Universidad Nacional de Luján

Departamento de Ciencias Básicas

Asignatura: Preparación y Evaluación de Proyectos

Tecnología de Clusters: Un estudio por niveles de implementación

Trabajo final para la obtención del título de Licenciado en Sistemas de Información

Autor: Efraim Wainerman

Director: Lic. Fernando Bordignon

Septiembre 2005

Índice de contenido

TECNOLOGÍA DE CLUSTERS: UN ESTUDIO POR NIVELES DE IMPLEMENTACIÓN

Resumen

El presente trabajo persigue mostrar el estado actual de la tecnología de clusters. Para ello, en una primera parte, se indagará en los objetivos generales de estos sistemas como así también los particulares de cada una de las áreas donde se desarrollan y se dará una taxonomía por niveles según los modos de implementación.

En una segunda parte se describen en profundidad tecnologías concretas desde sus aspectos arquitectónicos y de funcionamiento y se analizarán experiencias de laboratorio efectuadas con cada una de éstas. En particular, se estudiarán soluciones de software libre ya que por su naturaleza permiten ser analizadas en profundidad y a su vez pueden ser implementadas en ambientes académicos eliminando los costos de licencias.

1. INTRODUCCIÓN

A lo largo de la historia de la informática, el avance de las tecnologías relativas al hardware de computadoras ha permitido la aparición en el mercado de equipos con prestaciones cada vez mayores y más accesibles económicamente. Desde 1945 que se considera el comienzo de la computación moderna hasta mediados de la década pasada, el incremento de la relación costo/rendimiento se estima en 10^{11}. [1]

Sin embargo, a pesar de este crecimiento vertiginoso del poder computacional, los nuevos requerimientos que surgen hacen que en algunos casos la carga de trabajo sea mayor de la que puede soportar una unidad computacional individual. Estas dificultades han provocado que se avance en la búsqueda de soluciones basadas en el paralelismo. Desde esta perspectiva, se imponen dos modelos computacionales: el de los *sistemas fuertemente acoplados (*o *multiprocesador)* y el de los *sistemas débilmente acoplados (o distribuidos)*.

La primera solución se basa en la utilización de múltiples procesadores funcionando dentro de un único equipo compartiendo la memoria principal, el reloj, los periféricos y unidades de almacenamiento. La comunicación entre los procesadores se realiza a través de una memoria compartida.

En el segundo caso, los procesadores funcionan dentro de sistemas computacionales autónomos (también llamados nodos) que cooperan entre sí mediante el pasaje de mensajes. [2] [4]

Los sistemas multiprocesadores han sido ampliamente desarrollados durante la historia de la informática moderna, existiendo una gran consenso en el uso de estos sistemas y probada eficacia. A su vez, la implementación de este tipo de sistemas no requiere un cambio drástico en el software utilizado más allá del soporte del sistema operativo.

Con el surgimiento de Internet, millones de personas se fueron sumando a una red mundial. Esto provocó que muchas aplicaciones estuvieran disponibles a nivel global y también el surgimiento de nuevas aplicaciones como el comercio electrónico, el home banking, los motores de búsqueda, etc. A su vez, el avance de las comunicaciones provocó el acceso masivo a estos servicios. Hoy en día no es raro encontrar sistemas con miles e incluso millones de accesos diarios.

Estas aplicaciones plantean grandes exigencias en cuanto a tiempo de respuesta y disponibilidad de servicio. Por ejemplo, un sitio de comercio electrónico con miles de transacciones diarias pierde cientos o miles de transacciones ante la caída del sistema durante una hora. Por otro lado, un alto tiempo de respuesta podría hacer imposible en algunos casos la comunicación o eventualmente frustrar a los clientes potenciales perdiéndose muchas compras. En ambos casos tendríamos fuertes consecuencias negativas desde el punto de vista económico para la empresa que mantiene el sitio.

Los sistemas multiprocesadores se ven excedidos en su capacidad de escalabilidad (existen aplicaciones que exigen capacidades mayores que las que se pueden lograr a través de estas soluciones) y, más importante quizás, se observan relaciones de costo/rendimiento muy desventajosas (los costos ascienden en forma exponencial para un aumento lineal de rendimiento). Estos factores hacen que en muchos casos estas soluciones no sean implementables en la práctica.

Por otra parte, el surgimiento de las microcomputadoras, junto con el fenómeno de la computación masiva observado durante los últimos años, pone a nuestra disposición equipos de altas prestaciones a costos reducidos. Además, la tecnología de comunicaciones ha sufrido un avance vertiginoso, haciendo posible la interconexión de equipos con altas velocidades de transferencia y bajos retardos.

En este contexto, la surge la necesidad de aprovechar la capacidad computacional de las microcomputadoras para aplicaciones de altas exigencias, siendo las redes la herramienta fundamental para lograr este objetivo.

En la bibliografía, encontramos caracterizaciones acerca de las ventajas comparativas de los sistemas distribuidos con respecto a los sistemas multiprocesador: [1] [3]

- *Escalabilidad total*: se pueden implementar sistemas distribuidos que sobrepasen a los equipos autónomos mas grandes.
- *Escalabilidad incremental*: se puede comenzar con una configuración de pocos equipos cuando los requerimientos sean pequeños para luego ir incrementando el número de nodos a medida que vayan surgiendo mayores necesidades de rendimiento.
- *Confiabilidad*: puede implementarse tolerancia a fallas eliminando de la configuración los nodos que no estén funcionando. Se puede implementar con relativa facilidad redundancia entre múltiples equipos.
- *Costos*: dado que pueden utilizarse equipos disponibles en el mercado masivo, la relación costo/rendimiento de estos es mejor que la que se obtiene mediante equipos de mayores prestaciones.

Como desventajas o limitaciones de este modelo podemos encontrar las siguientes:

- *Software*: no existe consenso en una arquitectura común o que sirva para todos los casos en donde se requiera un sistema distribuido. Esto hace que se tenga que analizar puntualmente la aplicación que vaya a utilizarse o desarrollarse.
- *Redes*: en algunas aplicaciones los requisitos de rendimiento de las comunicaciones superan a los que ofrece la tecnología disponible o esta es muy costosa.

Durante la última década, surge y se hace muy popular el término *cluster* -o agrupación- para designar a sistemas distribuidos constituidos por computadoras autónomas conectadas mediante una red, generalmente de alta velocidad.

Estos fueron implementados en especial dentro de las universidades como alternativa económica

a las supercomputadoras para el estudio de sistemas distribuidos vivos como así también como herramienta de procesamiento distribuido con fines de investigación.

1.1. Objetivos de los clusters

Dada la diversidad de áreas de aplicación de los clusters, estos pueden hacer hincapié en distintos objetivos o cubrir parcialmente las necesidades de algunos de estos.

1.1.1. Transparencia

Una característica deseable en un cluster es que el acceso a los recursos que ofrece sea transparente a usuarios y aplicaciones. Esta propiedad también es conocida como imagen de sistema único o SSI por sus siglas en inglés (*Single System Image*).

Esto significa que no es necesario que las aplicaciones y usuarios conozcan la ubicación física de los recursos. Para lograr este objetivo es necesaria la unificación del acceso a los objetos del sistema operativo como por ejemplo, procesos, dispositivos de entrada/salida, información del sistema, mecanismos de comunicación, etc. Los sistemas existentes en general brindan transparencia en forma parcial, es decir, centrada en alguno de estos aspectos. [5]

1.1.2. Rendimiento

Para alcanzar este objetivo, un cluster debe aprovechar los recursos disponibles. Esto significa que debe lograr establecer una carga de trabajo proporcional en cada uno de sus nodos. Para ello es necesario el *balanceo de carga* entre estos.

Esta responsabilidad recae sobre el algoritmo de asignación de carga en los nodos. Una característica deseable de este algoritmo es que sea capaz de evaluar la carga individual de cada uno de estos y asignar la nueva carga que se genere a los servidores con menor carga. Otro mecanismo que puede utilizarse es la migración de carga desde nodos con mayor carga (o menor cantidad de recursos disponibles) hacia aquellos que tienen menor carga o mayor disponibilidad de recursos.

1.1.3. Alta Disponibilidad

Mediante este objetivo se busca que el cluster este operativo la mayor cantidad de tiempo posible (no es posible asegurar una disponibilidad total pero se intenta que el tiempo de caída del sistema sea mínimo). Para ello debe evitarse tener *puntos únicos de falla*, es decir componentes que al fallar provoquen la inoperatividad de todo el sistema.

Es necesario, entonces, que ante la falla de cualquier nodo del cluster, esta se pueda detectar y que otro equipo se haga cargo de la tarea que este desempeñaba en el menor tiempo posible.

Otro punto único de falla a tener en cuenta es la red, ya que si esta falla, todo el sistema pierde interconexión con la consecuente imposibilidad de operar. Para atacar este problema se pueden implementar enlaces redundantes.

1.1.4. Escalabilidad

Una característica inherente de los clusters es la escalabilidad, o sea la posibilidad de mejorar aspectos de rendimiento del sistema agregando nuevos equipos. Sin embargo, existen algunas limitaciones para lograr este objetivo.

Una limitante importante es el uso de algoritmos centralizados en nodos individuales. En estos diseños, la potencia del *cluster* se verá restringida a la capacidad de este de soportar la carga de trabajo.

Sería deseable que esto pudiera lograrse en caliente, es decir mientras el sistema se encuentra funcionando.

1.2. Áreas de aplicación

En la actualidad, los clusters son utilizados en los más diversos campos de aplicación. Podemos encontrar numerosos ejemplos de estos.

Procesamiento distribuido

Se trata de disminuir el tiempo requerido para resolver un problema computacional a través de la división del problema en subtareas para su ejecución en paralelo en los nodos que componen el cluster [38]. Algunos ejemplos son:

- Simulaciones: climatológicas [6], astrofísicas [9], aerodinámicas, de modelos socioeconómicos, etc.
- Búsqueda de vida inteligente en el espacio. [7]
- Entrenamiento de redes neuronales en inteligencia artificial. [8]
- Procesamiento de imágenes: satelitales, médicas, etc.
- Animación computada.

Aplicaciones comerciales

Las oportunidades que brinda el acceso a Internet de millones de personas abren las puertas a la aparición de nuevos negocios como así también su uso ligado a los sistemas de transacciones *on-line*. Algunos ejemplos son:

- Sitios de comercio electrónico
- Home banking
- Bases de datos distribuidas

Motores de búsqueda

Con la amplia difusión de la Web como medio de comunicación y la explosión de sitios de todo tipo, estos se han convertido en una herramienta fundamental para acercar los contenidos de a los usuarios. Algunos ejemplos son:

- Information Retrieval (IR)
- Indexación de documentos
- Servicio web
- Distribución de Contenidos

1.3. Tipos de clusters

Según la capa de software donde se implemente la solución de *clustering* podemos encontrar tres grandes categorías.

1.3.1. A nivel de aplicación

Una primera aproximación para la utilización de *clusters* para implementar sistemas distribuidos es desarrollar o modificar las aplicaciones para que utilicen los recursos disponibles en los nodos. Si bien esta es una solución posible, en este trabajo se analizarán aquellas soluciones tecnológicas de propósito general que puedan ser utilizadas para desarrollar o modificar aplicaciones existentes para el trabajo distribuido.

Existen diversas APIs *(Application Programming Interfaces)* para el desarrollo de sistemas distribuidos. En este caso, las aplicaciones deben ser desarrolladas específicamente utilizando alguna de estas. Ejemplos: MPI *(Message Passing Interface)*, PVM *(Parallel Virtual Machine)*, RMI *(Remote Method Invocation)*, CORBA *(Common Object Request Broker Architecture)*.

Otro enfoque consiste en la modificación transparente de las aplicaciones en tiempo de compilación. En este caso, el compilador agrega las rutinas y llamadas a funciones necesarias para que la aplicación pueda sacar provecho del *cluster*, la modificación del código fuente de la aplicación se realiza en forma automática. Un ejemplo de este tipo de solución es el proyecto Condor.

1.3.2. A nivel de sistema operativo

Este enfoque se basa en la migración transparente de procesos entre los nodos de la red. Una aplicación que ejecute tareas con un alto grado de independencia puede ser ejecutada en un Sistema Operativo con soporte de migración transparente de procesos. De esta forma, el núcleo es quien implementa la funcionalidad de balanceo de carga y tolerancia a fallas. Un ejemplo de este tipo de sistemas es MOSIX/OpenMOSIX.

1.3.3. A nivel de red

Muchas veces la aplicación que necesita aprovechar las ventajas del uso de un *cluster* es un servicio de red.

En la arquitectura cliente/servidor, normalmente se utiliza un equipo que cumple la función de servidor. Por otro lado, múltiples clientes pueden estar accediendo al mismo servidor en forma simultánea. Esto se realiza utilizando protocolos de comunicación bien definidos.

El servidor generalmente es administrado en forma centralizada por el proveedor del servicio, mientras que los clientes consisten en aplicaciones que se ejecutan en computadoras personales utilizando tecnologías heterogéneas.

En este contexto, mediante la modificación de la aplicación que implementa el servicio (por ejemplo las bases de datos replicadas) o bien por medio de la incorporación de dispositivos intermedios que realizan modificaciones en el flujo de los datos de la red, se puede lograr que la tarea se distribuya en un *cluster* de equipos. En particular, en el presente trabajo se hará hincapié en el segundo tipo de solución.

2. CASOS DE ESTUDIO

En este capítulo se analizarán implementaciones concretas de clusters.

En una primera etapa se explicarán las nociones acerca del funcionamiento de cada una de estas tecnologías. En una segunda, se describirá una experiencia de laboratorio realizada con cada una de ellas.

2.1. Caso 1: Linux Virtual Server

El proyecto *Linux Virtual Server* tiene por objetivo utilizar un conjunto de servidores de forma tal de que sean accedidos desde el exterior como si fuera un único servidor, el cual se conoce como *servidor virtual* o *redirector*.

Esto permite asignar numerosos equipos de pequeño tamaño y costo reducido conectados mediante una red local de alto rendimiento para que se distribuyan la carga de red que demanda el servicio. De esta forma, se logra un aumento casi lineal de la carga soportada por el servicio virtual con respecto a la capacidad que agrega cada equipo en forma individual.

2.1.1. Componentes

El principal componente del software es un conjunto de modificaciones (*patches*), que se aplican al código fuente del núcleo de Linux y que le agregan nuevas funcionalidades. Para ello, este nuevo código interactúa con las rutinas de manejo del protocolo TCP/IP.

En las distribuciones más recientes de GNU/Linux, es probable que la versión del kernel ya se encuentre con la funcionalidad compilada, ya que el código de Linux Virtual Server ya se encuentra incorporado en el código de las versiones actuales del núcleo de Linux. En general, las distribuciones ofrecen en su núcleo estándar compilado todas las opciones que se encuentran disponibles como módulos en el núcleo estándar.

A su vez, es necesario contar con la herramienta de usuario que interactúa con el núcleo modificado (*ipvsadm*). Normalmente, esta se encuentran incluida en la mayor parte de las distribuciones.

En caso de compilarse el kernel de Linux, se requieren en forma genérica que las siguientes opciones se encuentren activadas para compilación estática o en forma de módulos [14]:

Sección *"Networking Options"*:

- Packet socket
    - Kernel/User netlink socket
    - Routing messages
    - Netlink device emulation
    - Network packet filtering
    - Socket Filtering

- Unix domain sockets
- TCP/IP networking
- IP: multicasting
- IP: advanced router
- IP: policy routing
- IP: use netfilter MARK value as routing key
- IP: equal cost multipath
- IP: use TOS value as routing key
- IP: verbose route monitoring
- IP: large routing tables
- IP: tunneling

Sección "*IP: Netfilter Configuration*":

- Connection tracking (requerida por masq/NAT)
- FTP protocol support
- IP tables support (requerida por filtering/masq/NAT)
- limit match support
- MAC address match support
- netfilter MARK match support
- Multiple port match support
- TOS match support
- Connection state match support
- Packet filtering
- REJECT target support
- Full NAT
- MASQUERADE target support
- REDIRECT target support
- Packet mangling
- TOS target support
- MARK target support
- LOG target support

Estos parámetros están asociados a componentes que ya son estándar en el núcleo de Linux y por lo tanto se pueden utilizar para otro tipo de aplicaciones y también podrán variar de acuerdo a las necesidades específicas de la aplicación que se vaya a utilizar.

Tanto si los *patches* ya hubiesen sido aplicados al código fuente del kernel de Linux como si se utilizara una versión que ya incorpore estas modificaciones, se encontrará disponible la sección "*IP: Virtual Server Configuration"*. Las opciones para esta sección pueden ser activadas en su totalidad sin ningún problema y son las siguientes:

Sección "*IP: Virtual Server Configuration"*:

- virtual server support
- IP virtual server debugging
- IPVS connection table size = 12 (por defecto)
- round-robin scheduling
- weighted round-robin scheduling
- least-connection scheduling scheduling
- weighted least-connection scheduling
- locality-based least-connection scheduling
- locality-based least-connection with replication scheduling
- destination hashing scheduling
- source hashing scheduling
- FTP protocol helper

Las opciones de esta sección están asociadas en su mayoría con el código asociado a la implementación de los diversos algoritmos de planificación (*scheduling*) que soporta el sistema y es un área donde permanentemente se hacen avances agregándose nuevos métodos para balancear la carga en los servidores reales [11].

Otro componente que se desarrolla dentro del proyecto de LVS es la herramienta ipvsadm que, ejecutándose a nivel de usuario, permite la configuración del servidor virtual. Mediante esta utilidad se modifican los parámetros de funcionamiento del software del servidor virtual que se ejecuta en el núcleo del sistema operativo.

La ejecución puede hacerse mediante el intérprete de comandos interactivo o bien dentro de un script. Además se cuenta con los scripts ipvsadm-save e ipvsadm-restore que permiten guardar la configuración de los parámetros actuales del sistema en un archivo y restaurarlos respectivamente.

Es necesario para el correcto funcionamiento del sistema que estos componentes se encuentren disponibles en el/los redirectores. Sin embargo no se necesitan modificaciones en las aplicaciones clientes y servidoras, ya que las primeras perciben al sistema como un único servidor y, por otro lado, estas últimas perciben los requerimientos que le llegan como si vinieran directamente del cliente.

De esta forma, se requiere que el/los redirectores ejecuten el sistema operativo Linux, sin embargo no existen requisitos especiales de software para clientes y servidores, excepto para las configuraciones utilizando *IP Tunneling y Direct Routing,* donde se requerirá una configuración especial del sistema operativo, aunque no restringiéndose únicamente a Linux (ver la siguiente sección).

2.1.2. Arquitectura

La arquitectura planteada por LVS es la de un equipo funcionando como redirector de las conexiones entrantes al sistema distribuyéndolas de acuerdo a un algoritmo de planificación determinado en un grupo de servidores. Esto permite que los distintos equipos compartan la carga de trabajo, pudiéndose escalar el sistema simplemente agregando nuevos servidores a la red.

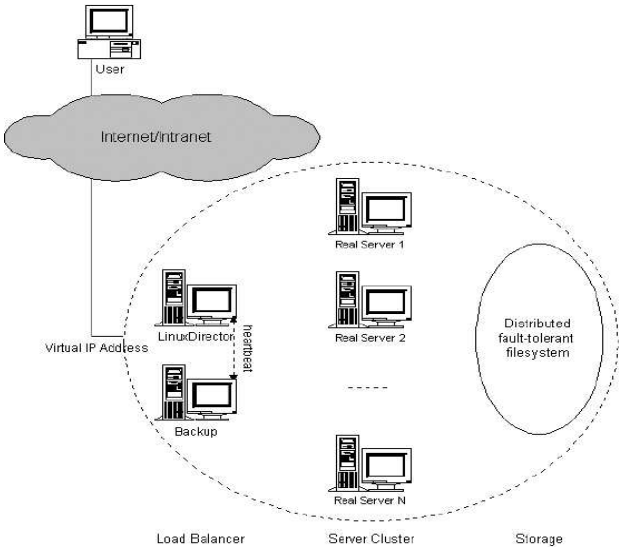

Figura 1. *Arquitectura de tres capas de Linux Virtual Server. Extraído de Linux Virtual Server for Scalable Network Services* [10].

Como se ve en la figura 1 el sistema utiliza una arquitectura de tres capas formadas por el/los redirectores, los servidores reales y el sistema de archivos compartido (aunque este último componente no es estrictamente indispensable).

2.1.3. El Redirector

Es visto desde el exterior como el servidor ya que tiene asignada la dirección IP con que se accede al servicio. Puede recibir múltiples conexiones y redirigirlas a los servidores reales según el o los criterios de balanceo de carga que se hallan adoptado.

Dado que la aplicación servidora no se ejecuta en este equipo y que el software encargado de redireccionar las conexiones se ejecuta en el núcleo de su sistema operativo, esto reduce drásticamente el overhead de procesamiento, con lo cual es capaz de atender muchas más conexiones que las que podría atender si las conexiones se hicieran directamente a este equipo, manteniendo los tiempos de respuesta.

2.1.4. Los servidores reales

Es el conjunto de servidores en cada uno de los cuales se ejecuta la aplicación. Esta puede ser cualquiera que utilice el protocolo TCP/IP. Sin embargo en el caso de que el servidor tenga que iniciar la apertura activa de conexiones con el cliente -por ejemplo FTP-, esto ocasiona que el

servicio no funcione. La solución a este problema consiste en que el redirector inspeccione los paquetes a nivel de aplicación, lo cual se implementa a través de la incorporación de módulos específicos en el redirector que implementan el manejo de la aplicación específica.

2.1.5. El repositorio de datos compartido

Aunque este es un componente opcional dentro de la arquitectura del sistema, provee la facilidad de poder administrar los datos en forma centralizada.

Este repositorio común de datos podría no estar presente en el sistema, utilizándose un acceso local a los datos por parte de los servidores reales, pero de esta manera se haría muy dificultoso el proceso de modificar los datos de la aplicación. Un ejemplo de estos datos son los archivos html en un servidor Web.

Según la aplicación de que se trate, el rol de repositorio de datos compartido podrá ser cumplido por un distintos elementos, como ejemplo podemos dar un sistema de archivos de red o un servidor de base de datos.

Dado que ahora todo el sistema depende de él, el uso de un repositorio nos plantea el problema de transformarse potencialmente en un cuello de botella. Es conveniente entonces que este componente sea escalable. Esto puede lograrse mediante el uso de sistemas de archivos o bases de datos distribuidas.

A su vez, otro problema a tener en cuenta es que el repositorio pasa a ser un punto único de falla. Es necesario para no perjudicar la disponibilidad del sistema que este componente sea tolerante a fallas.

Otro aspecto a tener en cuenta al implementar el sistema de archivos compartido es que, en caso de que la aplicación realizara cambios en los datos, es necesario contemplar el manejo de bloqueos, ya que las aplicaciones que se ejecutan en los servidores reales accederán a los datos en forma concurrente y es fundamental para la coherencia de los datos que no se permita a dos aplicaciones modificar un mismo dato compartido al mismo tiempo. Este esquema puede estar incorporado en el sistema de archivos compartido o puede ser manejado por alguna aplicación específica como por ejemplo un servidor de base de datos.

Ejemplos de este tipo de sistemas de archivos en Linux son GFS [19], Coda [20] e Intermezzo [21].

2.1.6. Alta Disponibilidad

Aunque el software del proyecto LVS no provee un juego de herramientas completo para el manejo de alta disponibilidad, se pueden utilizar herramientas libres desarrolladas para brindar una infraestructura estándar para cualquier tipo de servicio [12] [13]. Haciendo uso de estas herramientas se puede monitorear el estado de los servidores reales para que, en caso de falla de alguno de ellos, simplemente se elimine de la lista que mantiene el redirector y el sistema siga funcionando normalmente.

Para evitar la caída del sistema en caso de falla del redirector principal se utiliza un equipo de respaldo que cuando detecta una falla en el primero toma su lugar apoderándose de su dirección IP. Esto se realiza utilizando la técnica de arp-spoofing [18].

La detección tanto de la caída de un servidor real como del redirector por parte del servidor de respaldo se realiza a través del monitoreo constante a través de demonios específicos como por ejemplo *hearbeat*[1] ejecutándose en el redirector principal y el de respaldo respectivamente.

2.1.7. Técnicas de balanceo de carga

Como se mencionó anteriormente, Linux Virtual Server permite la implementación de un servicio virtual a través del uso de un balanceador de carga para una red formada por los servidores reales que ejecutan la aplicación.

En esta arquitectura las peticiones de servicio de los clientes son enviados hacia el balanceador de carga. El primer paquete que le llega a este es pedido de apertura de conexión (en el caso de TCP es un segmento con el *flag* SYN activado), el cual tiene como destino la dirección IP que el redirector tiene asignada para el servicio virtual. El redirector examina la dirección IP y el puerto de destino, si coincide con alguno de los servicios publicados según la tabla de servicios virtuales elige un servidor real para ese servicio mediante el algoritmo de planificación que haya sido seleccionado para ese servicio.

Una vez que el servidor real ha sido seleccionado, se agrega la conexión a la tabla correspondiente y se reenvía el paquete al servidor real correspondiente haciendo uso de alguna de las tres técnicas de balanceo de carga que se describen en las siguientes secciones. La técnica a utilizar dependerá del servidor real elegido, ya que este parámetro se puede configurar independientemente para cada servidor real.

Los paquetes que lleguen posteriormente y que pertenezcan a esta conexión serán identificados mediante la tabla de conexiones y luego reenviados al servidor real correspondiente.

Al recibir los paquetes al servidor real, este los percibe como provenientes del cliente, los procesa y envía la contestación al cliente. La forma en que es enviada esta información al cliente varía según la técnica de balanceo de carga utilizada, pero la aplicación percibirá siempre que está enviando los datos directamente al cliente.

2.1.8. LVS a través de NAT (Network Address Translation)

Esta técnica consiste en la modificación de los encabezados de los datagramas IP con el objeto de reemplazar sus direcciones IP de fuente (*source NAT*) o destino (*destination NAT*) realizando el mapeo de un grupo de direcciones a otro [17].

Un tipo especial de source NAT conocido como enmascaramiento (*masquerading*) es ampliamente utilizado en las redes actuales principalmente para dar acceso a Internet a equipos que utilizan direcciones IP privadas y que no podrían hacerlo directamente. Se utiliza entonces un dispositivo intermedio actuando como gateway que posee una dirección pública de Internet el cual

1 Estos demonios envian mensajes que pueden ser tipo ping o bien de prueba del servicio a intervalos regulares y si no reciben respuesta pueden proceder a eliminar un servidor real de la lista que se encuentra en el redirector o pueden iniciar la toma de posesión de la dirección IP del balanceador de carga para tomar su lugar.

modifica los paquetes provenientes de las estaciones de trabajo asignándoles como fuente su propia dirección IP para que los datagramas puedan atravesar la red de Internet.

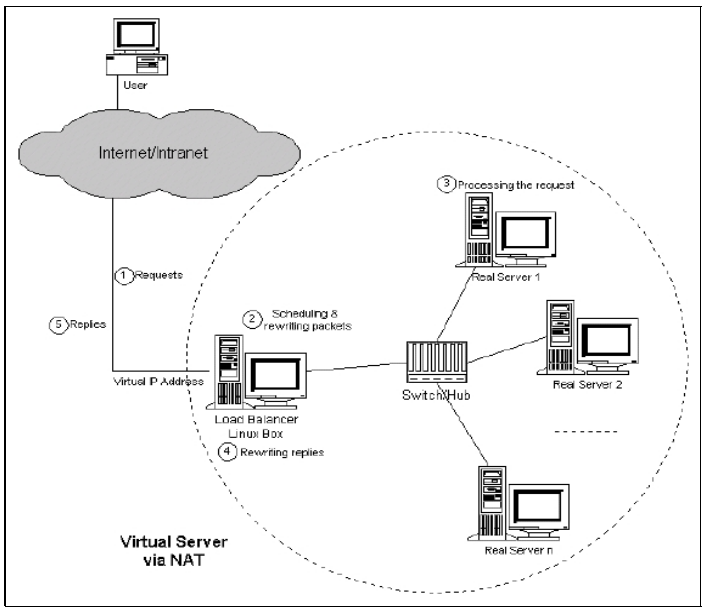

Figura 2. *Arquitectura de Linux Virtual Server mediante NAT. Extraído de* ***Linux Virtual Server for Scalable Network Services***. [10]

Una extensión a NAT es la técnica conocida como *network address port translation* la cual se basa en el mapeo de direcciones de red y puertos TCP/UDP de los paquetes en otros.

Esta última técnica permite que numerosos servicios ejecutándose en distintas direcciones de red y *"escuchando"* en distintos puertos puedan ser trasladados a un único puerto en la dirección de red del servidor virtual para lograr la apariencia de un único servicio.

En la figura 2 se puede observar una configuración típica de Linux Virtual Server a través de NAT en la que los servidores reales y el balanceador de carga se hallan interconectados a través de un Switch/Hub utilizando una topología de estrella. Esta es una configuración que utiliza en forma eficiente el ancho de banda disponible en la red al separar los datos a enviarse a los servidores reales en sus respectivos canales.

El reenvío de los paquetes paquetes provenientes de los clientes mediante NAT se realiza modificando los campos de dirección IP y puerto de destino de los mismos por los correspondientes a la configuración del servidor real escogido para recibir la petición.

Una vez que los servidores reales procesan la petición, envían su respuesta al cliente. Debido a que los servidores reales en esta configuración utilizan como gateway al balanceador de carga, la respuesta será enviada a través de este. El mismo entonces reescribe la dirección y puerto de origen de manera de que coincidan con los correspondientes al servicio virtual y lo reenvía al cliente.

La configuración de un servidor virtual mediante NAT es muy transparente, ya que no requiere modificaciones ni configuraciones especiales en los sistemas operativos de los servidores reales, lo cual facilita su implementación.

Como desventaja de esta técnica se puede mencionar la restricción de que los servidores reales se

deban encontrar en la misma red que el balanceador de carga, sin embargo esto normalmente no representa un impedimento grave dado que esta es una configuración estándar.

Sin embargo, la principal desventaja de esta técnica es su escalabilidad, debido al overhead en que se incurre al tener que reescribir todos los paquetes de datos originados por los servidores reales que se hace notorio cuando el número de estos últimos es muy grande. Este efecto adverso puede disminuirse utilizando un equipo más potente como redirector.

La solución a este problema consiste en que los servidores reales envíen los paquetes de datos de salida directamente hacia los clientes sin pasar por el redirector a través de las técnicas que se describen en las siguientes secciones.

2.1.9. LVS a través de IP Tunneling

La técnica de IP Tunneling consiste en encapsular un datagrama IP dentro de otro. Es posible entonces que el balanceador de carga encapsule los datagramas IP provenientes de los clientes dentro de otros datagramas IP que son enviados hacia los servidores reales logrando que estos luego envíen los paquetes de respuesta directamente hacia los clientes.

En la figura 3 se muestra la arquitectura de Linux Virtual Server utilizando la técnica de IP Tunneling. Para el reenvío de los paquetes, el redirector encapsula los datagramas con dirección IP del cliente como origen y destino en la dirección IP del servicio virtual dentro de datagramas con dirección IP del redirector como origen y dirección IP del servidor real como destino.

Estos nuevos paquetes pueden dirigirse directamente hacia los servidores reales o ser reenviados por dispositivos intermedios como routers, atravesando varias redes IP en el camino. Esto se logra porque estos paquetes son vistos por cualquier dispositivo de red como paquetes IP normales.

Una vez que el paquete llega al servidor real, este lo desencapsula y comprueba que está dirigido hacia la dirección IP virtual configurada en su dispositivo de tunneling, así que procesa el datagrama y luego envía la respuesta directamente hacia el cliente.

La configuración de Linux Virtual Server a través de IP Tunneling es la más flexible respecto a la configuración física de la red, no se tienen restricciones en cuanto a la ubicación física de los servidores reales, los cuales podrían no encontrarse en la misma red local.

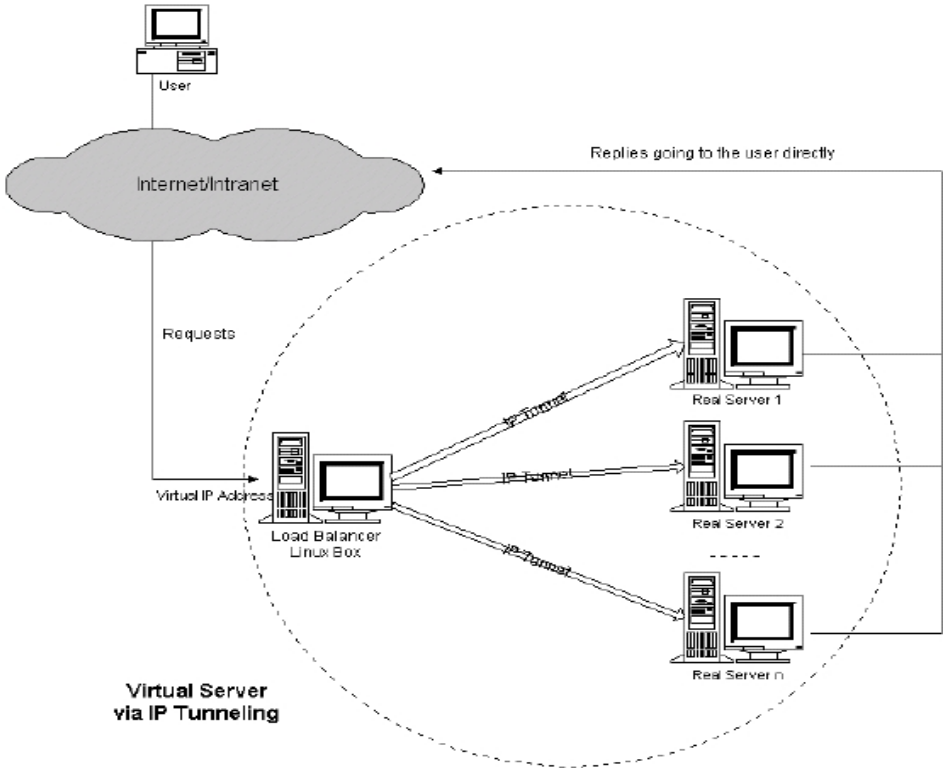

Figura 3. *Arquitectura de Linux Virtual Server mediante IP Tunneling. Extraido de Linux Virtual Server for Scalable Network Services* [10].

Por otro lado, esta técnica impone la restricción de que los sistemas operativos de los servidores reales deben soportar el mecanismo de IP Tunneling y estar configurados apropiadamente.

Utilizando esta técnica se evita el overhead que representa realizar NAT en el redirector para los paquetes de salida. Esto multiplica la escalabilidad del sistema para prácticamente todas las aplicaciones cliente/servidor ya que normalmente el volumen de los datos que envía el cliente al servidor es significativamente menor con respecto a los que el servidor envía al cliente.

De esta forma el límite en cuanto a cantidad de servidores reales que pueden agregarse al sistema obteniendo un aumento lineal de rendimiento es mucho más alto que el que se tiene utilizando NAT.

2.1.10. LVS a través de Direct Routing

Esta técnica de balanceo de carga utiliza una red LAN para la interconexión de los servidores y el balanceador de carga con la particularidad de que sus interfaces de red deben estar interconectadas utilizando un segmento ininterrumpido de la red como por ejemplo un Switch/HUB (ver fig. 4).

En esta configuración, todos los servidores reales tienen un dispositivo de red al cual se ha asignado la dirección IP del servidor virtual. Este dispositivo podrá ser cualquiera que pueda ser configurado en el servidor, típicamente el dispositivo de loopback, pero para que el sistema funcione correctamente el mismo no deberá responder a los mensajes ARP, ya que esto ocasionaría conflictos de red al utilizar todos los equipos una única dirección (ver *the arp problem* en *LVS HOWTO* [6]).

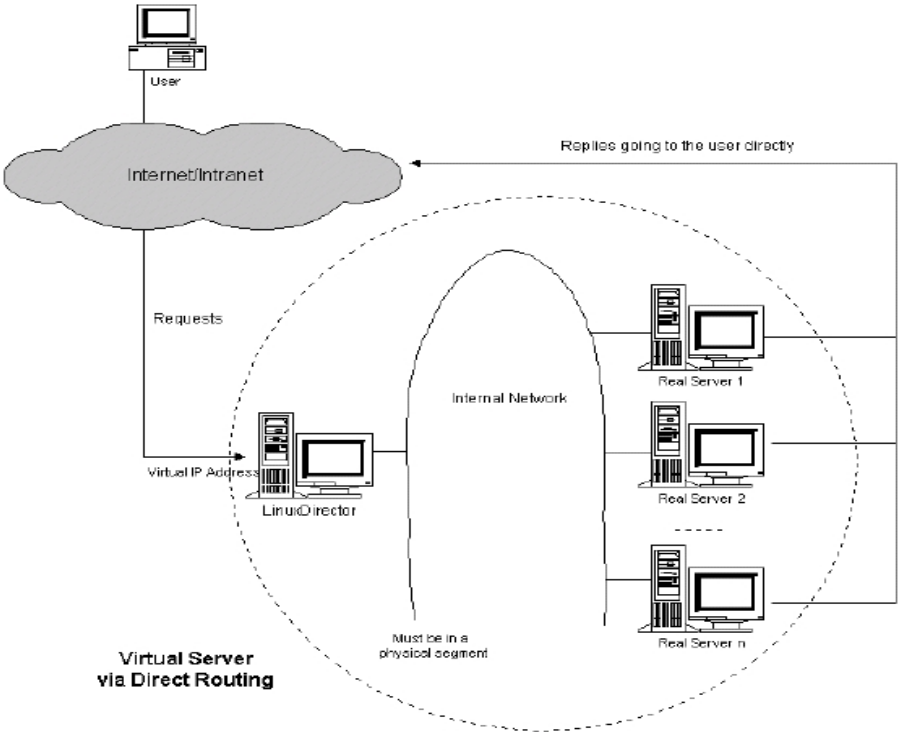

Figura 4. *Arquitectura de Linux Virtual Server mediante Direct Routing. Extraido de Linux Virtual Server for Scalable Network Services [10].*

Para realizar el reenvío de los paquetes, el balanceador de carga sólo cambia las direcciones MAC para direccionar los paquetes al servidor real elegido mientras que la dirección IP de destino sigue siendo la del servicio virtual. Los servidores reales pueden aceptar los paquetes IP entrantes ya que tienen un dispositivo con esa dirección IP asignada. Luego procesan el requerimiento y envían la respuesta directamente hacia el cliente.

La configuración de Linux Virtual Server a través de Direct Routing impone la restricción de que los dispositivos de red que tengan asignada la dirección del servicio virtual deban no responder a las peticiones de ARP, lo cual puede no ser un problema trivial en algunos sistemas operativos (curiosamente esto se da en los kernels 2.2.x y 2.4.x de Linux, ver *LVS HOWTO* [15]).

Por otro lado esta técnica, al igual que la de IP Tunneling, multiplica la escalabilidad del sistema al descargar del balanceador de carga la tarea de redireccionar los paquetes de datos de los servidores a los clientes, pero como ventaja adicional con respecto a IP Tunneling cuenta con el hecho de que se elimina el overhead derivado del encapsulamiento/desencapsulamiento de los datagramas IP.

2.1.11. Algoritmos de planificación

Linux Virtual Server implementa distintos algoritmos de planificación para que el redirector pueda decidir a que servidor real asignar una nueva conexión entrante. El objetivo de éstos es que la carga entre los servidores reales permanezca balanceada y que los clientes tengan el menor tiempo de respuesta a sus peticiones.

Los algoritmos que se encuentran implementados en Linux Virtual Server son:

- ***Round Robin***: mediante este algoritmo se asignan las conexiones entrantes hacia los servidores reales secuencialmente entre los nodos referenciados en la lista siguiendo el orden en que se encuentran. Una vez que se ha llegado al último nodo en la lista se empieza por el primero nuevamente.
- ***Weighted Round Robin***: este algoritmo agrega al anterior la característica de que sea asigna un peso que consiste en un número entero a cada nodo en la lista de servidores reales que refleja el poder de procesamiento de cada uno de estos. El valor por defecto es 1. De esta forma, si el nodo A tiene asignado un peso de 1 y el nodo B tiene un peso de 2, el nodo B recibirá el doble de conexiones que el nodo A. En este algoritmo no asegura que la cantidad de conexiones del nodo B será el doble que el nodo A, ya que no evalúa la cantidad de conexiones que tiene el nodo en un momento dado sino que sólo tiene en cuenta cómo se asignarán las nuevas conexiones.
- ***Least Connection***: las nuevas conexiones se asignan al servidor real con menor cantidad de conexiones asignadas. Este parámetro permite tener una aproximación bastante fidedigna de la carga que tiene un servidor en un momento dado pero no es un número totalmente certero ya que la ciertas conexiones pueden ocasionar un costo computacional más altas que otras y además el nodo puede estar procesando otras tareas que no están relacionadas con las conexiones asignadas por el redirector.
- ***Weighted Least Connection***: busca el servidor con menor cantidad de conexiones haciendo una ponderación por poder computacional.
- ***Locality-Based Least-Connection Scheduling:*** asigna las conexiones entrantes siempre al mismo servidor siempre y cuando no este sobrecargado (el número de conexiones es mayor que su peso). En caso contrario utiliza el algoritmo de *weighted least connection*. Generalmente este algoritmo es utilizado para servidores de cache ya que de esta forma aumenta la probabilidad de que los objetos incluidos en las peticiones se encuentren en las unidades de almacenamiento de los nodos por haber sido pedidos con anterioridad.
- ***Locality-Based Least-Connection with Replication***: este algoritmo, al igual que el anterior está optimizado para servidores de cache. Se mantiene una asociación entre objetivos y conjuntos de servidores que pueden servir ese objetivo. Las peticiones para un objetivos son derivadas al nodo con menor cantidad de conexiones dentro del conjunto de servidores para ese objetivo. Si todos los nodos del conjunto están sobrecargados, se elige el nodo con menor cantidad de conexiones del cluster y se agrega al conjunto de servidores para ese objetivo. Si el conjunto de servidores no se ha modificado por un periodo de tiempo, se elimina el nodo con mayor número de conexiones del conjunto para eliminar un alto grado de replicación.
- ***Destination Hashing***: una conexión entrante se asigna estáticamente a un servidor dado una función hash que se calcula en base a la dirección IP de destino de la petición.
- ***Source Hashing***: una conexión entrante se asigna estáticamente a un servidor dado una función

hash que se calcula en base a la dirección IP de origen de la petición.

2.1.12. Experiencia de Laboratorio: Implementación de un Servidor Web Virtual mediante NAT

Para realizar la experiencia práctica de implementación de un servidor virtual en linux se optó por configurar un servicio http, ya que este es de fácil configuración y, en su modo de operación básica, no presenta problemas para ser implementado mediante un servicio virtual.

2.1.12.1. Software utilizado

El principal componente de software que se usó para realizar la experiencia fue el código encargado de ejecutar el servicio de balanceo de carga que provee Linux Virtual Server, el cual se integra al núcleo del sistema operativo linux. En el caso de la experiencia realizada se optó por una solución a través de núcleo precompilado utilizando el que se halla incluido en la distribución *Linux Mandrake 8.1* (cuya versión es la 2.4.8)[2].

Para poder utilizarse la funcionalidad presente en el núcleo de linux es necesario contar con la utilidad a nivel de usuario *ipvsadm* que es la que se encarga de agregar los servicios virtuales y servidores reales a la lista que maneja el módulo de software que se ejecuta en modo kernel. Esta utilidad también puede descargarse gratuitamente de la página principal del proyecto *Linux Virtual Server*. La que se utilizó en la experiencia fue la incluida en la distribución de Linux Mandrake 8.1.

Para completar los requisitos de software en el servidor virtual fue necesario utilizar un componente de software capaz de realizar NAT, ya que esta fue la técnica utilizada para implementar el servidor virtual. En este caso, el software del proyecto LVS se integra con el que ya se provee en forma estándar dentro del núcleo de Linux para manejo de NAT (la infraestructura *Netfilter* [17]).

Ambas versiones del componente de NAT de Linux, en forma similar al software de LVS, constan de una parte que se integra al núcleo de Linux y una utilidad que se ejecuta en modo usuario para la administración de las reglas de NAT, la utilidad para la administración de *Netfilter* es *iptables*. [16]

El software que se utilizó como aplicación servidora fue el servidor web Apache. En este caso también se utilizaron las versiones del software incluidas con las respectivas distribuciones, es decir la versión 1.3.14 para la PC 15 con Conectiva Linux mientras que en la PC 16 con Suse Linux se utilizó la versión 1.3.17 del producto.

Por último el cliente utilizado fue Netscape Navigator versión 4.76 ejecutándose en Windows NT Server 4.0.

2.1.12.2. Configuración de la red

Para realizar la experiencia se utilizaron cuatro computadoras personales (Pcs) conectadas mediante una red local Ethernet (de 10 Mbps). Adicionalmente se utilizó una quinta computadora

2 Véase la sección 2.1.1 para más detalles acerca de este componente de software.

para realizar la captura de las tramas enviadas por la red LAN durante la prueba.

La conexión física consistió en un único segmento de red que utiliza un hub para la interconexión en bus de los equipos. La topología de red es la que se muestra en la figura 5.

Como puede advertirse, el cliente y el servidor virtual utilizan una configuración en donde ambos pertenecen a la misma red IP 170.210.122.0/25, es decir que pueden alcanzarse sin necesidad de utilizar nodos intermedios. Aunque esto no es un requisito indispensable, sí es necesario que exista una ruta por la cual los paquetes puedan transmitirse entre estos dos equipos. A su vez la dirección IP del servidor virtual es una dirección públicamente conocida –y eventualmente, aunque este no es el caso, registrada con un nombre de dominio–.

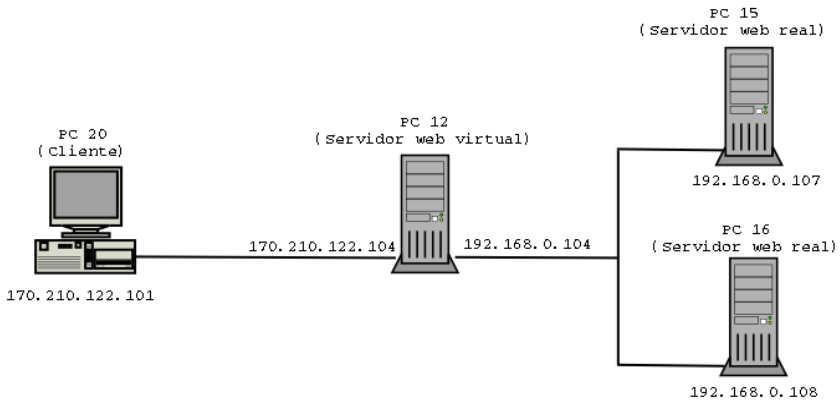

Figura 5. *Topología de la red que se utilizó para la implementación del servidor virtual.*

Para los servidores reales se definió la red IP privada 192.168.0.0/25.

La configuración de hardware y software de cada una de las máquinas se muestra en la tabla 1.

Tabla 1. Configuración de hardware y software de las máquinas utilizadas para realizar la experiencia.

Nombre	***CPU***	***Memoria***	***Interfaces de red***	***Dirección IP***	***Sistema Operativo***
PC12	Pentium 233 MMX	64 MB	eth0: placa de red ethernet de 10 mbps PCI. eth0:0: interface de red virtual	170.210.122.104 192.168.0.104	Mandrake Linux 8.1
PC15	Pentium 233 MMX	64 MB	eth0: placa de red ethernet de 10 mbps PCI	192.168.0.107	Suse Linux 7.1
PC16	Pentium 233 MMX	64 MB	eth0: placa de red ethernet de 10 mbps PCI	192.168.0.108	Conectiva Linux 6.0

Nombre	*CPU*	*Memoria*	*Interfaces de red*	*Dirección IP*	*Sistema Operativo*
PC20	Pentium 233 MMX	64 MB	eth0: placa de red ethernet de 10 mbps PCI	170.210.122.101	Windows NT Server 4.0

2.1.12.3. El cliente

Este equipo no cuenta con ninguna configuración especial mas que un navegador web para realizar el requerimiento http al servidor virtual. Cabe mencionar sin embargo que se modificó la configuración de red del navegador para que no hiciera los requerimientos mediante proxy como en su configuración normal, sino que por el contrario las realizase mediante conexión directa con el objeto de hacer más claras las capturas de red.

2.1.12.4. El redirector

Este equipo posee una interfaz de red tipo Ethernet referenciada por el sistema como eth0 a la cual se ha asignado la dirección IP pública 170.210.122.104. Esta dirección está disponible para ser accedida desde cualquier máquina en Internet.

Sobre esta interfaz física se ha configurado una interfaz virtual denominada eth0:0 a la cual se ha asignado la dirección IP privada 192.168.0.104 para poder efectuar la comunicación con los servidores reales.

Dicha configuración se realizó utilizando la herramienta *linuxconf*.

Luego de realizada la configuración básica de la red, se agregaron las reglas de funcionamiento de LVS y NAT mediante el script para configuración del firewall que en la distribución utilizada se encuentra en /etc/rc.d/rc.firewall. El contenido de este archivo se muestra en el anexo 1.

La primera línea del script establece cual será el intérprete de comandos a utilizar, en este caso bash.

```
#!/bin/bash
```

A continuación se activa el reenvio de paquetes, es decir la posibilidad de rutear paquetes de red que le lleguen al redirector y cuya dirección IP de destino no sea este equipo.

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

A continuación se describen las líneas del script utilizadas para realizar la configuración de la

funcionalidad de NAT en el equipo. Esto es necesario debido a que la técnica de balanceo de carga elegida fue justamente esta última.

Debido a que en la configuración del núcleo utilizado el código encargado del manejo de NAT se encuentra compilado en forma de módulo, este debe ser cargado para poder usarse. En la siguiente línea se carga dicho módulo [16].

```
modprobe iptable_nat
```

Luego se establece cual será el comando encargado de realizar la configuración de los parámetros del módulo. En este caso sólo se utilizará para la configuración de las funciones de NAT.

```
IPTABLES=/sbin/iptables
```

Se define la red local donde se encuentran los servidores reales. Se usa la dirección 192.168.0.0 y la máscara de red 255.255.255.128.

```
REDLOCAL=192.168.0.0/25
```

A continuación se limpia cualquier configuración anterior que pudiera encontrarse en memoria.

```
$IPTABLES -F
```

En la siguiente línea se agrega una regla a la tabla POSTROUTING [17] que indica que a cualquier datagrama IP que llegue desde cualquier nodo perteneciente a la red local debe aplicársele la técnica de masquerade (ver sección 2.1.8).

```
$IPTABLES -t nat -A POSTROUTING -s $REDLOCAL -j MASQUERADE
```

Adicionalmente debe agregarse esta línea para que se acepte el ruteo de los datagramas originados en la red local.

```
$IPTABLES -A FORWARD -s $REDLOCAL -j ACCEPT
```

En la última sección del script se configuran las reglas del servidor virtual propiamente dicho.

Para esto se define en primer lugar cual será el comando encargado de modificar los parámetros de funcionamiento del módulo LVS. Se configura con ipvsadm, utilidad incluida con el software de

LVS.

IPVSADM=/sbin/ipvsadm

Luego se limpia cualquier configuración previa del módulo que pudiera encontrarse en memoria.

$IPVSADM -C

Por último se define el servicio virtual que se va a poner en funcionamiento.

En este caso se especificó un servicio virtual en la dirección IP 170.210.122.104 y puerto 80 con un algoritmo de planificación de round robin. Todo datagrama IP dirigido a la dirección de red 170.210.122.104 y que contenga un segmento TCP con dirección de puerto 80 ya no será procesado por el código normal de TCP/IP sino que será "interceptado" por el código de LVS que se ejecuta dentro del núcleo.

$IPVSADM -A -t 170.210.122.104:80 -s rr

Por último se agregan al servicio virtual definido anteriormente dos servidores reales -con direcciones IP 192.168.0.107 y 192.168.0.108 respectivamente-. El parámetro -t especifica que este es un servicio tipo TCP, mientras que -m especifica que el balanceo de carga se hará mediante masquerade -es decir NAT-. Cabe aclarar que el puerto donde aceptan peticiones no necesariamente tiene que ser coincidente con el del servicio virtual.

$IPVSADM -a -t 170.210.122.104:80 -r 192.168.0.107:80 -m

$IPVSADM -a -t 170.210.122.104:80 -r 192.168.0.108:80 -m

Es importante notar que dada la forma en que se definen los servicios virtuales se permiten tener múltiples servicios virtuales utilizando cada uno diferentes algoritmos de planificación. Por otro lado, tal como se definen los servidores reales pueden usarse distintas técnicas de balanceo de carga con cada uno de ellos (por ejemplo podría tenerse algún servidor real que se encontrase fuera de la red local haciendo necesario el uso de IP Tunneling).

2.1.12.5. Los servidores reales

En los mismos se estableció como router por defecto la dirección 192.168.0.104 correspondiente a la interfaz virtual eth0:0 del redirector. La configuración se realizó utilizando las herramientas de configuración del sistema propias de cada una de las distribuciones de linux sobre las que se montaron los servidores web (YaST en SuSE y Linuxconf en el caso de Conectiva).

Por último hay que mencionar que ejecutan un servidor web apache con una configuración estándar atendiendo a los requerimientos http en su puerto 80.

2.1.12.6. Análisis del funcionamiento de LVS a través de las capturas de red

En esta sección se mostrará un caso práctico del funcionamiento del sistema a través del comentario de las capturas de red que se tomaron durante una sencilla prueba realizada utilizando la implementación que se describió durante el transcurso de esta sección. La misma consistió en realizar una petición http de una página web conteniendo texto y con tamaño lo suficientemente pequeño como para poder ser transportado en una sola trama ethernet para no generar demasiadas tramas que sólo entorpecieran el análisis de la captura.

Las capturas de red se presentan completas en el Anexo I, y se hallan numeradas para ser referenciadas en el comentario que se desarrolla en la siguiente sección.

2.1.12.7. Descripción de las tramas

La primera trama ethernet corresponde al pedido ARP que realiza el cliente al servidor virtual. Cabe aclarar que esto es así debido a que tanto el cliente como el servidor virtual se hallan en el mismo segmento de red. En un caso típico este pedido ARP podría ser realizado por un router de entrada a la red local. El objetivo de esta trama es obtener la dirección ethernet del servidor virtual conociendo su dirección IP.

La trama 2 es la contestación que realiza el servidor virtual dando a conocer la dirección física de la interfaz de red donde presta servicio.

La trama 3 lleva un paquete IP que a su vez contiene un segmento TCP de apertura de conexión (flag SYN activado) enviado desde el cliente hacia el servidor virtual. El puerto destino para realizar esta conexión es el 80 correspondiente al servicio web (en este caso virtual, pero este es un detalle que el cliente no conoce), mientras que se seleccionó 1454 como puerto efímero del lado del cliente.

La trama 4 corresponde al pedido ARP que realiza el servidor virtual mediante su interfaz de red virtual eth0:0 -con dirección IP 192.168.0.104- con el objeto de obtener la dirección física de la interfaz de red correspondiente a la dirección 192.168.0.108 (PC 16) dentro de su misma red IP. Esta dirección corresponde al servidor real escogido de acuerdo al algoritmo de scheduling asignado al servicio virtual, en este caso round robin.

El redirector, a continuación agrega la dirección IP del servidor real junto con la identificación de la conexión dada por el par 170.210.122.101:1454 como un ítem dentro de la tabla hash de conexiones. Esto permite que los nuevos paquetes entrantes al redirector provenientes del cliente y que correspondan con esta conexión sean redirigidos hacia un único servidor real durante el tiempo que dure la misma con el objetivo de mantener la coherencia.

La trama 5 es la contestación al pedido ARP por parte de la PC 16.

La trama 6 corresponde al reenvío del segmento de pedido de apertura de conexión originado en el cliente y dirigido hacia el redirector. En este caso se observa como el redirector realizó el proceso de NAT y más precisamente *destination NAT*, al modificar la dirección de destino del datagrama IP

que pasó de ser 170.210.122.104 a ser 192.168.0.108.

En la trama 7, el servidor real confirma el pedido de apertura de conexión mediante un segmento TCP con los flags SYN y ACK activados. Desde el punto de vista de IP, se trata de un datagrama con origen en la dirección del servidor real y destino en la dirección del cliente. Sin embargo la dirección ethernet corresponde a la del redirector. Esto es así porque el servidor real está utilizando al redirector como *gateway* por defecto para enviar el datagrama al cliente.

En la trama 8, el redirector reenvía la respuesta del servidor real hacia el cliente, con la particularidad de que modifica la dirección IP de origen utilizando la suya. Este proceso es otra forma de NAT conocida como *masquerading*.

En la trama 9, el cliente termina el proceso de establecer la conexión TCP enviando al servidor virtual un segmento TCP con el flag ACK activado.

En la trama 10 el redirector reenvia el datagrama anterior hacia el servidor real, realizando nuevamente *destination NAT*. En este caso la dirección de destino no se elige mediante un mecanismo de scheduling, ya que se asignó anteriormente el servidor con dirección IP 192.168.0.108 como destino para esta conexión.

En la trama 11, el cliente envia al servidor virtual el pedido de http, el cual es reenviado por el redirector hacia el servidor real en la trama 12.

Las tramas 13 y 14 corresponden a la confirmación del pedido http que genera el servidor real y que es enviada hacia el cliente pasando por el redirector.

Las tramas 15 y 16 llevan los segmentos TCP con la carga http como respuesta del servidor real al pedido realizado por el cliente.

Las tramas 17 a la 24 corresponden al proceso de cierre de la conexión TCP. Este es iniciado por el servidor real realizándose como siempre a través del redirector y no se diferencia por lo demás de un proceso normal de cierre de conexión TCP.

2.2. Caso 2: OpenMOSIX

MOSIX (Multicomputer Operating System for Unix) es un software desarrollado en la Universidad Hebrea de Jerusalem destinado a la implementación de clusters basados en computadoras personales. Fue diseñado a los efectos de dotar al kernel del sistema operativo Unix con capacidades de cómputo en ambientes de clusters. Se basa en la implementación de un planificador de procesos distribuidos sobre una red basada en el juego de protocolos TCP/IP, cuyo objetivo es el balanceo de cargas de CPU de modo transparente a los usuarios y programadores de aplicación. [22]

El proyecto OpenMOSIX es un desprendimiento del proyecto MOSIX debido al desacuerdo de Barak (líder del proyecto) de utilizar la licencia GPL para la distribución del software, requisito indispensable para su implementación en Linux (dado que es necesario partir de un proyecto distribuido a través de GPL como es el kernel de Linux). Por esta razón, el resto de los integrantes del proyecto deciden continuar por su propia cuenta con el desarrollo del proyecto con una implementación libre dentro del kernel de Linux fundando OpenMOSIX. [23]

En lo que sigue, se utilizará el nombre MOSIX/OpenMOSIX, indistintamente, haciéndose las aclaraciones pertinentes cuando corresponda.

2.2.1. Arquitectura

En cuanto a su arquitectura, MOSIX entra en la categoría de sistema operativo distribuido, ya que la implementación se encuentra en el núcleo del sistema operativo. Bajo la dirección del autor original del proyecto, MOSIX fue implementado en diversas versiones de Unix comerciales.

MOSIX utiliza un modelo *home* [24]. En este se considera que cuando un proceso se crea, el nodo desde donde es lanzado pasa a ser su nodo *home*. La migración de los procesos es realizada por el propio núcleo del sistema operativo. Los datos de memoria e instrucciones relativos a la parte del proceso que se ejecuta en modo usuario se trasladan al nodo elegido donde continuarán su ejecución. La parte del proceso que se ejecuta en modo kernel se mantiene en el nodo *home*. De esta forma la visión del sistema que tiene el proceso es que se está ejecutando en su nodo *home*.

Un cluster MOSIX se implementa bajo una arquitectura compañero a compañero, dado que cada nodo puede ser maestro de sus procesos creados localmente y esclavo de los procesos creados en otros nodos. A partir de que el algoritmo de balanceo de procesos se ejecuta de forma descentralizada, cada nodo toma el rol de maestro para los procesos creados localmente y de esclavo para aquellos procesos que migraron desde otros nodos. Esto implica que nuevas estaciones de procesamiento pueden ser agregadas o removidas en un cluster, con efectos mínimos sobre los procesos en ejecución. En MOSIX se implementa una técnica de balanceo de cargas dinámico que analiza donde se pueden ejecutar nuevos procesos (crear y migrar) o donde se pueden migrar procesos que están en ejecución a los efectos de lograr una mayor perfomance de cómputo.

Para lograr la funcionalidad, OpenMOSIX utiliza los siguientes módulos:

- ***Migración de Procesos***: La CPU que crea un proceso le asigna un identificador UHN (*Unique Home Node*). En una migración al proceso se lo divide en dos partes, el contexto de usuario

(*user part*) migra a la CPU destino y el contexto del sistema (*system part*) queda en la CPU que originalmente creó el proceso.

- ***Sistema de Archivos MOSIX***: MFS (*MOSIX File System*) está disponible solamente en la versión OpenMOSIX. Establece un sistema de archivos distribuidos, replicados en cada nodo.

- ***Acceso Directo al Sistema de Archivos***: DFSA (*Direct File System Access*) está disponible solamente en la versión OpenMOSIX. Es un método alternativo de acceso directo a archivos locales y remotos.

2.2.2. Objetivos

Según sus creadores, los objetivos perseguidos en el diseño del sistema MOSIX fueron los siguientes:

- ***Migración dinámica de procesos****:* Cada nodo puede evaluar en cada momento si migra procesos que ha creado a otros nodos pares.

- ***Imagen única del sistema****:* MOSIX presenta a cada proceso una vista unificada del sistema de archivos, dispositivos y facilidades de red.

- ***Autonomía en cada nodo de procesamiento****:* Cada nodo de procesamiento es independiente de los demás y puede decidir acerca de su grado de participación en el cluster. Es posible implementar nodos de procesamiento tipo diskless.

- ***Configuración dinámica****:* Los nodos de procesamiento pueden ingresar y salir del cluster en cada momento. Los procesos que usen recursos o estén corriendo en uno nodo saliente no se verán afectados.

- ***Escalabilidad****:* Al no implementarse algoritmos centralizados de control del cluster es posible crecer de forma eficiente, pudiendo escalar a una gran cantidad de nodos.

El cluster no aporta una eficiencia significativa al ejecutar programas que hayan sido programados bajo la técnica de procesos ligeros (*threads*), a lo sumo, tales procesos, pueden ser migrados a una CPU con mayor potencia. Java y MOSIX no son la mejor opción, por lo antedicho, en este caso se recomiendan alternativas, como por ejemplo RMI o CORBA.

2.2.3. Modelos de clusters

MOSIX implementa varios modelos de configuración de clusters:

- ***Single-pool***: Cada estación de usuario puede migrar procesos a las de sus compañeros.

- ***Server-pool***: Las estaciones de usuario no son parte del cluster. Mediante una sesión telnet, los usuarios se conectan al cluster y ejecutan sus aplicaciones.

- ***Adaptative-pool**:* En base a configuraciones de usuario, las estaciones pueden pasar a formar parte del cluster a partir de un momento determinado. Ejemplo, en una organización cuando los empleados dejan su trabajo pueden dejar sus estaciones encendidas y en modo estación de procesamiento hasta el otro día, cuando retornen a su trabajo.

2.2.4. Métodos de balanceo de procesos

Existen dos métodos básicos de balanceo de procesos:

a) Por selección de CPU de creación (*job initiation only*) es un método de tipo no pre-emptive.

b) por migración, como es el caso de MOSIX (*PPM Pre-emptive Proccess Migration*) donde continuamente cada nodo analiza donde migrar procesos en base a información recibida acerca del estado de carga de los distintos equipos componentes del cluster. A la hora de decidir donde migrar un proceso utilizará una métrica de carga promedio que le dirá cual nodo de procesamiento está con más recursos libres.

A los efectos de lograr transparencia de migración de procesos las llamadas al sistema son redireccionadas al nodo creador del proceso a través de una capa de enlace propia. En la figura 6 se grafica este comportamiento.

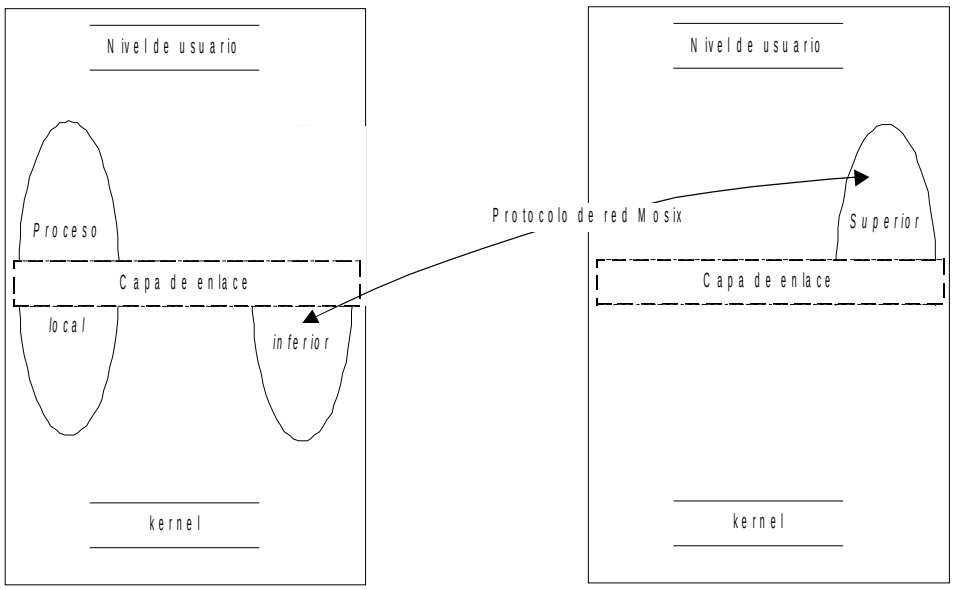

Figura 6. *Arquitectura de dos capas de OpenMOSIX*

La arquitectura del sistema divide al kernel en dos capas: inferior y superior, vinculadas por una capa de enlace. Cada objeto, tal como un archivo abierto, posee único identificador universal en el cluster. Por lo cual la capa superior del kernel referenciará a los objetos con tal identificador, y cuando migre un proceso los identificadores de los objetos asociados no cambiarán y serán parte de los datos a migrar. La capa de enlace tendrá asociados a cada identificador universal un identificador local de objeto provisto por la capa de kernel inferior y será la encargada de las comunicaciones entre ambas capas a través de llamadas a procedimientos remotos de kernel.

Un proceso que se ejecuta en un nodo no tiene conocimiento explícito acerca de sobre cual nodo de procesamiento está corriendo, ni sobre si seguirá ejecutándose sobre el mismo. El éxito de MOSIX reside en que la capa superior del kernel es independiente de la localización de los recursos físicos que el proceso, cuando la capa superior necesita un recurso la capa de enlace se encarga de vincular tal petición con el proveedor de servicio. Cuando un proceso migra se transfiere el área de usuario y las páginas sucias.

El algoritmo de migración es cooperativo, por eso el nodo destino seleccionado debe aceptar o no un pedido de migración. Tal característica permite que cada nodo de procesamiento pueda regular su participación en el cluster. Nodos individuales pueden forzar a que procesos de terceros nodos que ellos estén satisfaciendo con recursos propios migren. Tal situación es útil ante una orden de apagado de la plataforma.

En un cluster MOSIX, cada nodo a intervalos regulares envía información a parte de sus pares (elegidos al azar), donde detalla su estado de uso de ciertos recursos (carga actual, número de procesadores, velocidad de los procesadores, memoria utilizada y memoria libre). Los nodos a los cuales llegue el mensaje almacenarán la información recibida en un buffer para futuros usos y contestarán con información propia al nodo emisor. La idea de enviar la información a un subconjunto de nodos de procesamiento es prevenir que todos los nodos vean un único nodo con menos recursos ocupados, esta técnica previene la sobrecarga de trabajo sobre un único nodo.

Existe una técnica alternativa de migración, donde si un nodo detecta que posee poca memoria principal libre ejecutará un algoritmo denominado "memory ushering" que tratará de migrar el proceso local con mayor requerimiento de memoria al nodo compañero con menor cantidad de memoria utilizada.

2.2.5. Protocolos de red utilizados en MOSIX

El protocolo de transporte no orientado a la conexión UDP es utilizado a los efectos de transmitir información de estado. En cambio, se utiliza el protocolo de transporte TCP a los efectos de establecer conexiones entre nodos que desean migrar procesos. En MOSIX las comunicaciones no se encriptan ni autentifican, debido a tales flaquezas de seguridad si se utiliza una red WAN como soporte de cluster se aconseja que el sistema operativo se ejecute sobre una red privada virtual (VPN).

2.2.6. MFS. Sistema de Archivos MOSIX

Como se mencionó anteriormente, MOSIX utiliza su propio sistema de archivos (MFS), para hacer disponible a todos los usuarios aquellos directorios y archivos regulares que compartan. Una de las ventajas de MFS es que proporciona consistencia sobre una memoria caché donde están los archivos que necesitan ser vistos desde distintos nodos.

MFS cumple con los estándares DFSA (Direct File System Access), haciendo que se amplíe la capacidad de un proceso emigrado para realizar operaciones de I/O localmente, en el nodo donde se encuentre. Esta disposición reduce la necesidad de procesos I/O-bound de comunicarse con su nodo home, permitiendo así que tales procesos (así como procesos con uso mixto de I/O y de CPU) emigren más libremente entre los nodos del cluster logrando una mejor distribución de la carga

entre estos. A su vez, esto también permite paralelizar los accesos de I/O.

Como se mencionó anteriormente cada proceso posee un identificador UHN que se le asigna en la CPU donde se crea. Antes de realizar una migración, MOSIX divide al proceso a migrar en dos contextos, el contexto de usuario y el contexto de sistema. El contexto de usuario, que contiene el código del programa, la pila de usuario, datos, mapa de memoria y los registros asociados al proceso, el cual se moverá a un destino remoto para seguir procesándose. El contexto de sistema, que contiene otros recursos asociados al proceso queda en el nodo que le asignó el UHN.

Luego de una migración, las interacciones entre el contexto de usuario y el contexto del sistema serán interceptadas por una capa de enlace, la cual establece un canal de comunicaciones entre ambos contextos.

En el archivo /proc/<pid>/where puede visualizarse donde está corriendo un determinado proceso, siendo 0 en la CPU local. Si por ejemplo se tipea el siguiente programa:

```
awk{BEGIN(for()i=0;i<10000;i++)for(j=0;j<10000;j++);}
```

Luego puede averiguarse el número de proceso y verificar donde verdaderamente está ejecutándose.

Alternativamente, un cluster MOSIX puede programarse a los efectos de que a cierta hora del día las computadoras de usuario comiencen a integrarse al sistema. Es decir cuando los usuarios se retiran a sus hogares y sus estaciones quedan encendidas.

Los siguientes escenarios son propicios para que un cluster MOSIX pueda aportar ventajas de procesamiento distribuido:

- ***Procesos CPU-bound***, tales como tareas ingenieriles o científicas que requieran extensos recursos de cómputo.
- ***Procesos paralelos,*** especialmente aquellos que posean tiempo no predecible de ejecución.
- ***Procesos I/O-bound paralelos,*** mediante el uso de DFSA con MFS o GFS para acceder a los archivos localmente sin usar la LAN.
- ***Clusters con diferentes capacidades de procesamiento en los nodos***, tales como diferentes velocidades de CPU y capacidad de memoria RAM.
- ***Ambientes multiusuario de tiempo compartido.***
- ***Servidores web escalables.***

En cambio, los siguientes escenarios no son beneficiosos para implementar un cluster MOSIX:

- Aplicaciones con alto nivel de comunicación y bajo nivel de procesamiento, esto se solucionaría con la implementación de sockets migrables.

- Aplicaciones que usen memoria compartida, debido a que no hay soporte de DSM en Linux. MOSIX soportará DSM a través del proyecto "Network RAM" cuyo objetivo es que los procesos migren hacia los datos en lugar de los datos a los procesos.

- Aplicaciones dependientes del hardware, esto es que requieran acceder al hardware de un determinado nodo.

2.2.7. Interfase de control Mosixview

Es un software de aplicación que tiene por finalidad la administración de un cluster MOSIX. Sus componentes principales son:

- ***Mosixview*** es la aplicación principal y brinda la interfase de usuario.

- ***Mosixcollector*** es un demonio que recolecta información acerca del funcionamiento del cluster.

- ***Mosixload*** analiza la información recolectada por el módulo anterior, la procesa y la muestra en forma gráfica.

- ***Mosixmem*** es un módulo que muestra la evolución del uso de memoria.

- ***Mosixhistory*** es un módulo que permite consultar sobre la historia de ejecución de procesos.

La aplicación está diseñada a los efectos de reducir el trabajo administrativo invertido en la administración de un cluster Mosix. Permite gerenciar un número significativo de nodos, permitiendo cambiar configuraciones de nodos.

El usuario administrador, utilizando la interfase principal de estado del cluster, puede conocer la eficiencia del sistema, memoria utilizada, entre otros datos importantes. Manualmente, este puede migrar procesos a través de la mencionada interfase. Es posible acceder al estado global del cluster en un tiempo pasado t, como así también la evolución de las migraciones y eficiencia asociada a una tarea de cómputo, utilizando el módulo *Mosixhistory*.

2.2.8. Proyectos basados en MOSIX

2.2.8.1. Cluster LSTP-MOSIX

Es una variante al proyecto LSTP (*Linux Terminal Server Project*), que consiste en montar un cluster MOSIX sobre un servidor de recursos y un conjunto de nodos que operan en modo Xterminal sin unidades de almacenamiento permanente. Los autores plantean que lograron un cluster de fácil configuración que permite en poco tiempo (minutos) agregar nuevos nodos de procesamiento, y por ende es ideal para convertir estaciones de trabajo basadas en sistemas operativos Microsoft en nodos del cluster, cuando sus usuarios abandonan el horario de trabajo

normal.

2.2.8.2. Clumpos

Es una distribución diskless en CDROM, que permite de forma automática agregar nodos de procesamiento a un cluster MOSIX. Al arrancar una estación de procesamiento ClumpOS, obtiene de un servidor DHCP una dirección IP de red, configura un archivo mosix.map y queda a la espera que el servidor principal de migración comience a entregarle trabajos.

Los requerimientos de hardware de la plataforma donde opera son mínimos: CPU 80586 o superior, booteo desde unidad de CDROM, 64MB o más de memoria central, interfase de red Ethernet.

2.2.9. Experiencia de laboratorio: prueba de eficiencia de un cluster OpenMOSIX

El autor del presente trabajo participó en la configuración de los equipos en un trabajo de investigación donde se analizó el rendimiento de un cluster OpenMOSIX [37]. Se realizaron una serie de pruebas con el objetivo de medir el rendimiento del sistema en función de la variación de los siguientes parámetros:

a) la cantidad de nodos que colaboran.

b) costo computacional de la ejecución de un proceso hijo particular.

c) el tipo de red local utilizada.

2.2.9.1. Recursos utilizados

Se construyó un cluster OpenMOSIX utilizando la distribución ClusterKnoppix, que es una variante de Knoppix, y al igual que ésta no requiere instalación previa, simplemente inicia el sistema operativo desde la unidad de CDROM. En particular se uso la versión 3.2.

Hardware utilizado:

- 6 PCs con procesador AMD Duron de 1.1 Ghz, 128 MB de memoria RAM, sin disco rígido.
- 1 PC con procesador Intel Pentium de 100 Mhz y 64 MB de memoria RAM, disco rígido de 4 GB.
- 1 Switch de 100 Mbps.
- 1 HUB de 100 Mbps.

2.2.9.2. Pruebas

Los valores tomados en cada prueba son tiempos de ejecución y se expresan bajo la forma de índice base 1. Por ejemplo, un valor 4 de índice indicaría que el tiempo de proceso es de 1/4 con respecto a la base.

En una primera experiencia (prueba A) se evaluó el rendimiento del cluster en función de la cantidad de nodos que lo componen (desde 2 hasta 7) y del costo computacional de cada proceso hijo, expresados en iteraciones (desde 5 millones hasta 30). El modo de trabajo de la red local fue Ethernet conmutado. Para todas las configuraciones de la prueba se crearon 25 procesos hijos. El valor resultado de cada configuración surge de promediar 5 repeticiones. En la tabla 2 se pueden ver los valores obtenidos.

Tabla 2. Prueba A: eficiencia de procesamiento con distintas configuraciones de equipos/duración de procesos

Prueba A	30 millones	20 millones	10 millones	7.5 millones	5 millones
2 equipos	1,00	1,00	1,00	1,00	1,00
3 equipos	1,74	1,68	1,74	1,70	1,58
4 equipos	2,47	2,38	2,33	2,26	2,18
5 equipos	3,27	3,15	3,11	3,03	2,87
6 equipos	3,59	3,51	3,29	3,17	3,12
7 equipos	4,00	3,78	3,65	3,53	3,28

Nótese que la mayor eficiencia se logra con 7 nodos y con procesos hijos de 30 millones de iteraciones cada uno. Tales valores eran esperables dado que a la misma cantidad de nodos y menos iteraciones el costo de tiempo de migración de procesos toma más relevancia. En la figura 7 se puede ver más claramente como se incrementa la eficiencia al ir incorporando más nodos al cluster.

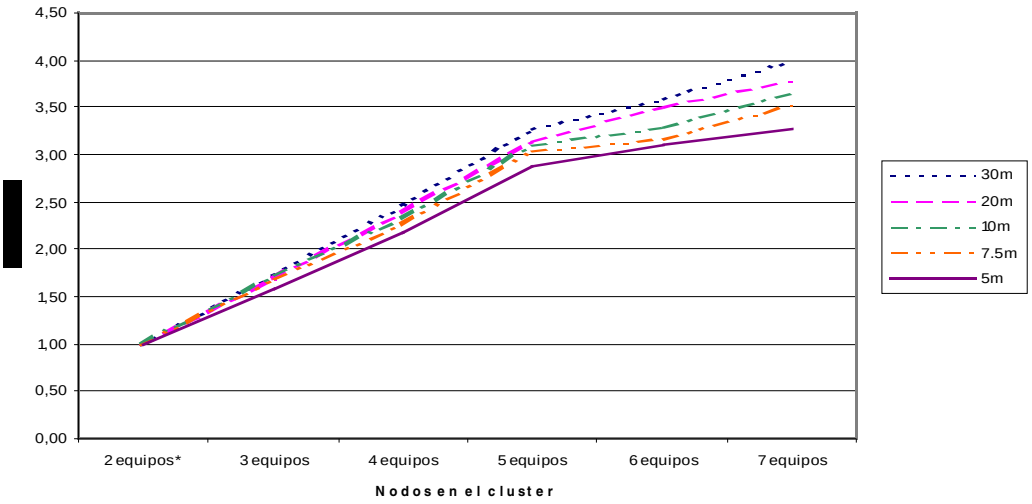

Figura 7. *Gráfico de los valores de la Prueba A.*

En una segunda experiencia (Prueba B), se evaluó el desempeño del cluster en función de la tecnología de red local utilizada y de la cantidad de bytes de parámetros pasados a cada proceso hijo. En un caso los nodos se vincularon entre sí a través de una red Ethernet con un repetidor multipuerto (Hub), en el otro caso los nodos se conectaron a un conmutador (Switch) Ethernet. Toda la prueba se realizó en base a un cluster de 6 nodos y 25 procesos hijos.

Tabla 3. Prueba B: eficiencia de procesamiento con distintas configuraciones de tamaños de parámetros/tecnología de red

Prueba B Tamaño de los parámetros pasados (en Kbytes)	Switch	Hub
0	1,00	1,00
5	1,00	0,99
50	0,98	0,94
500	0,83	0,53
1000	0,59	0,34

La prueba muestra la significativa pérdida de rendimiento del cluster cuando se lo opera bajo el modelo de comunicaciones de canal compartido.

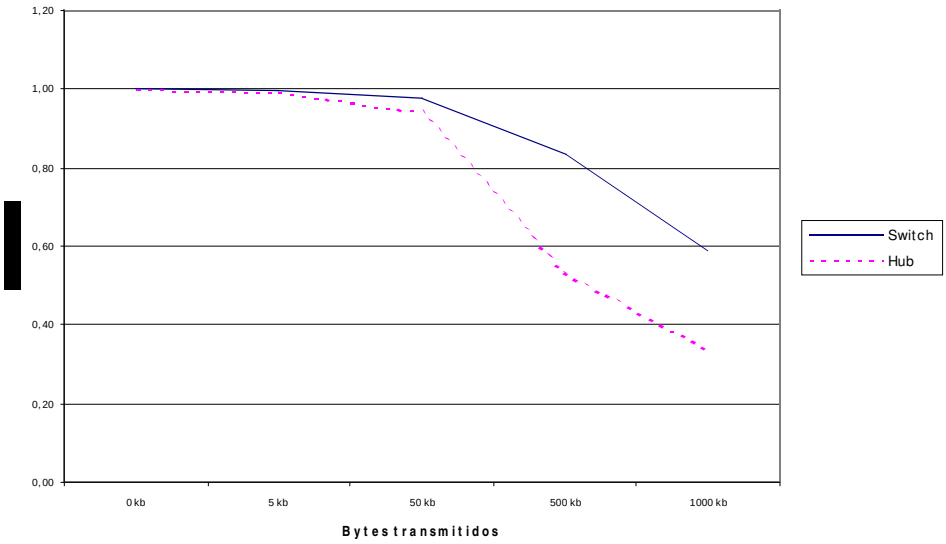

Figura 8. *Gráfico de los valores de la Prueba B.*

Por último en la prueba C se evaluó el desempeño en función de la cantidad de procesos hijos a distribuir entre los nodos del cluster. La prueba se realizó bajo una red Ethernet conmutada.

Tabla 4. Prueba C: eficiencia de procesamiento con distintas cantidades de procesos hijos/cantidad de equipos

Prueba C	25 hijos	26 hijos	27 hijos	28 hijos	29 hijos	30 hijos	31 hijos
2 equipos	1,00	1,00	1,00	1,00	1,00	1,00	1,00
3 equipos	1,74	1,77	1,72	1,73	1,71	1,76	1,77
4 equipos	2,33	2,34	2,31	2,32	2,34	2,43	2,24
5 equipos	3,11	3,15	3,08	2,98	3,10	3,03	3,03
6 equipos	3,29	3,44	3,44	3,52	3,56	3,61	3,63

7 equipos	3,65	3,77	3,75	3,84	3,92	3,90	4,05

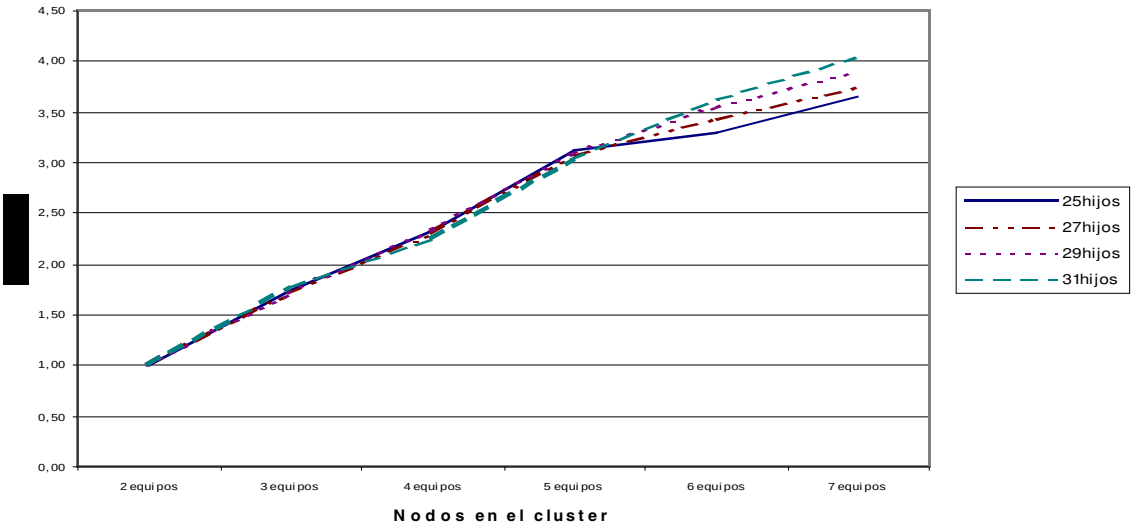

Figura 9. *Gráfico de los valores de la Prueba C.*

2.2.9.3. Análisis de los resultados obtenidos

En base a los datos resultantes de las pruebas realizadas se calcularon los índices de eficiencia que permiten evaluar la escalabilidad del cluster. Como era de esperar, OpenMOSIX tiene un mejor rendimiento operando sobre una red local Ethernet conmutada que sobre una red no conmutada, especialmente cuando los procesos hijos requieren de parámetros de operación.

Las pruebas muestran el aumento de la eficiencia conforme se anexan nodos al cluster y cómo se ve afectada por el costo computacional del algoritmo a ejecutar.

2.3. Caso 3: Beowulf

2.3.1. Objetivos

El proyecto Beowulf tiene su origen en el año 1994 como una iniciativa de Thomas Sterling y Donald Becker, construyendo un cluster de 16 PCs 486 DX4 comunicándose a través de una red Ethernet en la NASA.

Esta idea inicial desarrollada dentro de la NASA, pronto se difundió dentro del ámbito académico como así también entre los investigadores en computación paralela en general tomando el término Beowulf como una clasificación dentro de los clusters que hace referencia a la implementación de clusters económicos utilizando como nodos equipos disponibles en el mercado masivo al igual que las tecnologías de redes necesarias para interconectarlos. A su vez, el software que se ejecuta en estos es libre, no imponiendo costos extra por licencias. La solución de software más ampliamente utilizada consiste en usar alguna distribución del sistema operativo GNU/Linux. [28] [29]

Actualmente esta clase de clusters han tomado un lugar importante como soluciones de computación paralela no sólo compitiendo en cuanto al costo sino también llegando a ocupar puestos importantes en el listado de las 500 supercomputadoras más poderosas confeccionado por investigadores de la Universidad de Tennessee y el Laboratorio Nacional Lawrence Berkeley. Este listado es tomado como referencia en la comunidad de la computación paralela y que se realiza en base a datos de pruebas de rendimiento de supercomputadoras de todo el mundo. [34]

2.3.2. Tipos de clusters Beowulf

Aunque como ya se dijo antes los clusters Beowulf se caracterizan por la utilización de hardware disponible en el mercado masivo, dependiendo de la aplicación que se vaya a implementar puede que sea conveniente la incorporación de tecnologías que ofrezcan alto rendimiento para evitar cuellos de botella.

2.3.2.1. Beowulf Clase I

Se trata de utilizar sólo partes disponibles en el mercado masivo. Esto tiene la ventaja de que disminuye los costos notablemente como así también la disponibilidad del equipamiento. Otra ventaja añadida es que es más simple y económico de mantener al existir múltiples fuentes de soporte para estas tecnologías.

Concretamente, en la actualidad, las soluciones de hardware más económicas para la implementación de clusters Beowulf Clase I consisten en la utilización de ***Pcs*** conectadas mediante redes ***Ethernet***, ***Fast Ethernet*** o incluso ***Gigabit Ethernet***, siendo un rendimiento más alto el que se obtiene mediante Ethernet conmutada. Adicionalmente el rendimiento de las redes Ethernet puede incrementarse mediante la distribución de carga entre diferentes interfases de red (*channel bonding*), cuya implementación original fue realizada por Donald Becker utilizándose en la implementación del primer cluster Beowulf. Otras tecnologías a considerarse para interconectar los nodos son: ***cable***

paralelo (a través de los drivers del proyecto *CAPERS* pueden obtenerse muy bajos tiempos de retardo) y USB. [29]

Sin embargo, en aplicaciones que requieran mucho intercambio de datos entre los nodos, este tipo de cluster puede ofrecer rendimientos muy por debajo de los requeridos debido por ejemplo a altos retardos en el envío de los mensajes de red, requiriéndose en estos casos recurrir a tecnologías de enlace de red que ofrezcan bajos retardos en la transmisión de los datos.

2.3.2.2. Beowulf Clase II

Bajo esta clasificación se encuadra a los clusters construidos de tal forma que utilicen algún componente no disponible en el mercado masivo.

Un ejemplo de este tipo de cluster podría ser uno que utilice nodos multiprocesador, optimizando el uso de la red y logrando mayor rendimiento de cálculo.

En la implementación de clusters de esta clase un factor crítico son las tecnologías de red que se utilicen, esto da un impulso al desarrollo de nuevas tecnologías que ofrezcan altas prestaciones tanto sea en cuanto a las tasas de transferencia como a los tiempo de retardo.

Ejemplo de estas tecnologías son *10 Gibabit Ethernet, Myrinet, Parastation, ArcNet, ATM, FC (Fibre Channel), FireWire, Hippi, Serial Hippi, IrDA, SCI, SCSI, SHRIMP, ServerNet, TTL_PAPERS, WAPERS.* [29]

2.3.3. Librerías disponibles

Las librerías más populares para procesamiento paralelo son PVM [26] [30] y MPI [31]. Estas presentan una API uniforme entre distintas implementaciones en arquitecturas heterogéneas.

En su implementación, son el middleware que se encarga de realizar el pasaje de mensajes entre los nodos del cluster utilizando la tecnología de red subyacente. Además, es responsabilidad de estas la conversión de los tipos de datos necesaria para que distintas arquitecturas puedan interpretar correctamente los datos mientras se provee una interfaz de programación ubicua.

2.3.3.1. PVM (*Parallel Virtual Machine*)

La programación en PVM está orientada a tareas que son definidas por el programador cooperando para ejecutar un algoritmo. PVM define funciones para el manejo de estas tareas tales como iniciar o detener una tarea, como así también para la comunicación y sincronización de las mismas.

Esta librería provee un ambiente en el cual los programas acceden a una API uniforme independientemente de la arquitectura de hardware donde se esté ejecutando encargándose del envío de los mensajes, la conversión de los datos y la planificación de las tareas. A su vez brinda a estos la perspectiva de que se están ejecutando en una única máquina virtual paralela.

Históricamente, ha sido la primera librería en difundirse masivamente dentro del ámbito de la computación paralela, dada su sencillez y su completitud. Por otra parte cuenta con soporte de alta

disponibilidad lo que permite el desarrollo de clusters que se ejecuten por largos períodos de tiempo, no interrumpiéndose la actividad de cómputo aún al fallar algún nodo. Sin embargo el desarrollo de esta librería está bajo la dirección de sus autores, no contando con un estándar ni una especificación consensuada por la industria.

2.3.3.2. MPI (*Message Passing Interface*)

MPI es una especificación de sistema para pasaje de mensajes realizada con amplia participación de la comunidad de investigación en computación paralela como fabricantes, organismos gubernamentales y universidades.

Se encuentran disponibles diversas implementaciones libres de MPI, dos de las más populares son LAM y MPICH, alcanzando un alto grado de madurez. [32] [33]

Un objetivo de los desarrolladores ha sido la portabilidad, dando especificaciones neutrales en cuanto al hardware y permitiendo que se desarrollen implementaciones en diversas arquitecturas.

2.3.4. Experiencia de laboratorio: prueba de eficiencia de un cluster Beowulf

La experiencia que se realizó consistió en la configuración de un cluster utilizando el programa mpi-povray y las librerías MPI para evaluar su rendimiento con distintos parámetros de funcionamiento:

a) la cantidad de nodos que colaboran.

b) el tamaño del segmento de datos a procesar por cada nodo.

2.3.4.1. Recursos utilizados

La implementación del cluster se hizo utilizando la distribución Debian GNU/Linux 3.1, dado que esta se encontraba instalada en los equipos del aula de informática de la Universidad Nacional de Luján al momento de realizar las pruebas siendo además una distribución muy flexible en cuanto a la instalación de software por lo que no se presentaron inconvenientes para realizar la experiencia.

Hardware utilizado:

- 20 Pcs con procesador Celeron de 2.2 Ghz, 256 MB de memoria RAM, 40 GB de disco rígido.
- 1 Switch de 100 Mbps.

Software utilizado:

- *mpi-povray 3.1:* Esta es una versión extraoficial del programa POV-Ray [35], modificada para el procesamiento paralelo a través de la API de MPI. Dado que este paquete no viene incluido en la distribución Debian y además la versión de POV-Ray incluida en la distribución no incluye soporte para MPI (como sí lo tiene para PVM), se utilizó la versión mpi-povray. [36]
- *Librerías MPI*: se utilizó el paquete *lam4* provisto por la distribución Debian que contiene las librerías que son usadas por los programas en tiempo de ejecución. Este paquete sólo fue necesario en el nodo maestro.
- Entorno de ejecución LAM: se utilizó el paquete lam-runtime de Debian que incorpora las utilidades como compilador, lanzador de aplicaciones, demonio de pasaje de mensajes, etc. Este paquete fue necesario en todos los nodos del cluster ya que contiene entre otros componentes el demonio *lamd* que debe ejecutarse en cada uno de los nodos participantes del cluster.
- Cliente/servidor SSH: se utilizó el paquete *ssh* incluido en Debian que provee la implementación OpenSSH [27] tanto del cliente como del servidor del protocolo ssh, como así también funciona como reemplazo a los comandos *rlogin*, *rcp* y en particular *rsh* que es usado por el demonio *lamd* ejecutándose en el nodo maestro como mecanismo de comunicación con los nodos del cluster.

2.3.4.2. Aplicación

La experiencia consistió en una prueba de rendimiento utilizando una versión del software de generación de imágenes fotorrealísticas POV-Ray.

La generación de las imágenes se realiza mediante una técnica conocida como trazado de rayos. Esta se basa en la idea de que las imágenes que percibimos pueden pensarse como una matriz de puntos o píxeles que se forman cada uno por la acción directa de un rayo de luz (con un color asociado) sobre un elemento fotosensible en nuestras retinas. Estos rayos de luz provienen de distintas fuentes de luz, desde las cuales se esparcen partiendo de un punto en un espacio de tres dimensiones. En este espacio pueden encontrarse distintos objetos con variadas propiedades. Al interactuar con éstos, los rayos de luz pueden sufrir variaciones en su intensidad, color y dirección hasta llegar a la retina del observador.

Dado este modelo, la mayor parte de los rayos que se generan en las fuentes de luz son irrelevantes a los efectos de considerar la imagen producida en la retina del observador por lo que calcularlos redundaría en un costo computacional innecesario. Por esta razón, los programas de trazado de rayos realizan el cálculo en dirección inversa a la que se daría en la realidad, es decir desde la matriz de píxeles de la "retina" hacia los objetos y las fuentes de luz, es decir que sólo se tienen en cuenta aquellos rayos de luz que son relevantes a la imagen que se desea obtener.

A pesar de la optimización antes mencionada, el costo computacional que tiene la generación de imágenes mediante la técnica de trazado de rayos sigue siendo muy grande debido a que pueden definirse objetos de complejidad arbitraria, materiales con distintos grados de reflexión, refracción, etc, involucrando enormes cantidades de operaciones de punto flotante.

Como se puede notar, el trazado de rayos permite generar imágenes con tal grado de realismo que llegan en muchos casos a confundirse con fotografías reales. Dado que los objetos en general se definen mediante funciones matemáticas o figuras geométricas perfectas, la resolución de la imagen a procesar puede especificarse en valores arbitrariamente grandes sin observarse pérdida de calidad en el resultado.

El costo computacional para producir una imagen digital depende fundamentalmente de dos aspectos:

1. la complejidad propia de la escena.
2. la resolución de la imagen.

Las escenas en POV-Ray se especifican mediante archivos de texto utilizando un lenguaje descriptivo propio de esta herramienta. A su vez, en éstos pueden hacer referencia a otros archivos conteniendo definiciones adicionales, en el paquete de distribución oficial de POV-Ray se incluye una librería muy extensa de definiciones que pueden utilizarse para facilitar la construcción de escenas contando con elementos como por ejemplo texturas y figuras. Además, también se cuenta con una gran cantidad de ejemplos de escenas completas.

2.3.4.3. La experiencia

Para realizar la prueba de rendimiento se utilizó la escena contenida en el archivo skyvase.pov ya que ésta ha sido utilizada en otras pruebas utilizando las versiones modificadas de POV-Ray para cómputo paralelo. [29]

Dado que de los dos factores antes mencionados como determinantes en el costo computacional para la generación de una imagen digital y, siendo que el primero de ellos está determinado por la definición de la escena que se utilice y que ésta fue determinada, se considerará un parámetro fijo para la prueba. Por lo tanto un parámetro importante a seleccionar fue la resolución de la imagen final.

Para esto se tuvo en cuenta cómo hacer la división de los datos a procesar por los nodos. MPI-POVRAY incluye dos parámetros adicionales que pueden utilizarse en la línea de comandos:

+NHXX: especifica el alto en píxeles del segmento de la imagen a enviar a cada proceso.

+NWXX: especifica el ancho en píxeles del segmento de la imagen a enviar a cada proceso.

Se decidió elegir una resolución que permitiera una distribución equánime de los datos a todos los nodos en el caso más extremo, es decir con todos los nodos participando y con el mayor tamaño del segmento de datos –lo que redunda en que se tenga la menor cantidad de segmentos-.

Dado que el cluster constaba de 20 nodos y que el mayor tamaño del segmento de datos que se consideró fue de 256x256 píxeles, la resolución para realizar la prueba se fijó en 1280x1280. Ésta permite dividir en 20 partes iguales la imagen a procesar.

La tabla que se muestra a continuación muestra la cantidad de segmentos que se generaron según los tamaños de segmento utilizados en cada prueba.

Tabla 5. Relación entre los tamaños de segmentos y las cantidades resultantes.

Tamaño del segmento de datos	***Cantidad de segmentos***	***Resolución de la imagen***
16x16	5120	1280x1280
32x32	1280	1280x1280
64x64	320	1280x1280
128x128	80	1280x1280
256x256	20	1280x1280

Puede verse que la cantidad de segmentos crece exponencialmente a medida que disminuye el tamaño del segmento de datos.

De igual manera que en las pruebas de OpenMOSIX, la variable se expresó como índice base 1, correspondiente en este caso a la ejecución en un solo nodo.

En la tabla que sigue se muestran los resultados de las pruebas realizadas.

Tabla 6. Eficiencia de procesamiento con distintos tamaños de segmentos de la imagen.

nodos	***16x16***	***32x32***	***64x64***	***128x128***	***256x256***
1	1	1	1	1	1
2	1,91	2,05	2,02	2,12	2
3	2,74	2,95	3,06	3,27	2,97
4	3,5	3,83	3,96	4,7	3,77
5	4,2	4,79	5,1	5,14	4,67
6	5,04	5,23	5,94	6,35	5,16
7	5,48	6,39	6,69	7,2	6,13
8	6	7,19	7,64	8,31	6,53
9	6,3	7,67	8,23	8,31	7
10	6,63	8,85	8,92	9	7
11	7	8,85	9,73	10,8	7,54
12	7	10,45	9,73	10,8	8,17
13	7,41	9,58	10,7	10,8	8,91
14	7,41	10,45	11,89	12	9,8
15	7,41	10,45	11,89	12	8,91
16	6,63	10,45	11,89	13,5	8,17
17	7	10,45	11,89	13,5	8,91
18	7	10,45	11,89	13,5	8,91
19	7	10,45	13,38	13,5	9,8
20	6	10,45	11,89	12	8,17

A continuación se puede ver el gráfico de estos valores.

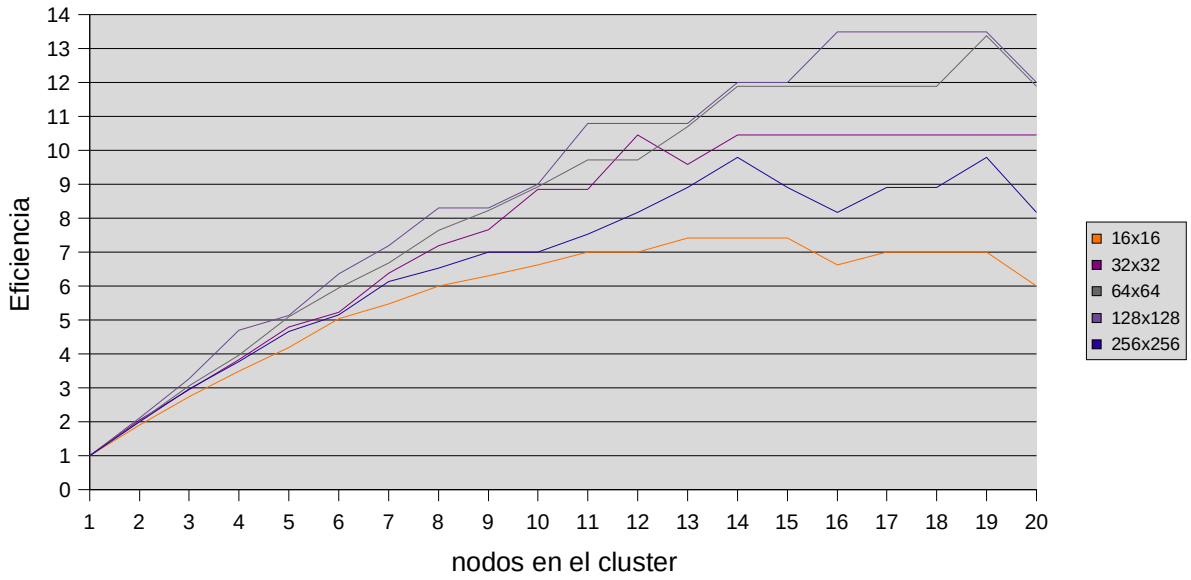

Figura 10. *Gráfico de los valores de la prueba de eficiencia*

Como puede verse, el peor rendimiento se obtuvo utilizando segmentos de 16x16, de la tabla 5 se desprende que los segmentos generados son 5120, estos generarán al menos una cantidad de

mensajes igual a esta cifra. Esto ocasiona que aumenten los tiempos muertos de procesamiento en los nodos debido a la espera de los datos para seguir procesando, dado que estos no llegan instantáneamente debido a la latencia de la red.

En el caso de los segmentos de 32x32 la causa de la pérdida de rendimiento puede explicarse también por el gran número de segmentos generados.

Los mejores rendimientos se obtienen con tamaños de segmento de 64x64 y 128x128, en este caso la cantidad de segmentos generados con respecto al tiempo necesario para procesarlos están mejor balanceados.

En el último caso, es decir los segmentos de 256x256, como era de esperarse, no se obtienen resultados óptimos. Esto puede explicarse por la espera excesiva que se genera hasta que el coordinador puede recolectar los datos resultantes, evento que ocurre secuencialmente cuando todos los nodos terminan su procesamiento y envían los datos a este.

2.3.4.4. Análisis de los resultados obtenidos

En base a los datos resultantes de las pruebas realizadas se calcularon los índices de eficiencia que permiten evaluar la escalabilidad del cluster.

Los resultados permiten determinar el tamaño óptimo de los segmentos de datos en que se dividió la tarea (en este caso 128x128), teniendo en cuenta la prueba en particular que se realizó.

Si se toma en cuenta este parámetro como referencia, pueden analizarse las limitaciones en cuanto a escalabilidad que se tiene utilizando la tecnología en este momento como son los equipos utilizados y una red Fast Ethernet. Se puede observar que se obtiene una muy buena eficiencia con hasta 11 equipos (10,8), sin embargo, a partir de este número el aumento de nodos participantes del cluster no conlleva un aumento lineal en el rendimiento.

3. TRABAJO FUTURO

3.1. *Linux Virtual Server*: persistencia de las conexiones establecidas ante la caída del redirector primario sin pasaje de mensajes entre los redirectores

3.1.1. Objetivos

El esquema de tolerancia a fallas que se plantea en Linux Virtual Server utiliza chequeos de sanidad que son enviados periódicamente por él o los redirectores de respaldo hacia el redirector primario para poder detectar la caída éste. En este caso, un redirector de respaldo puede asumir el rol de redirector primario iniciando el proceso de recuperación de fallas (*failover*). Este consiste en la apropiación de la dirección IP del redirector principal mediante *arp spoofing* y la activación de las reglas de LVS y Netfilter en una configuración idéntica a la que se utilizaba en el redirector principal.

De esta manera, el nuevo redirector puede atender las nuevas conexiones que se establezcan contra el servicio virtual. Sin embargo, dado que el redirector en modo de funcionamiento de respaldo no tuvo conocimiento de las conexiones que se iban estableciendo, al recibir paquetes dirigidos a una conexión existente los descartará.

Si bien LVS plantea una solución a este problema utilizando pasaje de mensajes entre los redirectores con información de las nuevas conexiones establecidas, se pensó en una solución alternativa que no requiere comunicación entre los redirectores ni ninguna modificación al software de LVS.

3.1.2. Modo de funcionamiento

Se propone un modo de funcionamiento de Linux Virtual Server donde los redirectores son configurados con las mismas direcciones MAC, IP y configuraciones idénticas de las reglas en los módulos de LVS.

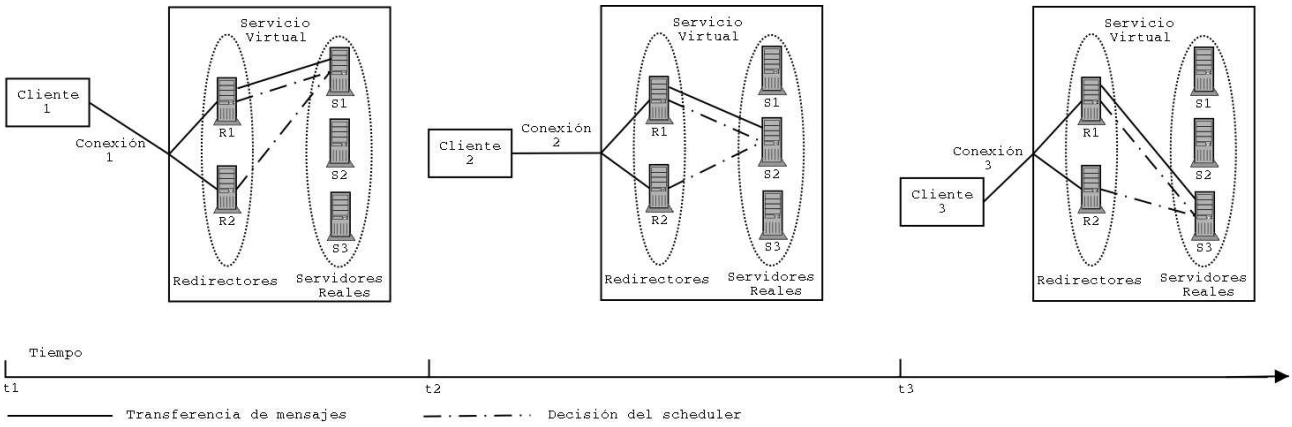

Figura 11. *Secuencia de asignaciones de conexiones en un esquema con dos redirectores utilizando un algoritmo de planificación round robin.*

A modo de ejemplo, en la figura 11 se muestra un caso con dos redirectores (R1 el principal y R2 el de respaldo) y tres servidores reales (S1, S2 y S3). Los pedidos de apertura de conexión de los clientes llegan a los redirectores. El módulo de LVS activo en cada uno de éstos, será el encargado de crear la entrada en la tabla de conexiones para que los paquetes subsiguientes sean redirigidos al mismo servidor real (la decisión de los planificadores en cada uno de éstos se muestra con línea punteada). Nótese sin embargo que sólo el redirector principal es quien reenvía los mensajes a los servidores reales. Esto se logra mediante la configuración de reglas de filtrado (mediante *Netfilter*) que bloquean los paquetes salientes en la interfaz que se comunica con los servidores reales. Cuando un redirector de respaldo asume el rol de principal, estas reglas se eliminan.

Supongamos que la configuración y el estado interno es idéntico en ambos redirectores. La secuencia de eventos ocurre de la siguiente manera:

t1: El primer cliente hace un pedido de conexión al servicio virtual. Este llega a R1 y R2. Los dos redirectores dan inicio al algoritmo de round robin asignando para esta conexión al primer nodo de sus listas resultando seleccionado S1. Luego el pedido de conexión es reenviado así como todos los paquetes subsiguientes por R1 mientras que R2 sólo mantiene la entrada en su tabla de conexiones sin reenviar ningún paquete.

t2: Un segundo cliente realiza un pedido de conexión. El pedido es recibido por R1 y R2. Ambos redirectores tienen entre las variables que componen su estado interno una referencia al nodo asignado para la última conexión que en este caso es S1. Los redirectores calculan el sucesor de S1 dentro de la lista de nodos obteniendo una referencia a S2. Luego ambos asignan a S2 como servidor real para esta conexión. Igual que en el caso anterior R1 es el encargado de reenviar los paquetes.

t3: Un tercer cliente realiza un pedido de conexión. Ambos redirectores reciben el paquete. Análogamente al caso anterior, los dos redirectores asignan a S3 para esta conexión ya que este nodo es el sucesor de S2 en las listas que estos mantienen.

Vemos en este ejemplo que los dos redirectores han asignado los mismos servidores reales para las conexiones entrantes teniendo sus tablas de conexiones idénticas. Esto permite que en caso de tomar R2 el rol de redirector principal se conserven las conexiones existentes.

3.1.3. El problema de la planificación

Considerando una topología igual que en la figura anterior y el mismo algoritmo de planificación, en la figura 12 se muestra el caso en que por un error en la red, uno de los redirectores pierde una petición de apertura de conexión.

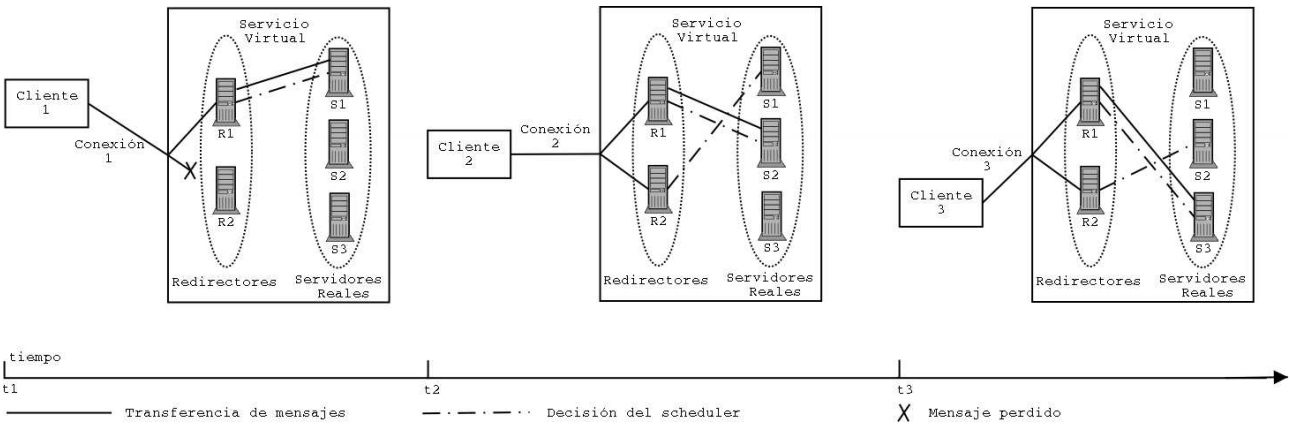

***Figura 12**. Ejemplo en el cual uno de los redirectores pierde un pedido de apertura de conexión.*

La secuencia de eventos en este caso se desarrolla de la siguiente manera:

t1: Un cliente realiza un pedido de conexión al servicio virtual. Este pedido es recibido por R1 pero no por R2 debido a un problema en la comunicacion. R1 inicia el algoritmo de round robin asignando a S1 como servidor real para esta conexión.

t2: Un segundo cliente realiza un pedido de conexión. El último nodo asignado por R1 es S1, por lo que asigna a S2 para esta conexión ya que es el sucesor de S1 en su lista de nodos. R2 también recibe el pedido de conexión, pero en su estado interno no tiene registro de ninguna conexión anterior por lo que asigna para esta conexión al primer nodo de la lista que es S1.

t3: Un tercer cliente realiza un pedido de conexión. El pedido es recibido por los dos redirectores. R1 tiene como último nodo asignado a S2, por lo que asigna a S3 para esta conexión. Sin embargo R2 asignó para la anterior conexión a S1 por lo que para esta conexión asigna a S2.

En este ejemplo se puede ver cómo la pérdida de un mensaje de petición de apertura de conexión puede ocasionar la pérdida de sincronización de los estados internos de los redirectores.

La primera consecuencia observable es que el redirector de respaldo pierde la información relativa a la conexión del primer cliente. En caso de asumir su rol de redirector principal R2 no tendrá la correspondiente entrada para la conexión 1 por lo cual rechazará todos los paquetes que se envíen para esta conexión provocando que ésta se interrumpa.

Sin embargo, lo que ocurre luego con las conexiones 2 y 3 muestra que la información incorrecta en R2 además provoca que todas las conexiones subsiguientes sean asignadas a nodos diferentes de los que escogió R1. En caso de asumir R2 el rol de redirector principal, reenviará los paquetes relacionados con estas conexiones a destinos incorrectos. Es decir, los paquetes que envíen los cientes serán enviados a servidores incorrectos y viceversa. Esto ocasiona que los destinos finales participantes en las conexiones afectadas rechacen los paquetes por no pertenecer a sockets que hayan negociado oportunamente.

En conclusión, la pérdida de un solo pedido de conexión puede provocar la inconsistencia de las tablas de conexiones entre los redirectores causando la pérdida de todas las conexiones abiertas contra un servicio virtual.

Esto se debe a que el algoritmo de round robin -como la mayoría de los algoritmos de

planificación que se utilizan normalmente- basan la elección del servidor real en datos que dependen de su estado interno. En caso de que este se desincronice entre los redirectores, las decisiones de planificación pueden variar ocasionando que escojan distintos servidores reales para las mismas conexiones.

3.2. Algoritmos de planificación dependientes de la petición

Para lograr que las conexiones sean asignadas a los mismos servidores reales en los redirectores participantes del servicio virtual es necesario que el algoritmo de planificación que ejecuten esté determinado por la petición del cliente. Esto determina que el nodo seleccionado en la decisión de planificación sea el mismo en todos los redirectores. En caso de que la petición de apertura de conexión no llegue a alguno de los redirectores, esto no afecta las decisiones subsiguientes que coincidirán en todos los redirectores.

Ejemplos de este tipo de algoritmos de planificación son ***source hash*** (que toma la decisión de planificación teniendo en cuenta la dirección IP de origen de la petición) y ***destination hash*** (la decisión en este caso depende de la dirección IP del destino de la petición).

3.3. Desincronización de las listas de nodos

Se ha visto como un algoritmo de planificación que utiliza los datos contenidos en la petición del cliente asegura que el redirector seleccione siempre el mismo nodo dentro de una lista para una conexión determinada.

Sin embargo si las listas de nodos difieren en los redirectores, las conexiones serán asignadas a diferentes nodos.

La lista de nodos es parte de la configuración inicial de los redirectores por lo que no debiera variar entre los redirectores. Sin embargo, debido a la utilización de mecanismos de tolerancia a fallas sobre los servidores reales, la lista puede modificarse dinámicamente.

El ejemplo de la figura 3 muestra cómo se puede producir la desincronización de las listas de nodos en los redirectores con la pérdida de un mensaje de *healthcheck* en uno de estos.

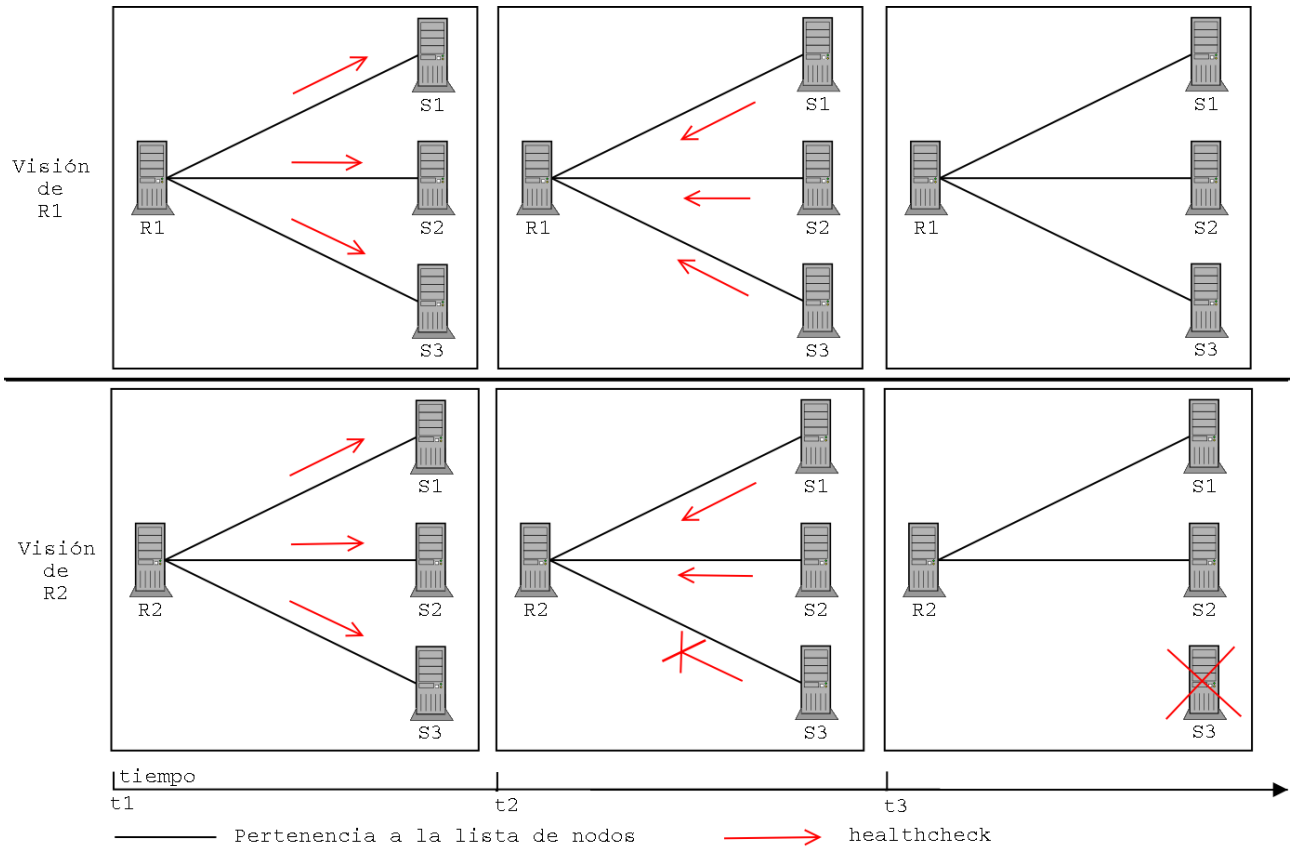

***Figura 13**. Modificación dinámica de las listas de nodos en los redirectores debida al intercambio de mensajes de healthcheck entre redirectores y servidores reales.*

La secuencia transcurre de la siguiente manera:

t1: R1 y R2 envían una consulta de *healthcheck* a los servidores reales que componen el cluster[3]. La consulta de *healthcheck* realizada por R1 llega correctamente a todos los servidores reales. Lo mismo ocurre con la consulta que envía R2.

t2: Los tres servidores reales responden al mensaje de R1. R2 recibe las respuestas de los *healthchecks* por parte de S1 y S2 pero, por un error en la comunicación, la respuesta de S3 no llega.

t3: R1 mantiene su lista de nodos sin modificaciones dado que recibe respuestas positivas a todas las consultas de *healthcheck* que envió. R2 elimina a S3 de su lista de nodos dado que no recibió confirmación positiva por parte de este nodo.

En el ejemplo anterior se observa cómo partiendo de una situación inicial donde las listas de nodos en los redirectores son idénticas se llega a una situación donde las listas difieren en un nodo debido al mecanismo de tolerancia a fallas.

3 Este conjunto de servidores puede diferir con respecto a la lista de nodos que mantienen los redirectores debido a que algunos servidores pueden estar momentáneamente caídos y deben ser chequeados periódicamente para poder ser activados cuando vuelvan a funcionar correctamente.

3.3.1. Planificación con diferencias en las listas de nodos

Considérese el siguiente servicio virtual:

Redirector	***Lista de nodos***	***cantidad de nodos***
R1	S1	3
	S2	
	S3	
R2	S1	2
	S2	

En esta configuración se tienen dos redirectores. R1 tiene tres nodos en su lista: S1, S2 y S3. R2 difiere en su lista de servidores reales, teniendo en su lista a S1 y S2.

Supóngase ahora que el algoritmo de planificación ejecuta el siguiente cálculo para tomar decidir que nodo asignar a la petición:

$$i = (p \bmod n) + 1$$

donde:

i: nodo a asignar para la petición p

p: un entero que representa la petición

n: cantidad de nodos

Supongamos que a este servicio virtual llega un pedido de conexión que puede ser representado como $p = 143$.

Al realizar el cálculo para la decisión de planificación, los redirectores obtendrán los siguientes resultados:

Redirector	***Cálculo de i***	***Nodo seleccionado***
R1	$i = (143 \textbf{ mod } 3) + 1 = 3$	S3
R2	$i = (143 \textbf{ mod } 2) + 1 = 2$	S2

Como puede observarse, los redirectores discrepan en su resultado para esta petición. El parámetro que provoca esta diferencia es la cantidad de nodos (n).

3.3.2. Listas de nodos sincronizadas entre los redirectores

Debido a que la desincronización de las listas entre los redirectores puede provocar discrepancias en las tablas de conexiones es necesario asegurar que las éstas sean idénticas en todos los redirectores o bien que el tiempo de desincronización sea mínimo (dado que sólo se ven afectadas aquellas conexiones que se abran durante la desincronización).

Se propone modificar el mecanismo de tolerancia a fallas en cuanto al chequeo de los servidores de la siguiente manera:

- Solo realiza los chequeos sobre los servidores reales el redirector principal.
- La lista que construye el maestro es transferida al/los redirectores de respaldo.

En caso de que algún redirector no recibiera el mensaje conteniendo la lista de nodos, su lista podría mantenerse desincronizada con respecto al maestro. Este caso se da cuando la lista de nodos difiere de la que tenía anteriormente el redirector en cuestión debido a la modificación por el cambio de estado de sanidad de algún servidor real. Durante el periodo de desincronización del redirector de respaldo podrían generarse diferencias en las tablas de conexiones debidas a las conexiones que se abran en este periodo. El periodo de desincronización dependerá del periodo entre las sincronizaciones de listas de nodos entre los redirectores.

3.3.3. Conclusiones

En este capítulo se analizó una solución posible al problema de la pérdida de conexiones establecidas en el caso de una caída del redirector principal en Linux Virtual Server.

Se analizaron los mecanismos existentes en esta tecnología y se plantearon algunas modificaciones que permitirían reducir drásticamente los tiempos de caída del servicio virtual en el caso antes mencionado.

4. CONSIDERACIONES FINALES

El presente trabajo me ha permitido interiorizarme en profundidad en los conceptos fundamentales de diseño de clusters como así también en las características técnicas de las implementaciones disponibles.

Por otro lado, mediante el estudio de la documentación y las experiencias de laboratorio, pude evaluar las características de alternativas tecnológicas concretas disponibles en forma de software libre. Éstas podrían brindar soluciones de bajo costo para las necesidades de alto rendimiento computacional que se requieran para proyectos de investigación en el ámbito de la Universidad Nacional de Luján.

Teniendo en cuenta que en la actualidad en la Universidad se dispone de un aula de informática con 25 equipos, se puede considerar la implementación de esta tecnología en horarios donde los recursos se encuentran ociosos. En este sentido, resulta interesante definir un proyecto para hacer uso de los mismos de forma remota por diferentes equipos de investigación que lo requieran.

5. REFERENCIAS

1. Tanenbaum, Andrew S. 1996. ***Sistemas Operativos Distribuidos***, primera edición. Prentice Hall. ISBN 968-880-627-7.
2. Silberschatz, A.; Galvin P. B. 1997. ***Operating System Concepts***, quinta edición. John Wiley & Sons, Inc. ISBN 0-471-41714-2
3. Stallings, William. 2001. ***Sistemas Operativos***, cuarta edición. Prentice Hall. ISBN 84-205-3177-4
4. Colouris, George; Dollimore, Jean; Kindberg, Tim. ***Sistemas Distribuidos - Conceptos y Diseño***, tercera edición. Addison Wesley. ISBN 84-7829-049-4.
5. Walker, Bruce J. ***OpenSSI: Open Single System Image Linux Cluster Project***. Disponible en: http://openssi.org/ssi-intro.pdf
6. Proyecto ClimatePrediction.net: http://www.climateprediction.net
7. Dubois, Paul F. 2001. ***SETI@home: Massively Distributed Computing for SETI***. Computer Society. Disponible en: http://www.computer.org/cise/articles/seti.htm
8. Aberdeen, Douglas; Baxter, Jonathan; Edwards, Robert. ***98¢ /MFlop, Ultra-Large-Scale Neural-Network Training on a PIII Cluster***. Disponible en: http://tux.anu.edu.au/Projects/Beowulf/sc2000.pdf
9. Swinburne Centre for Astrophysics. ***Reaching for the Stars***. Disponible en: http://www.dell.com/downloads/global/power/di4q03-15.pdf
10. Wensong Zhang. 2000. ***Linux Virtual Server for Scalable Network Services.*** Ottawa Linux Symposium 2000. Disponible en: http://www.linuxvirtualserver.org/ols/lvs.pdf
11. ***Virtual Server Scheduling Algorithms***, Disponible en: http://www.linuxvirtualserver.org/docs/scheduling.html
12. ***Proyecto UltraMonkey***: http://ultramonkey.sourceforge.net
13. ***Proyecto de Alta Disponibilidad en Linux***: http://www.linux-ha.org
14. Mack, Joseph. ***LVS mini-HOWTO.*** 2001. Disponible en http://www.linuxvirtualserver.org/Joseph.Mack/mini-HOWTO/LVS-mini-HOWTO.html
15. Mack, Joseph. ***LVS HOWTO.*** 2001. Disponible en http://www.linuxvirtualserver.org/Joseph.Mack/HOWTO/LVS-HOWTO-1.html
16. Andreasson, Oskar. ***Iptables Tutorial 1.1.19.*** 2003. Disponible en http://www.linuxsecurity.com/resource_files/firewalls/IPTables-Tutorial/iptables-tutorial.html
17. Russell, Rusty. ***Linux 2.4 NAT Como.*** Disponible en http://netfilter.samba.org/documentation/HOWTO/es/NAT-HOWTO.html
18. Whalen, Sean. ***An Introduction to ARP Spoofing.*** 2001. Disponible en http://chocobospore.org/arpspoof
19. ***Proyecto Global File System***: http://ww.globalfilesystem.org
20. ***Proyecto Coda***: http://www.coda.cs.cmu.edu

21. ***Proyecto Intermezzo***: http://intermezzo.org

22. Barak, Ammon; La'adan, Oren. 1998. ***The MOSIX Multicomputer Operating System for High Performance Cluster Computing***. Disponible en: http://www.mosix.org/pub/mosixhpcc.pdf

23. ***Proyecto OpenMOSIX***: http://openmosix.sourceforge.net

24. Lavi, Aharon Ron. 1999. ***The Home Model for Load Balancing in a Computing Cluster***. Institute of Computer Science, The Hebrew University of Jerusalem.

25. The LAM/MPI Team. 2004. ***LAM/MPI User's Guide Version 7.1.1***. Open Systems Lab. University of Indiana.

26. Geist, Al; Beguelin, Adam; Dongarra, Jack; Jiang, Weicheng; Manchek, Robert; Sunderam, Vaidy. 1994. ***A Users' Guide and Tutorial for Networked Parallel Computing.*** Massachusetts Institute of Technology. Disponible en: http://www.netlib.org/pvm3/book/pvm-book.html

27. ***OpenSSH***: http://www.openssh.com

28. Proyecto Beowulf: http://www.beowulf.org

29. Morrison, Richard S. 2003. ***Cluster Computing - Architectures, Operating Systems, Parallel Processing & Programming Languages***. Distribuido bajo licencia GPL.

30. Proyecto PVM: http://www.csm.ornl.gov/pvm/pvm_home.html

31. MPI Forum: http://www.mpi-forum.org

32. Proyecto LAM: http://www.lam-mpi.org

33. Proyecto MPICH: http://www.mcs.anl.gov/mpi/mpich/

34. ***Benchmark performance and ranking the of the top 500 Supercomputers***: http://www.top500.org. Universidad de Mannheim, Universidad de Tennessee, Laboratorio Nacional Lawrence Berkeley.

35. Proyecto POV-Ray: http://www.povray.org

36. Proyecto MPI-POVRAY: http://www.verrall.demon.co.uk/mpipov

37. Bordignon, Fernando R. A.; Tolosa, Gabriel. H; Lorge, Fernando. Poster: ***"EVALUACIÓN DE LA EFICIENCIA DE UN CLUSTER OPENMOSIX"***. Jornadas de la Ciencia y Tecnología. Universidad Nacional de Luján. Mayo 2004. Disponible en http://www.tyr.unlu.edu.ar/TYR-publica/poster-cluster-2004.doc

38. Fragati, Hernán M.; Director: Bordignon, Fernando R. A. Trabajo final para la obtención del título de Licenciado en Sistemas de Información: ***"PROCESAMIENTO MASIVO DISTRIBUIDO EN INTERNET"***. Universidad Nacional de Luján. Febrero 2002.

Anexo I: Capturas de red de la experiencia de LVS

Trama 1:

Ethernet (1013902103.199682)

Hardware source: 00:80:ad:c8:19:91
Hardware destination: ff:ff:ff:ff:ff:ff
Type /Length: 0x806 (ARP)
Media length: 60

ARP Header

Hardware type: 0x1 (Ethernet)
Protocol type: 0x800 (IP)
Hardware length: 6
Protocol length: 4
Opcode: 1 (request)
Sender Ether address: 00:80:ad:c8:19:91
Sender IP address: 170.210.122.101
Target Ether address: 00:00:00:00:00:00
Target IP address: 170.210.122.104
Padding: 0x20202020202020202020202020202020202000

Trama 2:

Ethernet (1013902103.199967)

Hardware source: 00:00:21:6c:5a:7f
Hardware destination: 00:80:ad:c8:19:91
Type /Length: 0x806 (ARP)
Media length: 60

ARP Header

Hardware type: 0x1 (Ethernet)
Protocol type: 0x800 (IP)
Hardware length: 6
Protocol length: 4
Opcode: 2 (reply)
Sender Ether address: 00:00:21:6c:5a:7f
Sender IP address: 170.210.122.104
Target Ether address: 00:80:ad:c8:19:91
Target IP address: 170.210.122.101
Padding: 0xc57601000001000000000000047063313200

Trama 3:

Ethernet (1013902103.200136)

Hardware source: 00:80:ad:c8:19:91
Hardware destination: 00:00:21:6c:5a:7f
Type /Length: 0x800 (IP)
Media length: 60

IP Header

Version: 4
Header length: 5 (20 bytes)
TOS: 0x00
Total length: 44
Identification: 11285
Fragmentation offset: 0
Unused bit: 0
Don't fragment bit: 1
More fragments bit: 0
Time to live: 128
Protocol: 6 (TCP)
Header checksum: 33860
Source address: 170.210.122.101
Destination address: 170.210.122.104

TCP Header

Source port: 1454 (unknown)
Destination port: 80 (http)
Sequence number: 388762
Acknowledgement number: 0
Header length: 6
Unused: 0
Flags: S
Window size: 8192
Checksum: 14614
Urgent: 0
Option: 2 (maximum segment size)
Length: 4
MSS: 1460

HTTP

Trama 4:
Ethernet (1013902103.200635)

Hardware source: 00:00:21:6c:5a:7f
Hardware destination: ff:ff:ff:ff:ff:ff

Type /Length: 0x806 (ARP)
Media length: 60

ARP Header

Hardware type: 0x1 (Ethernet)
Protocol type: 0x800 (IP)
Hardware length: 6
Protocol length: 4
Opcode: 1 (request)
Sender Ether address: 00:00:21:6c:5a:7f
Sender IP address: 192.168.0.104
Target Ether address: 00:00:00:00:00:00
Target IP address: 192.168.0.108
Padding: 0xc576010000010000000000000047063313200

Trama 5:
Ethernet (1013902103.201081)

Hardware source: 00:80:ad:c8:19:74
Hardware destination: 00:00:21:6c:5a:7f
Type /Length: 0x806 (ARP)
Media length: 42

ARP Header

Hardware type: 0x1 (Ethernet)
Protocol type: 0x800 (IP)
Hardware length: 6
Protocol length: 4
Opcode: 2 (reply)
Sender Ether address: 00:80:ad:c8:19:74
Sender IP address: 192.168.0.108
Target Ether address: 00:00:21:6c:5a:7f
Target IP address: 192.168.0.104

Trama 6:
Ethernet (1013902103.201435)

Hardware source: 00:00:21:6c:5a:7f
Hardware destination: 00:80:ad:c8:19:74
Type /Length: 0x800 (IP)
Media length: 60

IP Header

Version: 4

Header length: 5 (20 bytes)
TOS: 0x00
Total length: 44
Identification: 11285
Fragmentation offset: 0
Unused bit: 0
Don't fragment bit: 1
More fragments bit: 0
Time to live: 128
Protocol: 6 (TCP)
Header checksum: 59498
Source address: 170.210.122.101
Destination address: 192.168.0.108

TCP Header

Source port: 1454 (unknown)
Destination port: 80 (http)
Sequence number: 388762
Acknowledgement number: 0
Header length: 6
Unused: 0
Flags: S
Window size: 8192
Checksum: 40252
Urgent: 0
Option: 2 (maximum segment size)
Length: 4
MSS: 1460

HTTP

Trama 7:

Ethernet (1013902103.205322)

Hardware source: 00:80:ad:c8:19:74
Hardware destination: 00:00:21:6c:5a:7f
Type /Length: 0x800 (IP)
Media length: 58

IP Header

Version: 4
Header length: 5 (20 bytes)
TOS: 0x00
Total length: 44

Identification: 0
Fragmentation offset: 0
Unused bit: 0
Don't fragment bit: 1
More fragments bit: 0
Time to live: 64
Protocol: 6 (TCP)
Header checksum: 21632
Source address: 192.168.0.108
Destination address: 170.210.122.101

TCP Header

Source port: 80 (http)
Destination port: 1454 (unknown)
Sequence number: 1412927351
Acknowledgement number: 388763
Header length: 6
Unused: 0
Flags: SA
Window size: 5840
Checksum: 49836
Urgent: 0
Option: 2 (maximum segment size)
Length: 4
MSS: 1460

Trama 8:

Ethernet (1013902103.208711)

Hardware source: 00:00:21:6c:5a:7f
Hardware destination: 00:80:ad:c8:19:91
Type /Length: 0x800 (IP)
Media length: 60

IP Header

Version: 4
Header length: 5 (20 bytes)
TOS: 0x00
Total length: 44
Identification: 0
Fragmentation offset: 0
Unused bit: 0
Don't fragment bit: 1
More fragments bit: 0
Time to live: 63

Protocol: 6 (TCP)
Header checksum: 61785
Source address: 170.210.122.104
Destination address: 170.210.122.101

TCP Header

Source port: 80 (http)
Destination port: 1454 (unknown)
Sequence number: 1412927351
Acknowledgement number: 388763
Header length: 6
Unused: 0
Flags: SA
Window size: 5840
Checksum: 24198
Urgent: 0
Option: 2 (maximum segment size)
Length: 4
MSS: 1460

HTTP

Trama 9:

Ethernet (1013902103.209159)

Hardware source: 00:80:ad:c8:19:91
Hardware destination: 00:00:21:6c:5a:7f
Type /Length: 0x800 (IP)
Media length: 60

IP Header

Version: 4
Header length: 5 (20 bytes)
TOS: 0x00
Total length: 40
Identification: 11541
Fragmentation offset: 0
Unused bit: 0
Don't fragment bit: 1
More fragments bit: 0
Time to live: 128
Protocol: 6 (TCP)
Header checksum: 33608

Source address: 170.210.122.101
Destination address: 170.210.122.104

TCP Header

Source port: 1454 (unknown)
Destination port: 80 (http)
Sequence number: 388763
Acknowledgement number: 1412927352
Header length: 5
Unused: 0
Flags: A
Window size: 8760
Checksum: 27355
Urgent: 0

HTTP

Trama 10:
Ethernet (1013902103.209393)

Hardware source: 00:00:21:6c:5a:7f
Hardware destination: 00:80:ad:c8:19:74
Type /Length: 0x800 (IP)
Media length: 60

IP Header

Version: 4
Header length: 5 (20 bytes)
TOS: 0x00
Total length: 40
Identification: 11541
Fragmentation offset: 0
Unused bit: 0
Don't fragment bit: 1
More fragments bit: 0
Time to live: 128
Protocol: 6 (TCP)
Header checksum: 59246
Source address: 170.210.122.101
Destination address: 192.168.0.108

TCP Header

Source port: 1454 (unknown)
Destination port: 80 (http)
Sequence number: 388763
Acknowledgement number: 1412927352
Header length: 5
Unused: 0
Flags: A
Window size: 8760
Checksum: 52993
Urgent: 0

HTTP

Trama 11:

Ethernet (1013902103.258309)

Hardware source: 00:80:ad:c8:19:91
Hardware destination: 00:00:21:6c:5a:7f
Type /Length: 0x800 (IP)
Media length: 390

IP Header

Version: 4
Header length: 5 (20 bytes)
TOS: 0x00
Total length: 376
Identification: 11797
Fragmentation offset: 0
Unused bit: 0
Don't fragment bit: 1
More fragments bit: 0
Time to live: 128
Protocol: 6 (TCP)
Header checksum: 33016
Source address: 170.210.122.101
Destination address: 170.210.122.104

TCP Header

Source port: 1454 (unknown)

Destination port: 80 (http)
Sequence number: 388763
Acknowledgement number: 1412927352
Header length: 5
Unused: 0
Flags: PA
Window size: 8760
Checksum: 43972
Urgent: 0

HTTP

Header: GET /index.html HTTP/1.0
Header: If-Modified-Since: Sat, 26 Jan 2002 00:33:49 GMT; length=485
Header: Connection: Keep-Alive
Header: User-Agent: Mozilla/4.76 [en] (WinNT; U)
Header: Host: 170.210.122.104
Header: Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Header: Accept-Encoding: gzip
Header: Accept-Language: en
Header: Accept-Charset: iso-8859-1,*,utf-8

Trama 12:
Ethernet (1013902103.258859)

Hardware source: 00:00:21:6c:5a:7f
Hardware destination: 00:80:ad:c8:19:74
Type /Length: 0x800 (IP)
Media length: 390

IP Header

Version: 4
Header length: 5 (20 bytes)
TOS: 0x00
Total length: 376
Identification: 11797
Fragmentation offset: 0
Unused bit: 0
Don't fragment bit: 1
More fragments bit: 0
Time to live: 128
Protocol: 6 (TCP)
Header checksum: 58654
Source address: 170.210.122.101
Destination address: 192.168.0.108

TCP Header

Source port: 1454 (unknown)
Destination port: 80 (http)
Sequence number: 388763
Acknowledgement number: 1412927352
Header length: 5
Unused: 0
Flags: PA
Window size: 8760
Checksum: 4075
Urgent: 0

HTTP

Header: GET /index.html HTTP/1.0
Header: If-Modified-Since: Sat, 26 Jan 2002 00:33:49 GMT; length=485
Header: Connection: Keep-Alive
Header: User-Agent: Mozilla/4.76 [en] (WinNT; U)
Header: Host: 170.210.122.104
Header: Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Header: Accept-Encoding: gzip
Header: Accept-Language: en
Header: Accept-Charset: iso-8859-1,*,utf-8

Trama 13:

Ethernet (1013902103.259269)

Hardware source: 00:80:ad:c8:19:74
Hardware destination: 00:00:21:6c:5a:7f
Type /Length: 0x800 (IP)
Media length: 54

IP Header

Version: 4
Header length: 5 (20 bytes)
TOS: 0x00
Total length: 40
Identification: 0
Fragmentation offset: 0
Unused bit: 0
Don't fragment bit: 1
More fragments bit: 0

Time to live: 64
Protocol: 6 (TCP)
Header checksum: 21636
Source address: 192.168.0.108
Destination address: 170.210.122.101

TCP Header

Source port: 80 (http)
Destination port: 1454 (unknown)
Sequence number: 1412927352
Acknowledgement number: 389099
Header length: 5
Unused: 0
Flags: A
Window size: 6432
Checksum: 54985
Urgent: 0

Trama 14:
Ethernet (1013902103.259698)

Hardware source: 00:00:21:6c:5a:7f
Hardware destination: 00:80:ad:c8:19:91
Type /Length: 0x800 (IP)
Media length: 60

IP Header

Version: 4
Header length: 5 (20 bytes)
TOS: 0x00
Total length: 40
Identification: 0
Fragmentation offset: 0
Unused bit: 0
Don't fragment bit: 1
More fragments bit: 0
Time to live: 63
Protocol: 6 (TCP)
Header checksum: 61789
Source address: 170.210.122.104
Destination address: 170.210.122.101

TCP Header

Source port: 80 (http)

Destination port: 1454 (unknown)
Sequence number: 1412927352
Acknowledgement number: 389099
Header length: 5
Unused: 0
Flags: A
Window size: 6432
Checksum: 29347
Urgent: 0

HTTP

Trama 15:

Ethernet (1013902103.449884)

Hardware source: 00:80:ad:c8:19:74
Hardware destination: 00:00:21:6c:5a:7f
Type /Length: 0x800 (IP)
Media length: 683

IP Header

Version: 4
Header length: 5 (20 bytes)
TOS: 0x00
Total length: 669
Identification: 0
Fragmentation offset: 0
Unused bit: 0
Don't fragment bit: 1
More fragments bit: 0
Time to live: 64
Protocol: 6 (TCP)
Header checksum: 21007
Source address: 192.168.0.108
Destination address: 170.210.122.101

TCP Header

Source port: 80 (http)
Destination port: 1454 (unknown)
Sequence number: 1412927352
Acknowledgement number: 389099
Header length: 5
Unused: 0

Flags: PA
Window size: 6432
Checksum: 39864
Urgent: 0

HTTP

Header: HTTP/1.1 200 OK
Header: Date: Sat, 16 Feb 2002 23:28:23 GMT
Header: Server: Apache/1.3.17 (Unix) (SuSE/Linux)
Header: Connection: close
Header: Content-Type: text/html

Trama 16:

Ethernet (1013902103.451788)

Hardware source: 00:00:21:6c:5a:7f
Hardware destination: 00:80:ad:c8:19:91
Type /Length: 0x800 (IP)
Media length: 683

IP Header

Version: 4
Header length: 5 (20 bytes)
TOS: 0x00
Total length: 669
Identification: 0
Fragmentation offset: 0
Unused bit: 0
Don't fragment bit: 1
More fragments bit: 0
Time to live: 63
Protocol: 6 (TCP)
Header checksum: 61160
Source address: 170.210.122.104
Destination address: 170.210.122.101

TCP Header

Source port: 80 (http)
Destination port: 1454 (unknown)
Sequence number: 1412927352
Acknowledgement number: 389099
Header length: 5
Unused: 0
Flags: PA

Window size: 6432
Checksum: 14226
Urgent: 0

HTTP

Header: HTTP/1.1 200 OK
Header: Date: Sat, 16 Feb 2002 23:28:23 GMT
Header: Server: Apache/1.3.17 (Unix) (SuSE/Linux)
Header: Connection: close
Header: Content-Type: text/html

Trama 17:
Ethernet (1013902103.483036)

Hardware source: 00:80:ad:c8:19:74
Hardware destination: 00:00:21:6c:5a:7f
Type /Length: 0x800 (IP)
Media length: 54

IP Header

Version: 4
Header length: 5 (20 bytes)
TOS: 0x00
Total length: 40
Identification: 0
Fragmentation offset: 0
Unused bit: 0
Don't fragment bit: 1
More fragments bit: 0
Time to live: 64
Protocol: 6 (TCP)
Header checksum: 21636
Source address: 192.168.0.108
Destination address: 170.210.122.101

TCP Header

Source port: 80 (http)
Destination port: 1454 (unknown)
Sequence number: 1412927981
Acknowledgement number: 389099
Header length: 5
Unused: 0
Flags: FA
Window size: 6432

Checksum: 54355
Urgent: 0

Trama 18:

Ethernet (1013902103.483491)

Hardware source: 00:00:21:6c:5a:7f
Hardware destination: 00:80:ad:c8:19:91
Type /Length: 0x800 (IP)
Media length: 60

IP Header

Version: 4
Header length: 5 (20 bytes)
TOS: 0x00
Total length: 40
Identification: 0
Fragmentation offset: 0
Unused bit: 0
Don't fragment bit: 1
More fragments bit: 0
Time to live: 63
Protocol: 6 (TCP)
Header checksum: 61789
Source address: 170.210.122.104
Destination address: 170.210.122.101

TCP Header

Source port: 80 (http)
Destination port: 1454 (unknown)
Sequence number: 1412927981
Acknowledgement number: 389099
Header length: 5
Unused: 0
Flags: FA
Window size: 6432
Checksum: 28717
Urgent: 0

HTTP

Trama 19:

Ethernet (1013902103.484057)

Hardware source: 00:80:ad:c8:19:91
Hardware destination: 00:00:21:6c:5a:7f
Type /Length: 0x800 (IP)
Media length: 60

IP Header

Version: 4
Header length: 5 (20 bytes)
TOS: 0x00
Total length: 40
Identification: 12053
Fragmentation offset: 0
Unused bit: 0
Don't fragment bit: 1
More fragments bit: 0
Time to live: 128
Protocol: 6 (TCP)
Header checksum: 33096
Source address: 170.210.122.101
Destination address: 170.210.122.104

TCP Header

Source port: 1454 (unknown)
Destination port: 80 (http)
Sequence number: 389099
Acknowledgement number: 1412927982
Header length: 5
Unused: 0
Flags: A
Window size: 8131
Checksum: 27018
Urgent: 0

HTTP

Trama 20:

Ethernet (1013902103.484294)

Hardware source: 00:00:21:6c:5a:7f
Hardware destination: 00:80:ad:c8:19:74
Type /Length: 0x800 (IP)
Media length: 60

IP Header

Version: 4
Header length: 5 (20 bytes)
TOS: 0x00
Total length: 40
Identification: 12053
Fragmentation offset: 0
Unused bit: 0
Don't fragment bit: 1
More fragments bit: 0
Time to live: 128
Protocol: 6 (TCP)
Header checksum: 58734
Source address: 170.210.122.101
Destination address: 192.168.0.108

TCP Header

Source port: 1454 (unknown)
Destination port: 80 (http)
Sequence number: 389099
Acknowledgement number: 1412927982
Header length: 5
Unused: 0
Flags: A
Window size: 8131
Checksum: 52656
Urgent: 0

HTTP

Trama 21:
Ethernet (1013902104.041686)

Hardware source: 00:80:ad:c8:19:91
Hardware destination: 00:00:21:6c:5a:7f
Type /Length: 0x800 (IP)
Media length: 60

IP Header

Version: 4
Header length: 5 (20 bytes)
TOS: 0x00
Total length: 40
Identification: 12309

Fragmentation offset: 0
Unused bit: 0
Don't fragment bit: 1
More fragments bit: 0
Time to live: 128
Protocol: 6 (TCP)
Header checksum: 32840
Source address: 170.210.122.101
Destination address: 170.210.122.104

TCP Header

Source port: 1454 (unknown)
Destination port: 80 (http)
Sequence number: 389099
Acknowledgement number: 1412927982
Header length: 5
Unused: 0
Flags: FA
Window size: 8131
Checksum: 27017
Urgent: 0

HTTP

Trama 22:

Ethernet (1013902104.041894)

Hardware source: 00:00:21:6c:5a:7f
Hardware destination: 00:80:ad:c8:19:74
Type /Length: 0x800 (IP)
Media length: 60

IP Header

Version: 4
Header length: 5 (20 bytes)
TOS: 0x00
Total length: 40
Identification: 12309
Fragmentation offset: 0
Unused bit: 0
Don't fragment bit: 1
More fragments bit: 0

Time to live: 128
Protocol: 6 (TCP)
Header checksum: 58478
Source address: 170.210.122.101
Destination address: 192.168.0.108

TCP Header

Source port: 1454 (unknown)
Destination port: 80 (http)
Sequence number: 389099
Acknowledgement number: 1412927982
Header length: 5
Unused: 0
Flags: FA
Window size: 8131
Checksum: 52655
Urgent: 0

HTTP

Trama 23:

Ethernet (1013902104.042630)

Hardware source: 00:80:ad:c8:19:74
Hardware destination: 00:00:21:6c:5a:7f
Type /Length: 0x800 (IP)
Media length: 54

IP Header

Version: 4
Header length: 5 (20 bytes)
TOS: 0x00
Total length: 40
Identification: 0
Fragmentation offset: 0
Unused bit: 0
Don't fragment bit: 1
More fragments bit: 0
Time to live: 64
Protocol: 6 (TCP)
Header checksum: 21636

Source address: 192.168.0.108
Destination address: 170.210.122.101

TCP Header

Source port: 80 (http)
Destination port: 1454 (unknown)
Sequence number: 1412927982
Acknowledgement number: 389100
Header length: 5
Unused: 0
Flags: A
Window size: 6432
Checksum: 54354
Urgent: 0

Trama 24:
Ethernet (1013902104.043129)

Hardware source: 00:00:21:6c:5a:7f
Hardware destination: 00:80:ad:c8:19:91
Type /Length: 0x800 (IP)
Media length: 60

IP Header

Version: 4
Header length: 5 (20 bytes)
TOS: 0x00
Total length: 40
Identification: 0
Fragmentation offset: 0
Unused bit: 0
Don't fragment bit: 1
More fragments bit: 0
Time to live: 63
Protocol: 6 (TCP)
Header checksum: 61789
Source address: 170.210.122.104
Destination address: 170.210.122.101

TCP Header

Source port: 80 (http)

Destination port: 1454 (unknown)
Sequence number: 1412927982
Acknowledgement number: 389100
Header length: 5
Unused: 0
Flags: A
Window size: 6432
Checksum: 28716
Urgent: 0

HTTP
