

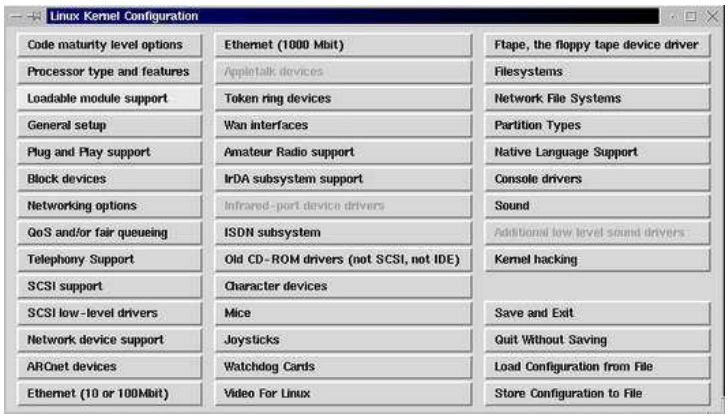

Figura 3.6: Paso 3. Loadable module support

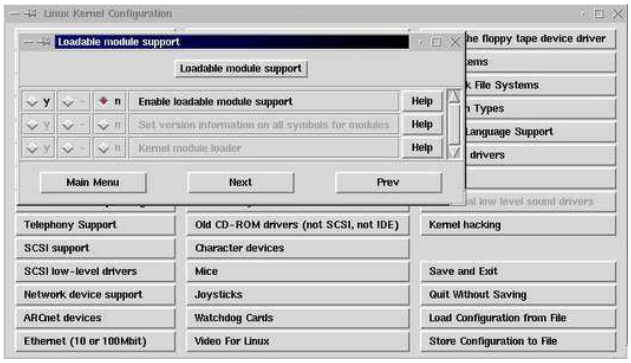

Figura 3.7: Paso 4. Desactivar loadable module support

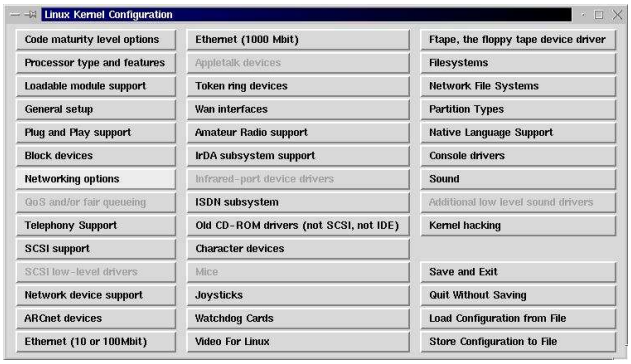

Figura 3.8: Paso 5. Networkin options

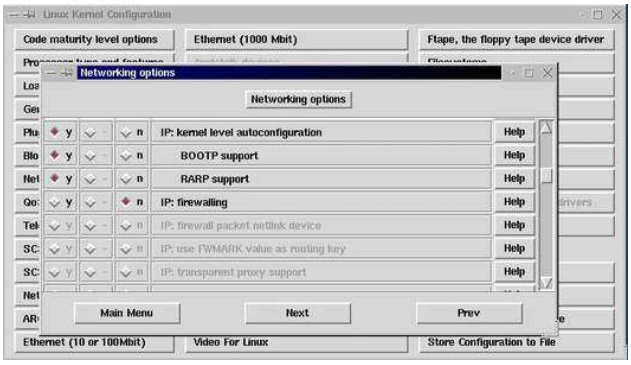

Figura 3.9: Paso 6. Activación del protocolo BOOTP

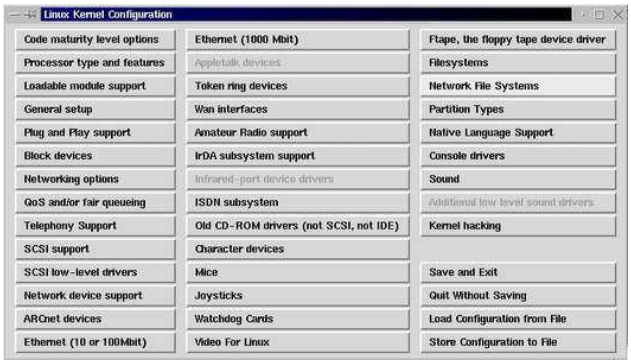

Figura 3.10: Paso 7. Networking File System

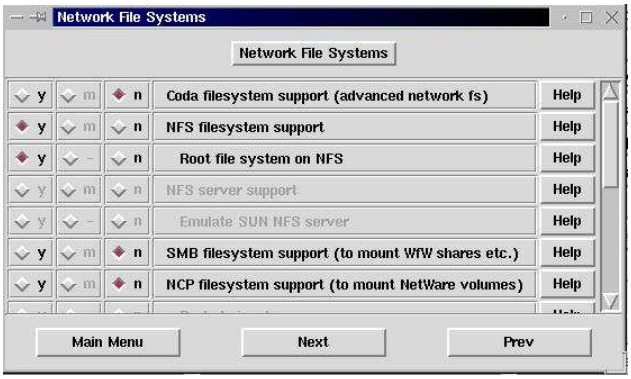

Figura 3.11: Paso 8. Activar NFS

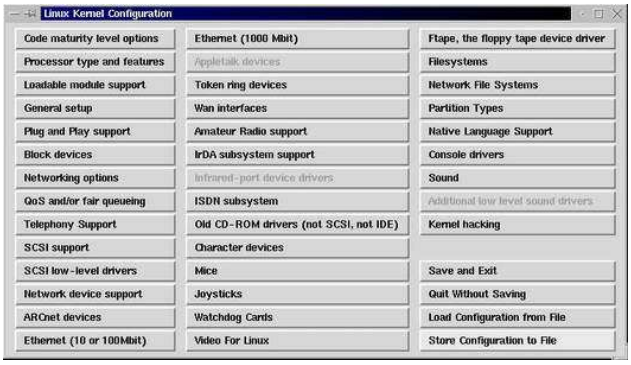

Figura 3.12: Paso 9. Guardar configuración

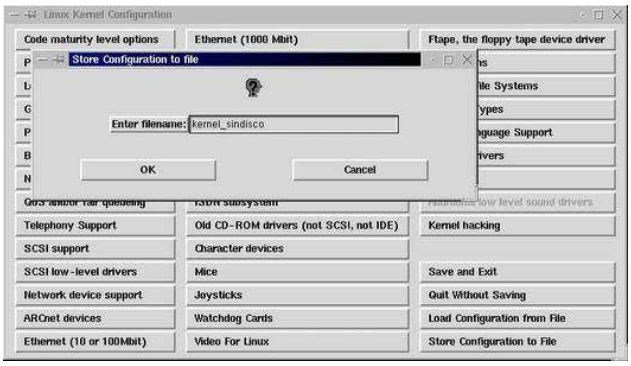

Figura 3.13: Paso 10. Guardar configuración a un archivo

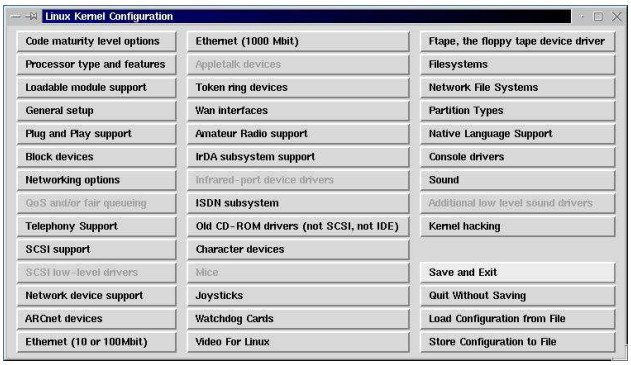

Figura 3.14: Paso 11. Salir

Figura 3.15: Paso 12. Configuración finalizada

3.10. Etherboot.

3.10.1. Introducción

Etherboot es un paquete software, cuya función es la creación de imagenes ROM que puede ser descargables a través de una red Ethernet para ser ejecutadas en computadores 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 diskeless. Esto es beneficioso en varias situaciones, por ejemplo:

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

Etherboot inicia computadores mas rápidamente que un disquete ya que no hay retardos en los giros del disco, etc. Realizando un pequeño calculo se observará que con una Ethernet de 10Mbit/s se enviará un kenel de 500kB en un par de segundos. Con una Ethernet de 100Mbit se obtendrá 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 ficheros NFS, o discos locales. Es un componente tecnologico que puede ser combinado con otras tecnologías para actuar como deseamos.

Etherboot se utiliza generalmente para cargar Linux, FreeBSD o el DOS. No obstante los formatos del fichero del protocolo y del cargador del programa inicial son generales.

Etherboot es Open Source bajo la GNU GPL2 (General Public Licence Version 2).

Los componentes necesitados por Etherboot 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 ficheros requeridos durante el proceso de arranque.

3.10.2. Funcionamiento.

A continuación se describirá 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 se le asignará en función de su dirección MAC y solicitará la transmisión del sistema archivos vía NFS.

3.10.3. Instalación.

Los fuentes del paquete Etherboot esta disponible en la web `http://etherboot.sourceforge.net/distribution.html`.

Una vez descargados dichos fuentes deberá ser compilados, a continuación se muestran los pasos a seguir para la instalación de Etherboot:

1. Descargar de la web los paquetes etherboot-5.0.2.tar.gz y mknbi-1.2.tar.gz.
2. Descomprimir los paquetes utilizando tar xvfz nombre_paquete.
3. Recompilar el núcleo con las opciones seguidas en el punto 3.9, con los siguientes comandos:

   $>make dep;make clean;make bzImage

4. Copiar la imagen del núcleo generada al directorio donde se haya descomprimido el paquete mknbi-1.2.tar.gz. La imagen del núcleo se encuentra en el directorio */usr/src/linux/arch /i386/boot*
5. En el arranque no se puede usar el fichero *bzImage*, generada en la compilación del núcleo. Esta imagen debe ser convertida en una *tagged image* (imagen etiquetada). Esta es una imagen normal 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 se usa el programa llamado *mknbi-linux*, que nos proveé el paquete mknbi-1.2.tar.gz.

   Posicionarse en el directorio donde se hay descomprimido el paquete mknbi-1.2.tar.gz (por ejemplo */home/usuario/mknbi-1.2*), transformar la imagen del núcleo, es decir, hacer una tagged imagen con el siguiente comando:

   ./mknbi -format=elf -target=linux bzImage -output=vmlinuz.nodos

6. Copiar el archivo generado vmlinuz.nodos al directorio /tftpboot
7. Posicionarse sobre el directorio src dentro del directorio en el cual se haya descomprimido el paquete etherboot-5.0.2.tar.gz, como por ejemplo */home/usuario/etherboot-5.0.2/src*
8. Introducción de un disquete en la unidad de disco y escribir los siguiente comandos:

   $>make
   $>make bin/boot1a.bin ⇒ se genera la imagen de los drivers de la tarjeta de red
   $>make bin32/3c90x.rom ⇒ esta línea variará en función de la tarjeta de red, en este caso concreto una 3COM 3c905
   $>cat bin/boot1a.bin bin32/3c90x.rom > /dev/fd0

Una vez finalizados los pasos anteriores ya estaría preparado el disquete de arranque, solamente quedaría irse a un cliente y comprobar que el proceso de arranque vía nfs funciona correctamente.

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

Capítulo 4

CONFIGURACIÓN FIREWALL.

Contents

4.1. Introducción.

Un *cortafuegos* en el mundillo de las redes de ordenadores e un dispositivo físico o lógico que protege una red privada del resto de la red (pública).

- Se toma un ordenador con capacidad de rutar (por ejemplo un PC con LINUX).
- Se le ponen dos interfaces (por ejemplo interfaces serie, o ethernet, o de paso de testigo con anillo (Token Ring), etc...).
- Se le deshabilita el reenvío de paquetes IP (IP forwarding).
- Se conecta una interfaz a la Internet.
- Se conecta la otra interfaz a la red que se quiere proteger.

Ahora hay dos redes distintas que comparten un ordenador. El ordenador que actua de cortafuegos, al que de ahora en adelante llamaremos çortafuegos", puede comunicarse tanto con la red protegida como con la Internet. La red protegida no puede comunicarse con la Internet, y la Internet no puede comunicarse con la red protegida, dado que hemos deshabilitado el reenvío IP en el único ordenador que las conecta.

Si se quiere llegar a la Internet desde la red protegida, hay que hacer primero un telnet al cortafuegos, y acceder a la Internet desde él. Del mismo modo, para acceder a la red protegida desde la Internet, se debe antes pasar por el cortafuegos.

Este es un mecanismo de seguridad excelente contra ataques desde la Internet. Si alguien quiere atacar la red protegida, primero tiene que atravesar el contrafuegos. De estar manera el ataque se divide en dos pasos, y, por lo tanto, se dificulta. Si alguien quiere atacar la red protegida por métodos más comunes, como el bombardeo de emails, o el nefasto "Gusano de Internet", simplemente no podrá alcanzarla. Con esto se consigue una protección excelente.

Un cortafuegos puede proteger una red de diversas formas. Puede proporcionar servicios de encubrimiento que nieguen o garanticen los accesos basados en: el nombre del usuario, el nombre del host, y el protocolo TCP/IP, etc. Un cortafuegos puede suministrar también una variedad de servicios que dejen paso a los usuarios autorizados (si se implementa un PROXY) mientras excluyen a los no autorizados. Al mismo tiempo, asegura que todas las comunicaciones entre la red e Internet dan la impresión de finalizar en el cortafuegos, si se usa NAT (Network Address Translation), evitando que el mundo externo pueda vislumbrar en modo alguno la estructura de la red.

4.2. Introducción al IPCHAINS.

Linux emphipchains es una modificación de la codificación de Linux IPv4 firewalling y una modificación de *ipfwadm*. Es necesaria para la administración del filtrado de paquetes de IP en las versiones 2.1.102 o posteriores.

Ipchains es la herramienta que te permite administrar los recursos de tu red a nivel ip, permitiendo el tráfico de determinados paquetes y denegando el acceso a otros, permite cerrar puertos, redireccionarlos, esconderlos, etc. Es la herramienta necesaria para montar y administrar los cortafuegos.

Ipchains se basa en una lista (cadena) de reglas que determinan el comportamiento y las decisiones a tomar sobre paquetes cuando alcanzan un interfaz de entrada o salida. Inicialmente siempre hay cadenas de reglas que son las básicas y sobre las que se construye todo lo demás. Estas son:

- Input
- Output
- Forward

Con ellas hacemos respectivamente alusión a los paquetes que entran, a los que salen, y a los que se enruta. Aparte de estas tres básicas se pueden definir otras por el usuario.

Cada lista de reglas contiene, como se ha comentado anteriormente, las reglas, a las cuales se van consultando secuencialmente desde la primera a la última, cuando se encuentra una con la que coincide el paquete, se aplica. Si al final de todas las reglas no se ha encontrado ninguna, se adopta la política por defecto que recordemos tenía dos formas: aceptar o denegar.

Los comandos para manipular las listas de reglas son (siempre en mayúsculas):

- *N*: Crea una nueva cadena de reglas.
- *X*: Borra una cadena de reglas que antes debe estar vacía.
- *P*: Cambia la política de la cadena de reglas. Esta puede ser ACCEPT, DENY, REJECT y MASQ (ésta solo valida 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 susodicha regla se saltará a donde indique -j que puede ser aceptar, denegar, rejectar 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 referenciamos a todo el mundo. también se pueden especificar los puertos, estos se especifican al final, tras la ip/mask y pueden 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, UDP, ICMP). Se pueden poner también los números equivalentes. Con el protocolo ICMP, 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 el 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.
  - **-y** referencia a los paquetes SYN que son los que se usan para iniciar una conexión. Con ! hacemos referencia a los que no son para iniciar una conexión.
  - **-f** la regla solo se aplicará al segundo y demás fragmentos de un paquete, no se permite especificar puertos.
  - **-j** especifica el objetivo de la regla que puede ser: ACCEPT, REJECT y DENY (deniegan), MASQ (aplica masquerade a un paquete, solo 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 al original. Si no se especifica -j, la regla solo 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.
  - **-v** en conjunción con -L aumenta la información ofrecida.

4.3. Configuración del firewall.

La configuración que se llevará a cabo permitirá la conexión desde el exterior al front-end a través del protocolo SSH, además se podrán conectar a la web de documentación a través del puerto 80.

```
#!/bin/sh
# firewall
# probe: true
# Source function library.
. /etc/rc.d/init.d/functions
# Source networking configuration.
if [ ! -f /etc/sysconfig/network ]; then
exit 0
fi
. /etc/sysconfig/network
# Check that networking is up.
[ $NETWORKING = "no" ] && exit 0
if [ ! -x /sbin/ipchains ]; then
exit 0
fi
# See how we were called.
case "$1" in
start)
# Activacion de firewall
echo "============================================="
echo "Activacion de las reglas del firewall para cluster.psa.es"
echo "============================================="
# Proteccion a nivel del nucleo
# Activacion ip_masq
echo 1 > /proc/sys/net/ipv4/ip_forward
# Enable IP spoofing protection
for f in /proc/sys/net/ipv4/conf/*/rp_filter; do
echo 1 > $f
done
# Enable TCP SYN Cookie Protection
echo 1 > /proc/sys/net/ipv4/tcp_syncookies
# Enable always defragging Protection
echo 1 > /proc/sys/net/ipv4/ip_always_defrag
# Enable broadcast echo Protection
echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
# Enable bad error message Protection
echo 1 > /proc/sys/net/ipv4/icmp_ignore_bogus_error_responses
# Disable ICMP Redirect Acceptance
for f in /proc/sys/net/ipv4/conf/*/accept_redirects; do
echo 0 > $f
done
# Disable Source Routed Packets
for f in /proc/sys/net/ipv4/conf/*/accept_source_route; do
```

```
echo 0 > $f
done
# Log Spoofed Packets, Source Routed Packets, Redirect Packets
for f in /proc/sys/net/ipv4/conf/*/log_martians; do
echo 1 > $f
done
# Inclusion de modulos de enmasqueramiento en el kernel
/sbin/modprobe ip_masq_ftp
/sbin/modprobe ip_masq_irc
/sbin/modprobe ip_masq_quake
/sbin/modprobe ip_masq_cuseeme
/sbin/modprobe ip_masq_raudio
/sbin/modprobe ip_masq_vdolive
# Variables de configuracion del firewall
PATH=/sbin:$PATH
export ppp0ADDR=ifconfig eth0 — grep inet addr — awk print $2 — sed -e s/.*:// /24"
export device=eth0
echo "Utilizando la ip $ppp0ADDR de la interfaz $device"newline # Reglas del firewall
# Limpiamos reglas anteriores
ipchains -F
# Set the default policy to deny
ipchains -P input ACCEPT
ipchains -P output ACCEPT
ipchains -P forward DENY
# Anti-spoofing
# Refuse packets claiming to be to the loopback interface
ipchains -A input -p all -j DENY -s 127.0.0.0/8 -i $device -l
# Refuse packets claiming to be to a Class A private network
ipchains -A input -p all -j DENY -s 10.0.0.0/8 -i $device -l
# Refuse packets claiming to be to a Class B private network
ipchains -A input -p all -j DENY -s 172.16.0.0/12 -i $device -l
# Refuse packets claiming to be to a Class C private network
ipchains -A input -p all -j DENY -s 192.168.0.0/16 -i $device -l
# Refuse Class D multicast addresses
ipchains -A input -p all -j DENY -s 224.0.0.0/4 -i $device -l
# Refuse Class E reserved IP addresses
ipchains -A input -p all -j DENY -s 240.0.0.0/5 -i $device -l
# Refuse malformed broadcast packets
ipchains -A input -p all -j DENY -s 255.255.255.255 -i $device -l
ipchains -A input -p all -j DENY -d 0.0.0.0 -i $device -l
# Refuse addresses defined as reserved by the IANA.
ipchains -A input -p all -j DENY -s 1.0.0.0/8 -i $device -l
ipchains -A input -p all -j DENY -s 2.0.0.0/8 -i $device -l
ipchains -A input -p all -j DENY -s 5.0.0.0/8 -i $device -l
ipchains -A input -p all -j DENY -s 7.0.0.0/8 -i $device -l
ipchains -A input -p all -j DENY -s 23.0.0.0/8 -i $device -l
```

```
ipchains -A input -p all -j DENY -s 27.0.0.0/8 -i $device -l
ipchains -A input -p all -j DENY -s 31.0.0.0/8 -i $device -l
ipchains -A input -p all -j DENY -s 37.0.0.0/8 -i $device -l
ipchains -A input -p all -j DENY -s 39.0.0.0/8 -i $device -l
ipchains -A input -p all -j DENY -s 41.0.0.0/8 -i $device -l
ipchains -A input -p all -j DENY -s 42.0.0.0/8 -i $device -l
ipchains -A input -p all -j DENY -s 58.0.0.0/7 -i $device -l
ipchains -A input -p all -j DENY -s 60.0.0.0/8 -i $device -l
ipchains -A input -p all -j DENY -s 65.0.0.0/8 -i $device -l
ipchains -A input -p all -j DENY -s 66.0.0.0/7 -i $device -l
ipchains -A input -p all -j DENY -s 68.0.0.0/6 -i $device -l
ipchains -A input -p all -j DENY -s 72.0.0.0/5 -i $device -l
ipchains -A input -p all -j DENY -s 80.0.0.0/4 -i $device -l
ipchains -A input -p all -j DENY -s 96.0.0.0/3 -i $device -l
ipchains -A input -p all -j DENY -s 112.0.0.0/3 -i $device -l
ipchains -A input -p all -j DENY -s 169.254.0.0/16 -i $device -l
ipchains -A input -p all -j DENY -s 192.0.0.0/24 -i $device -l
ipchains -A input -p all -j DENY -s 217.0.0.0/8 -i $device -l
ipchains -A input -p all -j DENY -s 218.0.0.0/7 -i $device -l
ipchains -A input -p all -j DENY -s 220.0.0.0/6 -i $device -l
ipchains -A input -p all -j DENY -s 248.0.0.0/5 -i $device -l
# Rechazamos la conexion con nuestra ip interna
ipchains -A input -p all -j DENY -d 192.168.1.0/25 -i $device -l
# ICMP
ipchains -A input -p icmp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR -l
# SSH abierto
ipchains -A input -p tcp -j ACCEPT -s 0.0.0.0/0 -i $device -d $ppp0ADDR 22 -l
# HTTP abierto
ipchains -A input -p tcp -j ACCEPT -s 0.0.0.0/0 -i $device -d $ppp0ADDR 80 -l
ipchains -A input -p udp -j ACCEPT -s 0.0.0.0/0 -i $device -d $ppp0ADDR 80 -l
# Set uid de conexion
ipchains -A input -p tcp -y -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR -l
# Bloqueo 1:21
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 1:21 -l
ipchains -A input -p udp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 1:21 -l
# Bloqueo 23:79
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 23:79 -l
ipchains -A input -p udp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 23:79 -l
# Bloqueo 81:1023
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 81:1023 -l
ipchains -A input -p udp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 81:1023 -l
# Acepta ssh
ipchains -A input -p tcp -s 0/0 -d 0/0 22 -j ACCEPT
ipchains -A input -p udp -j ACCEPT -s 0.0.0.0/0 -i $device -d $ppp0ADDR 22 -l
# Acepta http
ipchains -A input -p tcp -s 0/0 -d 0/0 80 -j ACCEPT
ipchains -A input -p udp -j ACCEPT -s 0.0.0.0/0 -i $device -d $ppp0ADDR 80 -l
# Bloqueo de otros puertos
```

```
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 1109 -l
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 1243 -l
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 1524 -l
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 1600 -l
ipchains -A input -p udp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 2001 -l
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 2001 -l
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 2003 -l
ipchains -A input -p udp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 2049 -l
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 2049 -l
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 2105 -l
ipchains -A input -p udp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 3001 -l
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 3001 -l
ipchains -A input -p udp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 3128:3130
-l
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 3128:3130
-l
ipchains -A input -p udp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 3306 -l
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 3306 -l
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 4444 -l
ipchains -A input -p udp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 6000:6100
-l
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 6000:6100
-l
ipchains -A input -p udp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 6600:6800
-l
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 6600:6800
-l
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 7000 -l
# Back Orifice
ipchains -A input -p udp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 31337 -l
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 31337 -l
# NetBus
ipchains -A input -p udp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 12345:12346
-l
ipchains -A input -p tcp -j DENY -s 0.0.0.0/0 -i $device -d $ppp0ADDR 12345:12346
-l
# Reglas de redireccionamiento
ipchains -A forward -p all -j MASQ -s 192.168.1.0/255.255.255.0 -d 0.0.0.0/0
;;
stop)
echo "Parando Servicios del Firewall: "
# Borra todas las reglas por defecto
ipchains -F
# Borra todas las chain definidas por el usuario para el filtrado
ipchains -X
# Resetea los poltica por defecto de fitrado
ipchains -P input ACCEPT
ipchains -P output ACCEPT
```

ipchains -P forward ACCEPT
;;
restart—reload)
$0 stop
$0 start
;;
**)*
echo "Uso: firewall start—stop—restart—reload"
exit 1
;;
esac
exit 0

Este fichero únicamente podrá ser manipulado por el superusuario y ser localizado en directorio: $>/etc/rc.d/init.d/

4.4. Activación del firewall en el arranque

El firewall se activará al iniciarse nuestro sistema justo después de activarse o levantarse todos los servicios en el arranque. Para llevar a cabo eso editaremos el ficheros */etc/rc.d/rc.local* y al final de este se escribirá la siguiente línea:

#activación de las reglas del firewall
/etc/rc.d/init.d/firewall start

Capítulo 5

PVM Y XPVM.

Contents

5.1. Introducción PVM.

PVM (Paralel Virtual Machine) es una herramienta diseñada para solucionarnos una gran cantidad de problemas asociados con la programación paralela. Sobre todo, el monetario. Para ello, nos va a crear 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. Es decir, disponemos de todas las ventajas económicas asociadas a la programación distribuida, ya que empleamos los recursos hardware de dicho paradigma; pero programando el conjunto de máquinas como si se tratara de una sola máquina paralela, que es mucho más cómodo.

La PVM es el estándar de facto del mundo científico. De hecho, en el área de la Física Computacional, la PVM es una biblioteca ampliamente usada.

La máquina paralela virtual es una máquina que no existe, pero un API apropiado nos permite programar como si existiese. El modelo abstracto que nos permite usar el API de la PVM consiste en una máquina multiprocesador completamente escalable (es decir, que podemos aumentar y disminuir el número de procesadores *en caliente*). Para ello, nos va a ocultar la red que estemos empleando para conectar nuestras máquinas, así como las máquinas de la red y sus características específicas. Este planteamiento tiene numerosas ventajas respecto a emplear un supercomputador, de las cuales, las más destacadas son:

- **Precio.** Así como es mucho más barato un computador paralelo que el computador tradicional equivalente, un conjunto de ordenadores de mediana o baja potencia es muchísimo más barato que el computador paralelo de potencia equivalente. Al igual que ocurrirá con el caso del computador paralelo, van a existir factores (fundamentalmente, la lentitud de la red frente a la velocidad del bus del computador paralelo) que van a hacer de que sean

necesarios más ordenadores de pequeña potencia que los teóricos para igualar el rendimiento. Sin embargo, aun 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), podemos emplear en el proceso los tiempos muertos de los procesadores de todas las máquinas de nuestra red a las que tengamos acceso. Por ello, si ya tenemos una red Unix montada, el costo de tener un supercomputador paralelo va a ser cero ya disponemos de las máquinas, no tendremos que comprar nada nuevo, y además la biblioteca PVM es software libre, por lo que no hay que pagar para usarla.

- **Disponibilidad**. Todo centro de cálculo tiene un mínimo de una docena de máquinas arrumbadas en una esquina, y que 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, podemos instalar Linux, la PVM y añadirlo al supercomputador paralelo virtual que conforma las máquinas que ya tendríamos en red.

- **Tolerancia a fallos**. Si por cualquier razón falla uno de los ordenadores que conforman nuestra PVM y el programa que la usa está razonablemente bien hecho. Nuestra aplicación puede seguir funcionando sin problemas. En un caso como el nuestro, en el que la aplicación va a estar corriendo durante meses, es crítico que la aplicación sea tolerante a fallos. 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**. Podemos crear una máquina paralela virtual a partir de ordenadores de cualquier tipo. La PVM nos va a abstraer 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. Este último punto es de extrema importancia, ya que el principal problema que tendríamos en los *sockets* era la programación de rutinas de conversión de formato de datos entre todos los ordenadores de la red, puesto que la codificación, tanto de enteros como de flotantes, puede ser distinta. Por último, nos permite incluir en nuestra PVM hasta máquinas paralelas. Una máquina paralela en una PVM se puede comportar tanto como una sola máquina secuencial (caso, por ejemplo, del 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. La hemos 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: nos podemos olvidar del paralelismo fuertemente acoplado. Si disponemos de una red Ethernet, simplemente la red va a dejar de funcionar para todas las aplicaciones (incluida PVM) de la cantidad de colisiones que se van a producir en caso de que intentemos paralelismo fuertemente acoplado. Si disponemos 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 coste. 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 parte es el daemon, llamado *pvmd*. En la versión actual de la PVM -la 3-, 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 (de hecho, la instalación por defecto es así). Esto nos va a permitir hacer supercomputación como usuarios, sin tener que discutir con el administrador de la red que programas vamos a poder ejecutar (aunque suele ser una buena idea comentar que vamos a instalar la PVM en el sistema, por la carga que puede llegar a producir en las comunicaciones globales en algunos casos). Una vez que un usuario (o superusuario) instaló en un directorio la PVM, todos los usuarios pueden hacer uso de esa instalación con el requisito de que el directorio donde esté instalada la PVM sea de lectura al usuario que quiera hacer uso de ella.

En muchos centros de computación, el administrador prefiere instalar él mismo la 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, nosotros como administradores podremos 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 nuestros programas para la PVM y de gerenciar 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. Esto nos va a evitar el problema que teníamos con los Sockets ya que teníamos que hacer una rutina de codificación y otra de decodificación, al menos, por cada arquitectura distinta del sistema.

Cada usuario, arrancará el daemon como si de un programa normal se tratase, 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 procesadores y alterar las propiedades de la máquina virtual. Toda aplicación se ha de enlazar a la biblioteca para poderse ejecutar después. Tendremos tres ficheros de bibliotecas, la *libpvm3.a* (biblioteca básica en C), la *libgpvm3.a* (biblioteca de tratamiento de grupos) y la *libfpvm3.a* (biblioteca para Fortran).

Un programa para PVM va a ser un conjunto de tareas que cooperan entre si. Las tareas se van a intercambiar información empleando paso de mensajes. La PVM, de forma transparente al programador, nos 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** (Task Identification Number). Es el número al que se mandan los mensajes habitualmente. Sin embargo, no es el único método de referenciar una tarea en la PVM. Muchas aplicaciones paralelas necesitan hacer el mismo 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 a las que nos podemos referir con el mismo código, el identificador de grupo. Para que una tarea entre o salga de un grupo, basta con avisar de la salida o entrada al grupo. Esto nos va a dotar de un mecanismo muy cómodo y potente para realizar programas empleando modelos *SIMD* (Single Instruction, Multiple Data), en el que vamos a dividir nuestros datos en muchos datos pequeños que sean fáciles de tratar, y después vamos a codificar la operación simple y replicarla tantas veces como datos unitarios tengamos de dividir el problema. Para trabajar con grupos, además de enlazar la biblioteca de la PVM (*libpvm3.a*) tenemos que enlazar también la de grupos (*libgpvm3.a*).

Habitualmente para arrancar un programa para la PVM, se lanzará manualmente desde un

ordenador contenido 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* (que va a ser la primera en ejecutarse), 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.

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 para codificar el mensaje.

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 necesariedad 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 seleccionados 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 si. Esto no es trivial, ya que empleando UDP puede que enviemos dos mensajes y que lleguen fuera de orden (UDP es un protocolo no orientado a conexión). 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, tendremos un mínimo de ($n)(n$-1) conexiones TCP activas. Provocando esto que hasta para números ridículos de n nos quedamos sin puertos por éste 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 debe a que, al ser comunicaciones locales, la carga derivada de la apertura y cierre de un canal es muy pequeño. Además, no vamos 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 podemos hacerla mediante primitivas bloqueantes, no bloqueantes o con un tiempo máximo de espera. La PVM nos dotará de primitivas para realizar los tres tipos de recepción. En principio nos serán más cómodas las bloqueantes, ya que nos darán un mecanismo de sincronización bastante cómodo. Las de tiempo máximo de espera nos serán útiles para trabajar con ellas como si fuesen bloqueantes, mas dando soporte al hecho de que puede que el que tiene que mandarnos 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 decirnos cuándo una tarea acabó. Para informarnos 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 trabajamos

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 del Fortran *libfpvm3.a*.

Si deseamos trabajar en 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 podemos invocar la función; bien directamente si el lenguaje lo permite, bien haciendo alguna pequeña rutina para adaptar el tipo de los datos, el formato de llamada a función o cualquiera de las restricciones que nos imponga el lenguaje que empleemos para invocar funciones en C.

Hemos de destacar que toda función en C *pvm_alguna cosa* tiene como equivalente en Fortran *pvmfalgunacosa*, y viceversa.

El programa PVM corresponde al interprete de comandos de nuestra 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 podremos eliminar la máquina desde la que estamos ejecutando el interprete 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. Tremendamente útil en los momentos de desesperación.
- **id**: Imprime el TID de la consola.
- **jobs**: Genera un listado de los trabajos en 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 ejecuciónón, 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 cuyas opciones veremos en una sección aparte.
- **trace**: Actualiza o visualiza la máscara de eventos traceados.
- **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.

Podemos obtener la PVM vía ftp anónimo: `ftp://netlib2.cs.utk.edu`

5.2. Instalación PVM.

El paquete rpm de PVM que se va ha instalar es el que nos provee el cd de Red Hat Linux 6.2, para llevar a cabo su instalación se realizará lo siguiente:

$>mount /mnt/cdrom
$>cd /mnt/cdrom/RedHat/RPMS/
$>rpm –ivh pvm-3.4.3-4.i386.rpm

En el sitio web `http://www.epm.ornl.gov/pvm/pvm_home.html` se podrá obtener la última versión de dicho software además de abundante información y sitios relacionados.

5.3. Configuración PVM.

En el directorio del usuario se tendrá que crear la siguiente estructura de directorios *$HOME/pvm3/bin/LINUX*. A continuación se modificará el archivo *.bashrc* quedando de la siguiente forma:

```
# .bashrc
# User specific aliases and functions
# 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
#
 # inscripcion 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

5.4. Compilación y ejecución de programas con PVM.

Antes de compilar se tendrá que comprobar que la PVM esta activa de la siguiente forma:

$>pvm

Una vez activada la PVM utilizaremos 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
```

5.5. Introducción 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 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 ficheros temporales relacionados con PVM.

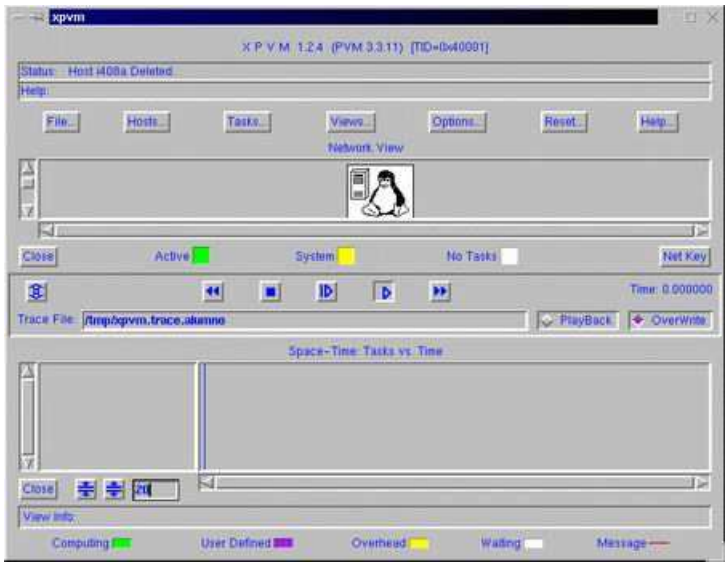

Figura 5.1: Apariencia XPVM

Las 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 nos permite añadir un nuevo host a la máquina virtual, seleccionado de entre todos los hosts listados en el fichero *.xpvm_hosts*.

En este caso vamos a añadir varios hosts. Cada vez que añadimos uno aparece un nuevo símbolo de host conectado a los símbolos existentes.

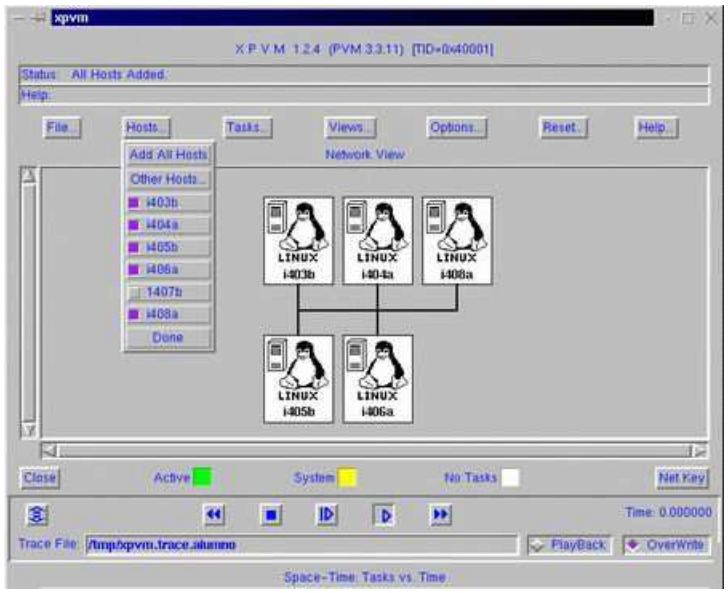

Figura 5.2: Conexión entre nodos

A través del menú Tasks-SPAWN pueden lanzarse tareas en cualquiera de los hosts que compone la máquina.

La vista 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 visualización de tareas. La visualización puede ser interrumpida o terminada en cualquier momento, utilizando los botones PAUSE y STOP. Una vez detenida la visualización se mover hacia el pasado o el futuro de las tareas utilizando los botones REWIND y FORWARD.

La vista de Representación de Tareas se compone de dos ventanas. La ventana izquierda contiene el nombre del host 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 que aparece a la izquierda de los botones anteriormente mencionados.

La ventana derecha muestra, para cada proceso, el estado de dicha tarea en cada momento, así como líneas rojas que emanan de cada proceso y que corresponden a envíos de mensajes entre procesos. El código de colores muestra el estado del proceso, que puede estar ejecutando tareas propias (verde), rutinas PVM (amarillo) o esperando mensajes (blanco).

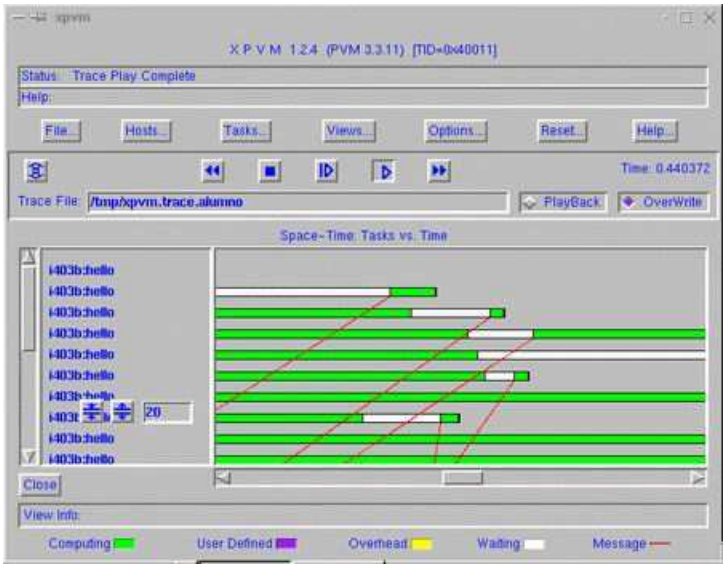

Figura 5.3: Representación de las tareas

El usuario puede recabar información detallada sobre un estado determinado o un mensaje, seleccionando con el botón izquierdo un estado u 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 enviado y el identificador de mensaje.

La representación de tareas en la ventana derecha puede ampliarse o reducirse (zooming) utilizando simultáneamente los dos botones del ratón (simula el botón central de un ratón de tres botones) y el botón derecho, respectivamente.

Existe una ventana de salida de tareas, accesible a través del menú VIEWS, que actúa como salida standard 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.

Capítulo 6

LAM/MPI.

Contents

6.1. Introducción MPI.

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, el concepto básico en el proceso de comunicación mediante mensajes es bien entendido. 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.

Al diseñarse MPI, se tomaron en cuenta las características más atractivas de los sistemas existentes para el paso de mensajes, en vez de seleccionar uno solo de ellos y adoptarlo como el estándar. Resultando así, en una fuerte influencia para MPI los trabajos hechos por IBM, INTEL, NX/", Express, nCUBE's Vernex, p4 y PARMACS. Otras contribuciones importantes provienen de Zipcode, Chimp, PVM, Chameleon y PICL.

La meta de MPI o Message Passing Interface (Interfaz de paso de mensajes), es el desarrollar un estándar (que sea ampliamente usado) para escribir programas que implementen el paso de mensajes. Por lo cual el interfaz intenta establecer para esto un estándar práctico, portable, eficiente y flexible.

El esfuerzo para estandarizar MPI involucro 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 involucrados 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 llego 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 el MPI fue presentado en la conferencia de Supercomputación en Noviembre de 1993, constituyéndose así el foro para el MPI.

En un ambiente de comunicación con memoria distribuida en la 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.

MPI es un sistema complejo, el cual comprende 129 funciones, de las cuales la mayoría tiene muchos parámetros y variantes.

Metas del MPI

- Diseñar una interfaz de programación aplicable (no necesariamente para compiladores o sistemas que implementan una librería).
- Permitir una comunicación eficiente. Evitando el copiar de memoria a memoria y permitiendo (donde sea posible) la sobreposición de computación y comunicación, además de aligerar la comunicación con el procesador.
- Permitir implementaciones que puedan ser utilizadas en un ambiente heterogéneo.
- Permitir enlaces convenientes en C y Fortran 77 para el interfaz.
- Asumir un interfaz de comunicación seguro. El usuario no debe lidiar con fallas de comunicación. Tales fallas son controladas por el subsistema de comunicación interior.
- Definir un interfaz que no sea muy diferente a los actuales, tales como PVM, NX, Express, p4, etc., y proveer de extensiones para permitir mayor flexibilidad.
- Definir un interfaz que pueda ser implementado en diferentes plataformas, sin cambios significativos en el software y las funciones internas de comunicación.
- La semántica del interfaz debe ser independiente del lenguaje.
- La interfaz debe ser diseñada para producir tareas seguras.

6.1.1. Modelo de programación.

En el modelo de programación MPI, un computo 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 del MPI es el soporte para la programación modular. Un mecanismo llamado comunicador permite al programador del MPI definir módulos que encapsulan estructuras internas de comunicación (estos módulos pueden ser combinados secuencialmente y paralelamente).

6.1.2. Bases.

Aunque MPI es un sistema complejo y multifacético, podemos resolver un amplio rango de problemas usando seis de sus funciones, estas funciones inician y terminan un cómputo, identifican procesos, además de mandar y recibir mensajes.

- **MPI_INT:** Inicia un computo.
  MPI_INT(int *argc,char ***argv), argc, argv son requeridos solo por el contexto del lenguaje C, en el cual son los argumentos del programa principal.
- **MPI_FINALIZE:** Termina un computo. MPI_FINALIZE().
- **MPI_COMM_SIZE:** Determina el número de procesos en un computo.
  MPI_COMM_SIZE(comm,size), IN comm comunicador(manejador[handle]),
  OUT size número de procesos en el grupo del comunicador(entero).
- **MPI_COMM_RANK:** Determina el identificador del proceso actual "mi proceso".
  MPI_COMM_RANK(comm,pid), IN comm comunicador (manejador[andel]), OUT pid identificador del proceso en el grupo del comunicador(entero).
- **MPI_SEND:** Manda un mensaje. MPI_SEND(buf, count, datatype, dest, tag, comm). IN buf dirección del buffer a enviar (tipo x), IN count número de elementos a enviar del buffer (entero¿=0), IN datatype tipo de datos del buffer a enviar (handle), IN dest identificador del proceso destino (entero), IN tag message tag (entero), IN comm comunicador (handle)
- **MPI_RECV:** Recive un mensaje. MPI_RECV(buf,count,datatype,dest,tag,comm). OUT buf dirección del buffer a recibir (tipo x), IN count número de elementos a recibir del buffer (entero¿=0), IN datatype tipo de datos del buffer a recibir (handle), IN source identificador del proceso fuente, o MPI_ANY_SOURCE(entero), IN tag message tag, o MPI_ANY_TAG(entero), IN comm comunicador (handle), OUT status estado del objeto (estado)
- **IN:** Significa que la función usa pero no modifica el parámetro.
- **OUT:** Significa que la función no usa pero puede modificar el parámetro.
- **INOUT:** Significa que la función usa y modifica el parámetro.

Todas las funciones (excepto las dos primeras) toman un manejador (handle) "comunicador" como argumento. El comunicador identifica el grupo de procesos y el contexto en el cual la operación se debe realizar. Como se menciono anteriormente, los comunicadores proveen un mecanismo para identificar subconjuntos de procesos durante el desarrollo de programas modulares y para garantizar que los mensajes provistos con diferentes propósitos no sean confundidos.

Por ahora, es suficiente proveer el valor de default MPI_COMM_WORLD, el cual identifica todos los procesos en un cómputo.

Las funciones MPI_INT y MPI_FINALIZE son usadas para iniciar y terminar un computo MPI, respectivamente MPI_INIT debe ser llamada antes que cualquier otra función MPI y debe ser llamada solamente una vez por proceso. Ninguna función MPI puede ser llamada después de MPI_FINALIZE.

Las funciones MPI_COMM_SIZE y MPI_COMM_BANK determinan el número de procesos en el cómputo actual y el identificador (entero) asignado al proceso actual, respectivamente. Los procesos en un grupo de procesos son identificados con un único y continuo número (entero), empezando en 0.

El estándar del MPI no especifica como un cómputo paralelo es iniciado. Pero un mecanismo típico podría ser el especificar desde la línea de comandos el número de procesos a crear.

Determinismo:

- El paso de mensajes en módulos de programación son por defecto no deterministicos; el orden de llegadas de los mensajes enviados desde dos procesos A y B hacia un tercer proceso C, no esta 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 solo escritor y un solo 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.

Especificaciones en el contexto del lenguaje C:

- Los nombres de las funciones son tal y como se presentan en la definición del MPI pero solo con el prefijo de MPI y la primer letra del nombre de la función en mayúsculas.

- Los valores de los estados son regresados como códigos enteros. El código de regreso para una ejecución exitosa es MPI_SUCESS.

- También esta definido un conjunto de códigos de error.

- Las constantes están en mayúsculas y son definidas en el archivo mpi.h

- Los handles son representados por tipos especialmente definidos en mpi.h

- Los parámetros de las funciones del tipo IN son pasados por valor, mientras que los parámetros OUT y INOUT son pasados por referencia (como apuntadores).

- Las variables de estado (status) tiene el tipo MPI_Status y es una estructura con campos status. MPI_SOURCE y status.MPI_TAG.

- Los tipos de datos del MPI están definidos para cada tipo de datos de C: MPI_CHAR, MPI_INT, MPI_LONG_INT, MPI_UNSIGNED_CHAR, etc.

6.1.3. Operaciones globales.

Como ya se ha explicado, los algoritmos paralelos ejecutan llamadas a operaciones para coordinar la comunicación en múltiples procesos. Por ejemplo, todos los procesos pueden necesitar cooperar para invertir una matriz distribuida o para sumar un conjunto de números distribuidos en cada proceso. Claramente, 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.

6.1.4. Comunicación asíncrona.

La necesidad por tener una comunicación asíncrona puede presentarse cuando un computo necesita acceder los elementos de un dato estructurado compartido en una manera no estructurada. Una implementación aproximada es el encapsular los datos estructurados en un conjunto de tareas de datos especializados, en la cual las peticiones de lectura y escritura pueden ser ejecutadas. Este método no es eficiente en MPI debido a su modelo de programación MPMD.

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:** chequea existencia de mensajes pendientes sin recibirlos, permitiéndonos escribir programas que generan cómputos locales con el procesamiento de mensajes sin previo aviso. El mensaje puede ser recibido usando MPI_RECV.

- **MPI_PROBE:** es utilizado para recibir mensajes de los cuales se tiene información incompleta. El siguiente fragmento de código hace uso de estas funciones para recibir un mensaje de una fuente desconocida y con un número de enteros desconocidos como contenido. Primero detecta la llegada del mensaje utilizado MPI_PROBE. Después, determina la fuente del mensaje y utiliza **MPI_GET_COUNT** para conocer el tamaño del mensaje.Finalmente, direcciona un buffer par recibir el mensaje.

6.1.5. Modularidad.

Conocemos ya tres formas generales de composición que puede ser usadas en la construcción modular de programas paralelos, a saber, secuencial, paralelo y concurrente.

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. Este 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 procesos 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 solo si este 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.

A continuación se describen las funciones que permiten a los comunicadores ser usados más flexiblemente.

- **MPI_COMM_DUP:** Un programa puede crear un nuevo comunicador, conteniendo el mismo grupo de procesos pero con un nuevo contexto para asegurar que las comunicaciones generadas para diferentes propósitos no sean confundidas, este mecanismo soporta la composición paralela.
- **MPI_COMM_SPLIT:** Un programa puede crear un nuevo comunicador, conteniendo solo un subconjunto del grupo de procesos. Estos procesos pueden comunicarse entre ellos sin riesgo de tener conflictos con otros cómputos concurrentes. Este mecanismo soporta la composición paralela.
- **MPI_INTERCOMM_CREATE:** Un programa puede construir un intercomunicador, el cual enlaza procesos en dos grupos. Soporta la composición paralela.
- **MPI_COMM_FREE:** Esta función puede ser utilizada para liberar el comunicador creado al usar funciones anteriores.

6.1.6. Instalación.

El paquete rpm de LAM/MPI que se va ha instalar es el que nos provee el cd de Red Hat Linux 6.2, para llevar a cabo su instalación se realizará lo siguiente:

$>mount /mnt/cdrom
$>cd /mnt/cdrom/RedHat/RPMS/
$>rpm –ivh lam-6.3.1-4.i386.rpm

En el sitio web `http://www.lam-mpi.org/download/` se podrá obtener la última versión de dicho software.

6.1.7. Configuración.

En el directorio */usr/boot* crearemos los siguiente archivos con sus correspondientes configuraciones.

1. Archivo conf.lam, se recogerá la topología de red utilizada.

   lamd $inet_topo

2. Archivo bhost.def, se añadirá al final los nombres de todos los nodos que forman parte del cluster.

   localhost pc1 pc2 ...

Para testear 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.

6.2. Compilación y ejecución de programas LAM/MPI

Se generará 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 necesitará primero arrancar el sistem Multicomputador de Area Local, esto se hace a través del comando *lamboot*.

Para ejecutar el programa se utilizará el siguiente comando:

$>mpirun –np n^o de procesos nombre_programa argumentos

Capítulo 7

BWATCH. HERRAMIENTA SOFTWARE PARA OBSERVAR EL ESTADO DEL CLUSTER.

Contents

7.1. Introducción.

La herramienta *bWatch 1.0.3* es un pequeño programa TCL/TK, el cual muestra en una ventana un conjunto de información a cerca de cada uno de los nodos que conforma el cluster, es decir, es un sistema de monitorización del cluster.

La información que nos proveé dicho software es:

- Host name.
- Nº de usuarios conectados.
- Hola en la cual se hizo la toma de los datos.
- Carga hace 1 min.
- Carga hace 5 min.
- Carga hace 15 min.
- Numero de procesos.
- Memoria total.
- Memoria libre.
- Memoria compartida.

- Buffers.
- Cache.
- Total de swap.
- Swap libre.

La 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.

bWatch fue desarrollado para plataformas LINUX.

7.2. Instalación.

Una vez descargado el software *bWatch-1.0.3.tar.gz* se descomprime en el directorio de usuario *$HOME* y se compila. A continuación se muestra la secuencia de pasos a realizar:

$>/home/usuario/
$>tar -xvfz bWatch-1.0.3
$>cd /home/usuario/bWatch-1.0.3
$>make bWatch ⇒ genera un ejecutable llamado bWatch.tcl
$>make install ⇒ inserta bWatch en /usr/local/bin

7.3. Configuración.

Se editará el fichero *$HOME/.bWatchrc.tcl* y en la sentencia *set listOfHosts* se incluirá el nombre de los nodos que conforman el cluster

```
set bWatchrcVersion 1.0.0
set listOfHosts pc0 pc1

# set the columns which should be displayed
# 1 - information will be displayed for every host
# 0 - column will not be shown

# number of users on the system
set display(numUsers) 1
# system time
set display(time) 1
# 1 minute averga load
#set display(load1) 1
# 5 minutes average load
set display(load5) 1
# 15 minutes average load
set display(load15) 1
# number of processes
set display(numProcesses) 1
# total RAM
set display(totalMemory) 1
```

```
# available free RAM
set display(freeMemory) 1
# shared RAM
set display(sharedMemory) 1
# buffered memory
set display(buffers) 1
# cached memory
set display(cache) 1
# total swap space
set display(totalSwap) 1
# available swap space
set display(freeSwap) 1

# foreground and background coulours
# of the heading of the table
set colour(headingBG) #066
set colour(headingFG) #FFF

# foreground and background colours
# of the first column showing host names
set colour(hostNameBG) #006
set colour(hostNameFG) #FFF

set colour(neutralFG) black
set colour(firstWarningFG) orange
set colour(secondWarningFG) #A00
set colour(errorFG) #F00
set colour(errorBG) #FFF

set loadLimit(firstWarning) 1.5
set loadLimit(secondWarning) 2.5
set loadLimit(error) 5.0

set freeMemLimit(firstWarning) 1024
set freeMemLimit(secondWarning) 512
set freeMemLimit(error) 1

set freeSwapLimit(firstWarning) 20480
set freeSwapLimit(secondWarning) 10240
set freeSwapLimit(error) 5120
set blank ——-
set cell(width) 8
```

La herramienta bWatch obtiene los resultados de los distintos nodos a través del protocolo RSH, por lo tanto, en el fichero *$HOME/.rhosts* deberán estar incluidos los nombres de los nodos que conforman el cluster. Un ejemplo podría ser:

```
pc0
pc1
...
```

7.4. Visualización del estado del cluster.

Para poder visualizar el estado se realizará lo siguiente:

$> cd $HOME/bWatch-1.0.3
$> ./bWatch.tcl

A continuación nos aparecerá la siguiente tabla en la cual podremos observar el estado de cada uno de los nodos que conforma el cluster

Figura 7.1: Monitorización Cluster