UNIVERSIDAD NACIONAL DE INGENIERIA

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

OPTIMIZACIÓN DE ACCESIBILIDAD DE RECURSOS DE INFORMACIÓN DIGITAL PÚBLICA EN ALTO TRÁFICO DE INTERNET

INFORME DE SUFICIENCIA

PARA OPTAR EL TÍTULO PROFESIONAL DE:

INGENIERO ELECTRÓNICO

PRESENTADO POR:

MARTÍN JIMÉNEZ BRAVO

PROMOCIÓN
2002-I

LIMA-PERÚ
2010

OPTIMIZACIÓN DE ACCESIBILIDAD DE RECURSOS DE INFORMACIÓN DIGITAL PÚBLICA EN ALTO TRÁFICO DE INTERNET

Agradezco a mis padres y hermanos
por encaminarme en una vida de trabajo y honestidad,
así cómo también a mi alma Mater
la Universidad Nacional de Ingeniería
a mis profesores y compañeros

SUMARIO

El presente trabajo tiene como objetivo optimizar el acceso de los recursos de información digital (revistas electrónicas, biblioteca digital, portal de tesis electrónicas, periódicos, etc.) cuando el tráfico de consultas es muy elevado. Asimismo proveer la seguridad de la información digital y una transparente escalabilidad.

Todo recurso de información digital (RID) que es consultado en grandes cantidades termina por hacer colapsar al servidor que aloja al RID. La alternativa de solución de colocar más servidores con la misma información para distribuir la carga tiene cómo desventaja el tedioso mantenimiento de los RID, además al existir interactividad entre el usuario y el servidor de forma directa, la seguridad de los servidores se ve comprometida. Otra desventaja de esta solución es la deficiente escalabilidad y el mayor consumo de recursos humanos para la gestión y administración de los servidores.

La metodología para la solución del problema es la colocación de servidores que guarden en memoria caché (sistema especial de almacenamiento de alta velocidad) la información que será utilizada por los usuarios. Todas las consultas hechas a los RID serán repartidas por un servidor balanceador a los servidores caché los cuales contendrán la información digital en su memoria. Si el servidor caché no tuviese la información requerida entonces esta será extraída del servidor real para tenerlo en memoria caché para las nuevas consultas que se den. Para lograr la alta disponibilidad se hace necesario poner otro servidor balanceador que realice la misma tarea.

ÍNDICE

INTRODUCCIÓN 1

CAPÍTULO I

MARCO TEÓRICO CONCEPTUAL 3

1.1 Planteamiento del Problema 3

1.1.1 Objetivos del Trabajo 3

1.1.2 Evaluación del Problema 3

1.1.3 Alcances 4

1.1.4 Sinopsis del trabajo 4

1.2 RID, los recursos de información digital 5

1.3 Capas de red y protocolos 5

1.4 LVS (Linux Virtual server) 6

1.4.1 Arquitectura 6

1.4.2 Mecanismos de balanceo de carga 7

1.4.3 Algoritmos de Balanceo de carga 8

1.4.4 Alta disponibilidad 9

1.4.5 Pirahna 9

1.5 Protocolos de aplicación para la web 9

1.5.1 Localizador Uniforme de Recursos (URL) 10

1.5.2 Sistema de nombres de dominio 11

1.5.3 Protocolo de transferencia de hipertexto 11

1.5.4 Intercambio de mensajes HTTP 12

1.5.5 Hiperenlaces y objetos incrustados 12

1.6 Soporte HTTP para almacenamiento (caching) y replicación 12

1.6.1 Solicitudes condicionales 12

1.6.2 Edad y tiempo de expiración de los objetos almacenados 14

1.6.3 Redirección de requerimientos 16

1.6.4 Rango de requerimientos 17

1.6.5 La cabecera cache-control 17

1.6.6 Cookies 22

1.6.7 Soporte para servidores compartidos 24

1.6.8 Identificadores de objetos expandido 24
1.6.9 Aprendizaje de la cadena de proxy 25
1.6.10 Cacheabilidad de contenido web 26
1.7 Reglas de comportamiento web para el caching 27
1.7.1 Tamaño del objeto 28
1.7.2 Tipos de objeto y cacheabilidad 30
1.7.3 Popularidad del objeto 31
1.7.4 Localidad de referencia 32
1.7.5 Tasa de modificaciones de objetos 34
CAPÍTULO II
OPTIMIZACIÓN DE LA ACCESIBILIDAD A LOS RID 35
2.1 Ubicación del problema en un escenario inicial 35
2.2 Análisis del problema 35
2.3 Planteamiento de una solución al problema 36
2.4 Selección de equipos 37
2.4.1 El LVS - Linux Virtual Server 37
2.4.2 El Servidor cache 38
2.4.3 El Servidor real 38
CAPÍTULO III
RESULTADOS 39
3.1 Escalabilidad 39
3.2 Alta disponibilidad 39
3.3 Balanceo de carga web 40
3.4 Tiempo de descarga de objetos 40
3.5 Almacenamiento de objetos en cache 40
3.6 Número de peticiones por segundo 40
3.7 Tiempo de expiración de objetos 40
3.8 Compresión de objetos 41
3.9 Servidor real 41
CONCLUSIONES Y RECOMENDACIONES 42
ANEXO A
INSTALACIONES Y CONFIGURACIONES 43
ANEXO B
GLOSARIO DE TÉRMINOS 59
BIBLIOGRAFÍA 61

INTRODUCCIÓN

Cuando el tráfico de consultas en un servidor de datos es muy elevado, el acceso de los recursos de información digital (revistas electrónicas, biblioteca digital, portal de noticias, etc.) es ineficiente. Existen periodos durante los cuales la tasa de consultas es tan grande que hace imposible que el usuario pueda verlos

Entre los RID o recursos de información digital, pueden ser considerados aquellos existentes en los portales de Internet, tales cómo música, noticias, venta de artículos en línea, bibliotecas digitales, juegos, películas, series televisivas y cualquier otro recurso de interés para el usuario.

Si el usuario detecta su imposibilidad momentánea de acceso a dichos recursos, puede no regresar a dicho portal de Internet.

Dado que el propósito de brindar RID de manera gratuita es para hacer propaganda de bienes y servicios de diversa índole, por la cual los anunciantes pagan una cantidad de dinero, un problema de acceso a cualquier portal se reflejará en la pérdida de clientes potenciales para dichos productos, al migrar a otros portales que no muestran dificultad alguna.

Para evitar esta problemática, se opta por la colocación de servidores que guarden en una memoria especial llamada “caché” (sistema especial de almacenamiento de alta velocidad) la información que será utilizada por los usuarios.

Todas las consultas hechas a los RID serán repartidas por un equipo denominado “servidor balanceador” a servidores caché que contendrán la información digital en sus memorias. Si al momento de realizarse una consulta al servidor caché, este no tuviese la información solicitada, ésta será obtenida del servidor real para así tenerlo en memoria para las nuevas consultas que se den.

Considerando que es sumamente importante que el portal no “caiga”, es decir que no pierda su capacidad de brindar servicio, se hace necesaria brindar al sistema una alta disponibilidad, para lo cual es imprescindible la colocación de otro servidor (no caché) que efectúe similar tarea. Este servidor da cara a Internet

Para la explicación del diseño se propondrá un escenario específico para el cual el sistema a describir permita almacenar en caché el 95% de los objetos estáticos (audio, video, páginas web, etc.), mejorando los tiempos de respuesta en más de 50%. La solución presentada es desarrollada mediante la utilización de software libre. Los detalles

serán explicados en el capítulo correspondiente.

Este informe de suficiencia ha sido desarrollado teniendo en consideración todas las especificaciones técnicas proporcionadas en los manuales técnicos, los protocolos actuales de Internet y la experiencia y conocimientos adquiridos durante los cuatro años de carrera profesional en empresas dedicadas a tecnologías de la información.

El informe de suficiencia se divide en cuatro capítulos. El capítulo 1 corresponde a la explicación de la problemática, y en el cual se hace una sinopsis del sistema propuesto, también presenta las bases teóricas indispensables para la comprensión del diseño. En el capítulo 2 se expone el diseño en sí, explicando todas las consideraciones técnicas en un escenario propuesto. En el capítulo 3 se muestran los resultados logrados en base al diseño de la solución.

CAPÍTULO I
MARCO TEÓRICO CONCEPTUAL

En el este capítulo se explica la problemática, haciendo una sinopsis del sistema propuesto, también se exponen los fundamentos teóricos básicos e imprescindibles para el entendimiento del informe de suficiencia. Los temas a tratar son los siguientes:

1. Planteamiento del problema.
2. RID (Recursos de información digital)
3. Capas de red y protocolos
4. LVS (Linux Virtual Server)
5. Protocolos de aplicación para la web
6. Soporte HTTP para almacenamiento (caching) y replicación
7. Reglas de comportamiento web para el caching

1.1 Planteamiento del problema

El problema es la demora o imposibilidad de acceso a los recursos de información digital (RID) en los portales de Internet bajo una alta demanda de consultas.

Ante una alta tasa de consultas en un portal de Internet de los recursos que él provee, el acceso a estos, sean revistas electrónicas, bibliotecas digitales, portales de noticias, periódicos, música, páginas web, etc., es ineficiente lo. Hay periodos en los cuales la tasa de consultas es tan grande que ocasiona un cuello de botella o incluso un colapso del servicio.

1.1.1 Objetivo del Trabajo

Diseñar un sistema que 1) optimice la accesibilidad a los recursos de información digital pública en alto tráfico de Internet mediante la utilización de servidores caché, 2) proveer alta disponibilidad de dichos recursos mediante la utilización de servidores redundantes (que dan cara a Internet y reparten equitativamente las consultas a los servidores caché), 3) Reducir costos de implementación mediante el uso de software libre

1.1.2 Evaluación del Problema

Los RID (recursos de información digital) son aquellos contenidos en los portales de Internet para una libre disponibilidad de parte de cualquier usuario (información pública). Estos portales contienen música, noticias, venta de artículos en línea, bibliotecas digitales, juegos, películas, series televisivas y todo recurso de interés para el usuario.

El propósito de brindar RID de manera gratuita es para hacer propaganda de bienes y servicios de diversa índole, por la cual los anunciantes pagan una cantidad de dinero. La publicidad es similar a la de otros medios de información (radio, tv) , en los cuales existe una programación diversa, y en las cuales se difunde propaganda de todo tipo.

Si existiera un problema de acceso a cualquier portal esto traería cómo consecuencia la pérdida de clientes potenciales para dichos productos. El usuario al experimentar una imposibilidad momentánea de acceso a los recursos públicos, podría no regresar a dicho portal de Internet.

El colapso de los servidores sin memoria caché, puede ocurrir por una alta demanda de visitas al portal web. Esto sucedió en muchas oportunidades, principalmente cuando una noticia de gran impacto, sea de desastre o de farándula concita la atención y expectativa de los usuarios.

Mantener el servicio ante cualquier situación es de suma importancia para los anunciantes, quienes contratan con el portal de Internet la colocación de sus avisos.

1.1.3 Alcances

El diseño se basará en un escenario específico al cual se le permitirá almacenar en caché el 95% de los objetos estáticos (audio, video, páginas web, etc.), mejorando así los tiempos de respuesta en más de 50%.

1.1.4 Sinopsis del trabajo

Se opta por la colocación de tres servidores de caché que guarden en una memoria la información más consultada por los usuarios. La Figura 1.1 muestra el sistema propuesto. Los nuevos elementos se encuentran dentro del círculo.

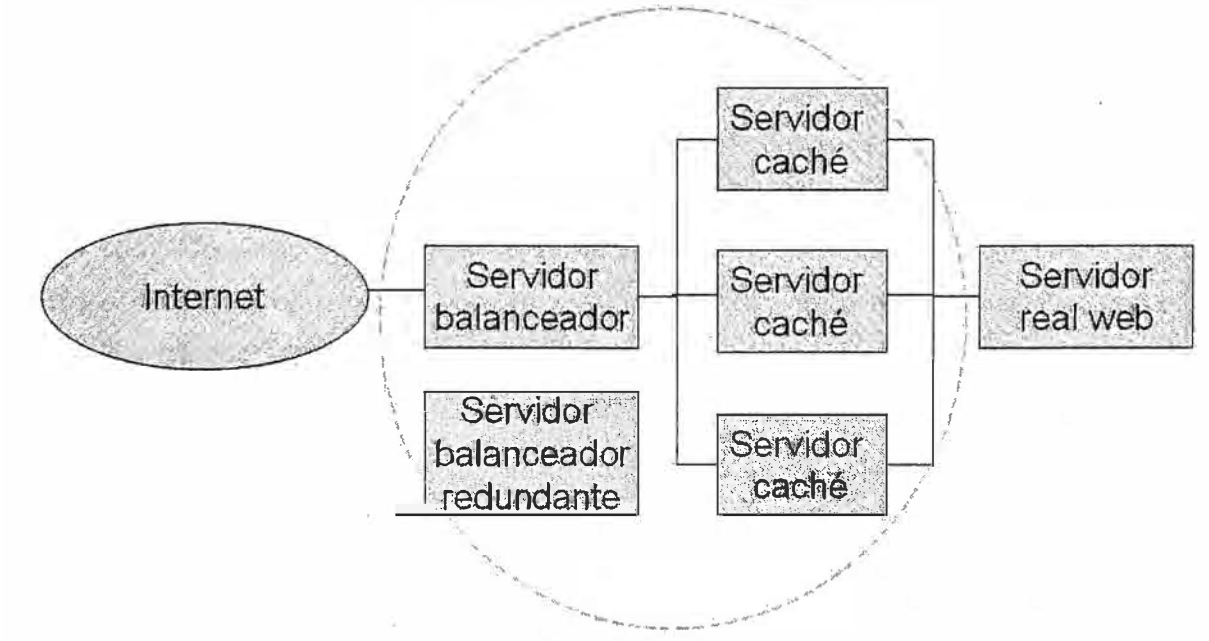

Figura 1.1 Sistema propuesto

Todas las consultas hechas a los RID serán repartidas por un equipo denominado

"servidor balanceador" a tres servidores caché que contendrán la información digital en sus memorias. Si al momento de realizarse una consulta al servidor caché, este no tuviese la información solicitada, ésta será obtenida del servidor real para así tenerlo en memoria para las nuevas consultas que se den. La solución presentada es desarrollada mediante la utilización de software libre.

1.2 RID, los recursos de información digital

Un recurso digital es cualquier tipo de información que se encuentra almacenada en formato digital, como por ejemplo páginas web, bibliotecas digitales, tesis electrónicas, revistas electrónicas, música, películas, etc.

Cómo ejemplo, el SISIB (Sistema de Servicio de Información y Bibliotecas), a través del portal de la Universidad (http://bibliotecas.uchile.cl), ofrece acceso a una variada gama de recursos de información académicos y científicos en formato electrónico y en texto completo, contando con más de 7.000 publicaciones.

Estas publicaciones se encuentran disponibles a través de los siguientes servicios:

Web institucional	(http://www.uchile.cl/)
Catálogo en línea institucional.	(http://catalogo.uchile.cl/)
Bases de datos especializadas	(http://bibliotecas.uchile.cl/)
Al-Día. Revistas Especializadas	(http://www.al-dia.cl)
Revistas electrónicas	(http://yb9vv9jh8j.search.serialssolutions.com/)
Biblioteca Digital	(http://www.bibliotecadigital.cl)
Tesis electrónicas	(http://www.cybertesis.cl)
Portal de tesis electrónicas	(http://www.cybertesis.net)
Museo de Arte Contemporáneo	(http://www.mac.uchile.cl)
Museo de Arte Popular	(http://www.mapa.uchile.cl)

Otro ejemplo se tiene en los portales web de almacenamiento masivo de imágenes, cómo Hi5 (40 millones de usuarios) y Facebook (350 millones de usuarios. Portales de videos cómo Youtube (300 millones de usuarios mas de 120 millones de videos).También se debe mencionar a los que brindan alojamiento para todo tipo de archivo digital, tal cómo los portales web Rapidshare, Megaupload, Wikifortio, etc. Finalmente están aquellos de consulta de información textual tal cómo Wikipedia, blogs, entre otros.

1.3 Capas de red y protocolos

Las redes de computadoras y protocolos de red son extremadamente complejos. Los diseñadores de red, se dieron cuenta desde el inicio que, para poder manejar esta complejidad, necesitaban definir algunas abstracciones.

Así, el concepto de capas fue introducido para proporcionar las funcionalidades, cada vez más complejas, a través de interfaces estandarizadas. A través de esto, la

funcionalidad de red es implementada como una "pila" de capas (stack de capas), donde cada capa usa una interfaz estándar proporcionada por la capa ubicada directamente debajo de ella en la pila (stack) y presenta una interfaz estándar a la capa ubicada directamente arriba de ella. Las capas ocultan la complejidad de cada capa a la siguiente capa de la pila [1].

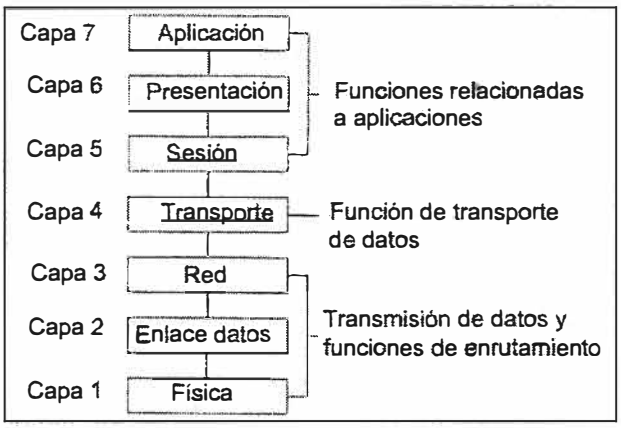

Figura 1.2 El modelo de referencia ISO/OSI

1.4 LVS (Linux Virtual server)

A continuación se enumeran los principales requerimientos que debe tener un sistema de alta disponibilidad para hacer frente a la inminente demanda de conexiones y servicios de Internet [2][3][4]:

a. Escalabilidad. Para cuando la demanda de un determinado servicio aumenta, el sistema debe ofrecer la capacidad de ampliarse con la finalidad de atender todas las solicitudes sin que los tiempos de respuesta sean alterados.

b. Alta Disponibilidad. Capacidad del sistema de estar disponible en todo momento, tomando en cuenta de que debe ser un sistema altamente tolerante a fallos.

c. Costos. Uno de los requerimientos fundamentales a la hora de hacer una inversión, los costos de montaje, mantenimiento y expansión del sistema deben ser accesibles en comparación a infraestructuras propietarias.

1.4.1 Arquitectura

La arquitectura del cluster de servidores virtuales está conformada por dos componentes fundamentales como se muestra en la Figura 1.3.

a. Servidor Virtual. Servidor que da la cara al exterior, es decir quien recibe en primer plano los requerimientos del cliente. Éste se caracteriza por tener configurada una dirección IP pública mediante la cual se publica el servicio ofrecido. Este servidor se encarga de recibir múltiples conexiones y redirigirlas a los nodos mediante un proceso de balanceo de carga.

b. Nodos. Servidores que se encuentran detrás del servidor virtual y cuya principal función es la de ejecutar el servicio que se está ofreciendo.

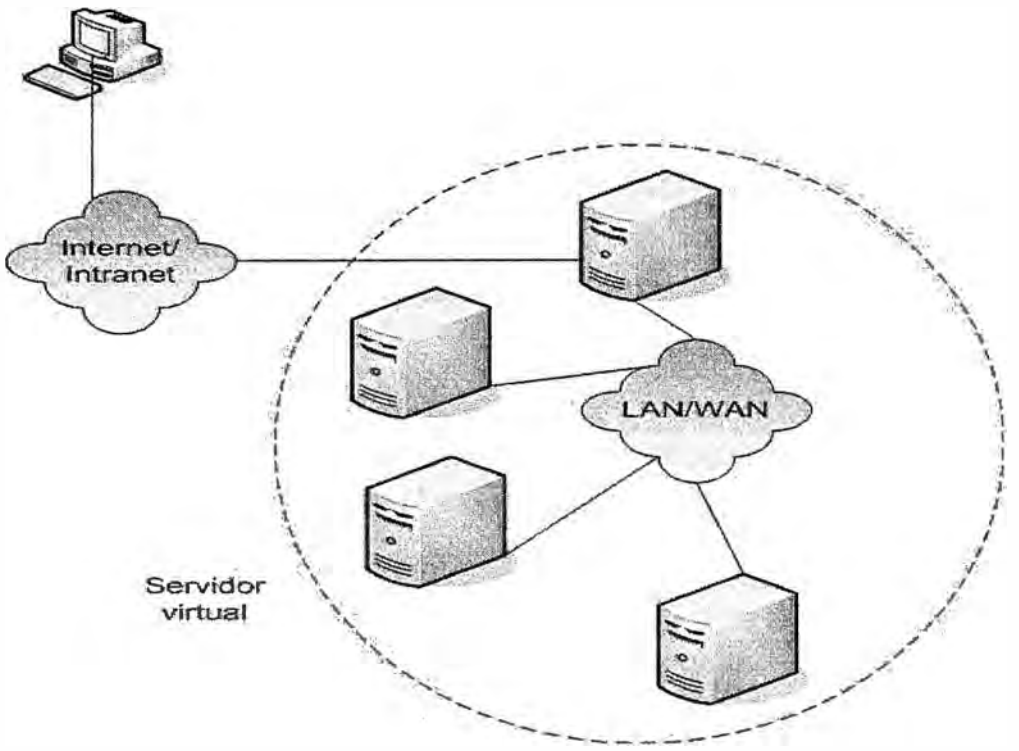

Figura 1.3 Arquitectura del LVS

Como se puede apreciar en la Figura 1.3, el sistema da la cara al usuario mediante un único servidor virtual conocido también como nodo director, éste se encarga del balanceo de carga y reenvía las peticiones a los nodos que conforman el cluster. El nodo director hace las funciones de ruteador, el cual basado en reglas de enrutamiento redirecciona el tráfico entrante a los nodosles. El momento que la comunicación se establece esta perdura durante el periodo que la conexión TCP se utiliza y cuando se inicia una nueva conexión el nodo director selecciona un nuevo nodo diferente al anterior.

Con esta arquitectura, además del balanceo de carga, estamos permitiendo que los servidores reales individuales puedan ser extraídos del LVS, actualizados o reparados y devueltos al cluster sin interrumpir la prestación de servicio. Asimismo, el sistema es fácilmente escalable. Para brindar alta disponibilidad en LVS se utiliza enlaces de respaldo o de latido de corazón (heartbeat) entre los nodos directores.

1.4.2 Mecanismos de balanceo de carga

Los mecanismos son los siguientes:

a. Balanceo por NAT

b. Balanceo por encapsulado IP

c. Balanceo por enrutamiento directo

a. Balanceo por NAT

Balanceo de carga donde el nodo director tenga un comportamiento de enrutador con NAT (Network Address Traslation); el nodo director utiliza una extensión del NAT conocida como Traducción de Direcciones de Red y Puerto o NAPT (Network Address

Port traslation), esta se caracteriza por usar una sola dirección IP válida con la que los host se conectan desde la red interna hacia el Internet por diferentes puertos.

NAPT se basa en el mapeo de direcciones de red y puertos TCP/UDP de los paquetes en otros puertos NAPT es una técnica que permite que numerosos servicios se ejecuten en distintas direcciones de red y se escuchen en distintos puertos y, a su vez puedan ser trasladados a un único puerto en la dirección de red del servidor virtual para lograr la apariencia de un único servicio.

b. Balanceo por encapsulado IP

Técnica caracterizada por encapsular un datagrama IP dentro de otro, es decir el encapsulado IP consiste en enmascarar una trama TCP/IP, con sus direcciones IP de origen y destino, dentro de otra trama con direcciones distintas, una vez que la trama más externa llegue a su destino, desencapsular la trama original y reenrutarla desde allí. Como requisito principal esta técnica requiere que todos los dispositivos soporten el protocolo IP tunneling o IPSec.

c. Balanceo por enrutamiento directo

Mecanismo de balanceo de carga, caracterizado por utilizar una red LAN con sus respectivos dispositivos de interconexión como un Switch/Hub para la interconexión entre el nodo director y sus nodos.

La característica de este método radica en que los nodos poseen configurado en el interfaz de red local, la dirección IP real del nodo director, pero como en el caso de balanceo por encapsulamiento IP deben estar configuradas para que no respondan a mensajes del protocolo ARP.

1.4.3 Algoritmos de Balanceo de carga

Los algoritmos son los siguientes:

a. Round Robin

b. Round Robin ponderado

c. Servidor con menos conexiones activas

d. Servidor con menos conexiones activas (ponderado)

a. Round Robin

También llamado FIFO (First In First Out), proceso por el cual el nodo director envía una petición al primer nodo (servidor) del cluster, la siguiente petición se envía al siguiente servidor disponible y así sucesivamente, hasta llegar al último servidor disponible, luego del cual se vuelve a enviar al primero.

Este mecanismo es el más sencillo de todos pero no el más justo ya que se puede dar el caso en el que toda la carga pesada vaya a un determinado servidor, mientras que el resto recibe peticiones más sencillas.

b. Round Robin ponderado

Algoritmo cuyo funcionamiento es igual al Round Robin, con la diferencia que a cada servidor se le da un indicativo de acuerdo a su potencia de cálculo; es decir servidores con alta potencia de calculo recibirán mayor carga que los que poseen una potencia menor. Pero este método también tiene la desventaja de que los servidores más capaces reciban mas carga y con ello llegar a saturarlos.

c. Servidor con menos conexiones activas

Proceso caracterizado por tener un mecanismo continuo de consulta acerca de cuántas conexiones activas poseen los servidores que están atendiendo los requerimientos del cliente, dependiendo de esta consulta el nodo director direcciona las peticiones a los servidores con menos cargas. Pero este proceso es muy útil cuando todos los servidores poseen una potencia de cálculo similar, logrando tener de esta manera un trabajo equilibrado.

El inconveniente surge cuando la potencia de los servidores no es la misma en todos, en este ámbito los servidores más rápidosrecibirán un mayor número de conexiones activas, así como conexiones en espera, los servidores más lentos tendrán un número menor de conexiones activas y en espera, lo cual que el nodo director envíe más carga a estos servidores por el hecho de tener pocas conexiones y con ello se logra que estos lleguen a saturarse.

d. Servidor con menos conexiones activas (ponderado)

Algoritmo que basa su funcionamiento de acuerdo al menor número de conexiones activas y de un determinado indicativo que se le da a los servidores en base a su capacidad de cálculo. Este proceso consiste en asignar una mayor carga a los servidores con menos conexiones activas pero verificando la capacidad de cálculo de éstos, y dependiendo de esta capacidad asignar una mayor o menor carga de los servidores.

1.4.4 Alta disponibilidad

Heartbeat, herramienta que ofrece alta disponibilidad y basa su funcionamiento en el monitoreo de dos nodos para conocer si un equipo se encuentra funcionando o no.

1.4.5 Pirahna

Paquete cuyo propietario es Red Hat. Inc, el cual incluye un conjunto de herramientas utiles para implementar un cluster de alta disponibilidad y balanceo de carga. Basa su funcionamiento en LVS y otros programas de propiedad de Red Hat.

1.5 Protocolos de aplicación para la web

La web es una aplicación que se ejecuta sobre Internet y es implementada utilizando un conjunto de protocolos y estándares de la capa de aplicación. Algunos protocolos de la capa de aplicación tales como DNS, NFS, Telnet, FTP, Gopher y HTTP, fueron

discutidos anteriormente. [5] [6]

1.5.1 Localizador Uniforme de Recursos (URL)

URLs, Uniform Resource Locators, o localizadores uniformes de recursos, permiten la denominación de objetos arbitrarios dentro de una red mundial. Muchos URLs están especificados en los objetos Web y, o bien se descargan automáticamente por los navegadores, o son sólo visibles a un usuario final de forma indirecta como hipervínculos. Si las URL se anuncian directamente a los usuarios, por lo general están incompletos y se confían en un navegador Web para ampliarlo a su forma correcta.

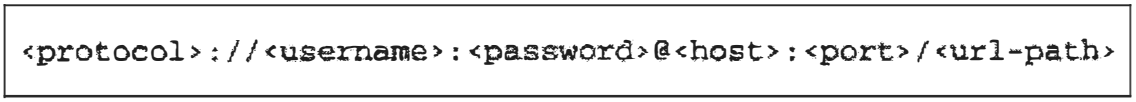

Figura 1.4 Sintaxis general del URL

La Figura 1.4 muestra la sintaxis general de una URL como indica el CFC 1738 y posteriormente actualizado en el RFC 1808 y RFC 2368. La dirección URL contiene información sobre el nombre del objeto y cómo este debe ser recuperado. Se inicia con el campo protocolo. El campo protocolo determina el protocolo que será utilizado para acceder al objeto. Las posibilidades comunes son file, news, mailto, http y ftp. El protocolo file se utiliza para recuperar los archivos locales, mailto para enviar correo electrónico, HTTP para descargar los objetos Web, y el protocolo FTP se utiliza para recuperar objetos desde un servidor FTP.

El campo protocolo es seguido por los campos opcionales username y password. Estos campos permiten la autenticación de un usuario, lo que permite una forma simple de control de acceso para el objeto. Una limitación importante de este esquema de autenticación es que el nombre de usuario y contraseña son transmitidos de de forma insegura a través de Internet. Por esta razón, en la práctica, este sistema se utiliza muy poco, y el nombre de usuario y contraseña son campos que normalmente se omiten, junto con los delimitadores asociados, los dos puntos (:) y el signo de arroba (@).

El siguiente campo URL es el campo hosts. Este campo contiene la dirección IP o el nombre de dominio completo (fully qualified domain name) del host donde reside el documento.

El campo host es seguido por un campo puerto que identifica el número de puerto utilizado por el servidor para aceptar conexiones de clientes para este objeto. En muchos casos, el puerto se omite, ya que todos los protocolos comunes tienen un puerto por default conocido.

El último campo del URL, url-path, identifica el objeto dentro del host. El campo URL-path puede ser tan simple como un nombre de archivo o puede ser un protocolo – o host- especificando código de de argumentos utilizados por el servidor para generar

dinámicamente el objeto.

1.5.2 Sistema de nombres de dominio

Hasta ahora se ha visto cómo un host individual se identifica por su dirección IP. Esta dirección numérica está perfectamente adecuada para ser procesada por los routers o computadoras, pero esta no es muy amigable para el usuario final. Por lo tanto, es deseable que, además de la dirección IP, los host tengan un nombre simbólico.

Por ejemplo, el servidor Web de UNI es conocido por muchos usuarios como www.uni.edu.pe y no como 69.93.97.199. El mapeo entre el nombre simbólico y la dirección IP es realiza por el Domain Name System (DNS).

El trabajo del DNS es mapear nombre de dominio de los host a direcciones IP (esto también se llama resolver un nombre de domino). Los hosts obtienen ese mapeo interactuando con el DNS utilizando el protocolo DNS.

El protocolo DNS incluye dos partes principales: un cliente host que quiere resolver un nombre de dominio a una dirección IP y un conjunto de Servidores de nombres, también llamados Servidores DNSs, que proveen la resolución.

El protocolo DNS típicamente corre sobre UDP. Un servidor DNS recibe un pedido para resolver un nombre de dominio desde un cliente o desde otro servidor DNS y retorna uno o más registros como respuesta. Esos registros (records) pueden tener una amplia variedad de tipos de registros. Los más comunes son:

a) El registro A es utilizado para devolver un nombre a su respectivo IP

b) El registro NS que refiere el pedido a otro servidor DNS que conoce como resolver los subdominios dentro del dominio.

c) El registro CNAME que mapea un nombre de dominio a otro nombre de dominio.

1.5.3 Protocolo de transferencia de hipertexto

Debido a que la Web se ha convertido en el motor de la Internet, el Protocolo de transferencia de Hipertexto (HTTP) ahora tiene una gran mayoría (alrededor del 70%) del tráfico de Internet. HTTP es un protocolo simple de requerimiento-respuesta que utiliza URLs para llamar los objetos que recupera. HTTP 0.9 fue la primera versión utilizada ampliamente, la cual fue reemplazada por HTTP 1.0 y luego por HTTP 1.1

Un mensaje HTTP consiste de una cabecera HTTP y un cuerpo, los dos están delineados por los caracteres de "retorno de carro" CR y el "retorno de línea" LF. Hay dos tipos de mensajes HTTP: una petición HTTP emitida por un cliente y una respuesta HTTP generada por un servidor. Para ambos tipos de mensajes HTTP, todos los campos de la cabecera están codificados en texto plano (ASCII) para que los mensajes sean fácilmente legibles para los humanos.

1.5.4 Intercambio de mensajes HTTP

La interacción HTTP básica incluye un requerimiento del cliente y una respuesta del servidor. La simplicidad del protocolo resulta de ser "stateless": no requiere que el cliente o el servidor registren alguna información más allá de los límites de esta simple interacción.

1.5.5 Hiperenlaces y objetos incrustados

HTTP puede ser utilizado para recuperar objetos arbitrarios incluyendo imágenes, archivos JavaScript, o páginas HTML. Mientras la codificación y otros detalles de esos objetos están fuera del alcance de este estudio, esto es importante en el contexto de caching y replicación para comprender la funcionalidad básica de hiperenlaces y objetos embebidos en los documentos HTML.

Un documento HTML es el framework básico de una página Web. En general, este contiene el texto, las instrucciones de formato, hiperenlaces y objetos embebidos mostrados en un navegador Web como parte del documento.

1.6 Soporte HTTP para almacenamiento (caching) y replicación

En los capítulos previos se discutió aspectos básicos de Internet, incluyendo una introducción básica a HTTP. Esta parte se enfocará en las características más importantes de HTTP que tienen interés para el almacenamiento (caching) y replicación. Si bien muchas de estas características pueden ser utilizadas por una amplia variedad de propósitos, esta sección se enfocará en los usos que dan soporte al almacenamiento (caching) y replicación [7][8].

1.6.1 Solicitudes condicionales

Una característica importante de HTTP para el caching y la replicación es el soporte para "requerimientos condicionales". Estos requerimientos especifican ciertas condiciones en sus cabeceras. Un servidor ejecuta un requerimiento condicional y responde con el cuerpo del mensaje sólo si la condición es verdadera. De cualquier otra forma, el servidor simplemente responde con un código de estatus especial, o "304 Not modified" ó "412 Precondition Failed", los cuales indican que la condición no se ha cumplido. Las cabeceras que especifican las condiciones son llamadas *cabeceras condicionales.*

Los clientes realizan requerimientos condicionales utilizando metainformación relacionada a los objetos. Los servidores proporcionan esta información junto con los objetos en las cabeceras last-modified y Etag de sus respuestas. La cabecera last-modified contiene la fecha y hora de la última modificación del objeto. ETag, que significa etiqueta de entidad, es un identificador único para una instancia particular de un objeto.

Por ejemplo, la función hash MD5 puede ser utilizada para generar un identificador digital único, o huella digital, de un objeto, el cual puede servir como su etiqueta de

entidad.

Las etiquetas de entidad proporcionan una manera más general y confiable para identificar las versiones de un objeto, ya que las fechas de last-modified tienen una granularidad de un segundo, lo que hace imposible identificar los cambios que ocurren en menos de un segundo desde un acceso previo al objeto.

MD5 es un algoritmo criptográfico de 128bits; este algoritmo es usado para mapear con extremadamente alta probabilidad diferentes objetos dentro de diferentes valores.

a. Cabeceras condicionales utilizadas para caching

La Tabla 1.1 lista las cabeceras condicionales más importantes utilizadas en el contexto de caching. La Cabecera if-modified-since contiene la fecha de última modificación del objeto almacenado. Cuando un servidor recibe un requerimiento con sus cabeceras (típicamente un requerimiento GET), el servidor sólo retorna el objeto si la fecha de última modificación (last-modified) del objeto en el servidor es diferente de la fecha contenida en la cabecera if-modified-since del requerimiento.

Tabla 1.1 Cabeceras comunes condicionales en almacenamiento (caching)

Cabecera	Valor	Significado
if-modified-since	Last.modified date	Ejecuta la solicitud, si el objeto solicitado esta actualizado desde la última fecha de actualización.
if-none-match	ETag	Ejecuta la solicitud, si Etag del objeto solicitado es diferente.

La cabecera if-none-match contiene la etiqueta de entidad del objeto almacenado. Como respuesta a un requerimiento con esta cabecera, el servidor retornará el objeto sólo si la etiqueta de entidad del objeto sobre el servidor no coincide con la etiqueta de la cabecera if-none-match del requerimiento. De cualquier otra forma el servidor retornará un código de estatus "304 Not Modified".

Los clientes utilizan estas cabeceras para chequear si sus copias de objetos almacenados son aún validas y si no, para descargar los objetos actuales en un servidor de acceso. Aunque la cabecera if-none-match utiliza etiquetas de entidad proporcionan una manera más directa para comparar objetos, la cabecera if-modified-since es mucho más común porque esta fue introducida versiones iniciales de HTTP.

b. Cabeceras condicionales utilizadas para replicación

Los requerimientos condicionales también pueden ser utilizados en replicación si los servidores emplean HTTP para propagar nuevas versiones de objetos entre ellos.

Se debe recordar que HTTP proporciona varios métodos para actualizar la información sobre un servidor: POST, PUT y DELETE. Un servidor puede utilizar POST para enviar objetos actualizados a otro servidor con un objeto replica, PUT para enviar la nueva versión de un objeto en su totalidad, y DELETE para borrar objetos replicas

cuando el objeto mismo ha sido removido.

En cualquiera de estos métodos de propagación de actualización, es importante detectar conflictos en las versiones de los objetos. Por ejemplo, dos replicas de un objeto pueden ser actualizados independientemente sobre dos servidores.

Si uno de los servidores intenta hacer PUT de su copia del objeto sobre el otro servidor, hacerlo ciegamente puede ocasionar la perdida de las actualizaciones realizadas sobre el otro servidor.

En lugar de sobrescribir una copia que puede causar conflicto, el sistema debería llamar la atención sobre la excepción y permitir al administrador del sistema reconciliar las replicas. Similarmente, si un objeto ha sido borrado sobre un servidor y actualizado independientemente sobre otro, un intento del primer servidor de borrar el objeto en el segundo servidor podría no tener éxito.

Dos cabeceras condicionales pueden ser utilizadas en conjunto con los requerimientos de actualización para asegurarse que la actualización es desarrollada sobre una versión adecuada del objeto; ellos son mostrados en la Tabla 1.2.

Tabla 1.2 Cabeceras condicionales usadas en replicación

Cabecera	Valor	Significado
if-unmodified-since	date	Ejecuta la solicitud, si la fecha de última modificación del objeto coincide con la fecha proporcionada
if-match	ETag	Ejecuta la solicitud, si Etag del objeto coincide con el actual ETag del servidor.

La cabecera if-unmodified-since es la inversa de la condición if-modified-since. Cuando recibe un requerimiento con esta cabecera, un servidor ejecuta el requerimiento solo si la fecha de última modificación del objeto es la misma que la fecha proporcionada en la cabecera if-unmodified-since del requerimiento.

La cabecera if-match especifica lo inverso de la condición if-none-match. Un servidor ejecuta el requerimiento con esta cabecera sólo si la etiqueta de entidad proporcionada en la cabecera if-match del requerimiento coincide con el actual ETag del objeto en el servidor. Si la condición descrita en ambas cabeceras no se presenta, el servidor no ejecuta el requerimiento y retorna un código de estatus "412 Precondition Failed"

1.6.2 Edad y tiempo de expiración de los objetos almacenados

HTTP también proporciona soporte a los caches para decidir cuándo comprobar la validez de los objetos almacenados (cacheados). Este soporte está basado en el concepto de TTL (Time to live – tiempo de vida), el cual es similar al concepto con el mismo nombre que se encontró en el almacenamiento de DNS. El tiempo de vida (TTL) de un objeto es el tiempo durante el cual el objeto es considerado valido en un cache.

Un objeto HTTP que ha sido almacenado más tiempo que su TTL se dice que ha

expirado. Cuando llega un requerimiento HTTP para un objeto expirado, el cache debe validar el objeto enviando un requerimiento condicional al servidor antes de entregar el objeto desde el cache.

Un concepto relacionado que es muy importante en HTTP 1.1 es la "edad" de un objeto almacenado, el cual es el tiempo desde que el objeto fue entregado o fue validado por última vez por el servidor origen. La Tabla 1.3 lista las cabeceras de las respuestas utilizadas para limitar el tiempo antes que un cache deba validar los objetos almacenados.

Un servidor HTTP puede proporcionar un valor de tiempo de vida (TTL) explícito para el objeto utilizando las cabeceras expires y max-age La cabecera expire proporciona la fecha (con la granularidad de un segundo) hasta la cual el objeto puede ser considerado como válido. Mas que ser una cabecera, max-age es en realidad una directiva en una cabecera compleja caché-control. Esta directiva establece la edad máxima de un objeto en caché antes de que este requiera la validación.

La cabecera expires proporciona el valor absoluto del TTL, mientras que la directiva max-age especifica el TTL relativo al tiempo en que el objeto fue entregado o validado por última vez por el servidor origen. Históricamente, expires es la cabecera más antigua, procedente de HTTP 1.0, si ambas cabeceras están presentes, la directiva máx-age tiene prioridad.

Tabla 1.3 Cabeceras de las respuestas utilizadas para limitar el tiempo

Cabecera	Valor	Significado
expires	Fecha	Fecha en el que expira el objeto
max-age	Segundos	Máxima edad del objeto para que sea validado
age	Segundos	Tiempo estimado desde que el objeto fue proporcionado (por el servidor) o desde la última validación.

Cuando un TTL relativo es utilizado para controlar la expiración de un objeto, un cache debe poder determinar la edad de un objeto. Esto puede ser difícil cuando el cache recibe el objeto a través de múltiples proxies.

Para habilitar la determinación de la edad, un proxy que obtiene el objeto directamente desde el servidor origen inserta una cabecera age en cualquiera de sus respuestas que incluye el objeto almacenado.

Esta cabecera, entonces, es actualizada por todos los proxies intermedios, basados en el valor de la edad de la cabecera cuando ellos obtienen el objeto y el tiempo que el objeto permaneció en su cache. Así, cualquier cache en el camino puede estimar la edad del objeto.

Desafortunadamente, los servidores a menudo fallan en proporcionar un TTL explicito

para los objetos que ellos proporcionan. Sin un TTL explicito, los caches recurren a la heurística para decidir cuándo validar sus objetos.

1.6.3 Redirección de requerimientos

Otra característica importante para almacenamiento (caching) y replicación son sus mecanismos para redireccionar un requerimiento desde un servidor a otro. Un servidor puede redireccionar un requerimiento a un proxy cache o a un servidor espejo (mirror), o puede notificar al cliente el conjunto de proxies y mirrors que pueden atender el requerimiento.

Los servidores HTTP implementan la redirección de requerimientos retornando una respuesta con un código especial de estado de redirección. Los principales códigos de estatus de redirección se muestran en la Tabla 1.4.

Un servidor puede utilizar una respuesta con el código de estatus "300 Multiple Choices" para proveer al cliente una lista de las locaciones que pueden atender el requerimiento. Las locaciones y sus características son especificadas en la porción de entidad de la respuesta.

Tabla 1.4 Principales códigos de estatus de redirección

Código de estatus	Como es proporcionada la información de redirección
300 Multiple Choices	cuerpo de la entidad
301 Moved Permanently	cabecera location
302 Found	cabecera location
303 See Other	cabecera location
305 Use Proxy	cabecera location
307 Temporary Redirect	cabecera location

El código de estatus "301 Moved Permanently" indica que el objeto solicitado ha sido permanentemente movido a otra ubicación. La respuesta proporciona la nueva URL en la cabecera location. El cliente debería utilizar la nueva URL en el futuro.

El código de estatus "307 Temporary Redirect" también proporciona al cliente una URL diferente para el requerimiento, sin embargo, este código de estatus indica que el objeto se ha movido sólo temporalmente, de esta manera el cliente debería utilizar el URL original en el futuro.

El código de estatus "303 See Other" indica que la respuesta al requerimiento debería ser entregada utilizando el URL proporcionado en la cabecera location y utilizando el método GET. Por ejemplo, una aplicación Web que normalmente es accedida utilizando el método POST puede utilizar este código de estatus para redireccionar al cliente a un objeto estático.

La respuesta "305 Use Proxy" redirecciona al cliente a un proxy. La cabecera de respuesta location proporciona la identidad del proxy. HTTP también proporciona un

código de estatus “302 Found” por razones históricas, el cual muchos navegadores, incorrectamente lo tratan de la misma manera como el código de estatus “303 See Other”, a pesar de la primera es, por definición, idéntica al código de estatus “307 Temporary Redirect”

1.6.4 Rango de requerimientos

Cuando un objeto descargado ha sido interrumpido, el cliente puede ya haber recibido una gran parte del objeto. El cliente podría entonces utilizar una cabecera de requerimiento “rango” para bajar solo las partes perdidas. La cabecera rango específica el intervalo de byte de la entidad que se solicita. En principio la cabecera puede contener varias cabeceras rango que especifican múltiples partes distintas de la entidad. Los requerimientos con cabeceras rango son llamados requerimientos rango.

Normalmente el servidor ejecuta el requerimiento con cabeceras rango, pero este puede retornar sólo la parte específica del objeto, en cuyo caso este utiliza el código de estatus “206 Partial Content” en lugar de “200 OK”.

Adicionalmente, si la descarga interrumpida incluye la cabecera Etag, el cliente también tiene la etiqueta entidad del objeto entero (ya que las cabeceras típicamente son transmitidas antes que el cuerpo).

Tener la etiqueta entidad le permite al cliente asegurarse que las partes perdidas que se solicitan en el requerimiento rango y la parte que el cliente ya tiene pertenecen a la misma instancia del objeto. Para este fin el cliente puede incluir una cabecera condicional if-range en su requerimiento rango. Cuando el servidor responde el requerimiento, responde con las partes requeridas del objeto solo si la etiqueta entidad proporcionada en la cabecera if-range coincide con el Etag actual del objeto. De otra manera, todo el objeto será proporcionado en la respuesta.

1.6.5 La cabecera cache-control

La cabecera cache-control puede contener múltiples directivas que controlan el uso de todos los caches que están situados entre el cliente que originalmente realiza el requerimiento y el servidor origen. Así, el valor de la cabecera cache-control es una lista de directivas, y cada directiva consiste de un keyword identificando la directiva y, opcionalmente, el valor de la directiva. Por ejemplo la siguiente es una cabecera cache-control valida conteniendo tres directivas:

“cache-control:max-age=120,no-transform,proxy-revalidation”

La cabecera cache-control puede estar presente tanto en el requerimiento como en la respuesta y es así considerado una cabecera general. Sin embargo, muchas directivas en esta cabecera o son especificas o son interpretadas de manera diferente en los requerimientos y respuestas.

a. Directivas de cabecera cache-control en los requerimientos

En HTTP 1.0, la cabecera pragma:no –cache en un requerimiento fuerza a recargar desde el servidor origen. Sin embargo, la funcionalidad proporcionada por esta cabecera no era suficiente, lo cual dio origen a la introducción de la cabecera cache-control en HTTP 1.1. La cabecera cache-control de un requerimiento HTTP puede contener las directivas listadas en la Tabla 1.5.

La directiva no-cache en un requerimiento fuerza a cualquier proxy intermediario a obtener un nueva copia del objeto desde el servidor origen. Esta directiva tiene la misma semántica que la cabecera pragm:no-cache de HTTP 1.0.

Tabla 1.5 Directivas cache-control de un requerimiento HTTP

Directiva	Valor	Significado
no-cache	ninguno	Objetos cache que no pueden ser usados para satisfacer la solicitud
no-store	ninguno	La respuesta a esta solicitud no puede ser almacenada en un cache
max-age	Segs.	Sólo los objetos cacheados más jóvenes pueden ser usados para satisfacer la solicitud
min-fresh	Segs.	Únicamente los objetos cacheados que no expirarán por un tiempo determinado pueden ser usados
max-stale	Segs.	Objetos cacheados que expiraron tiempo atrás específico, pueden ser usados
no-transform	ninguno	Únicamente la respuesta precisa proporcionada por el servidor de origen puede ser usada.
only-if-cached	ninguno	Un proxy no debería reenviar la solicitud en un cache perdido

La directiva no-store es más fuerte, y evita el almacenamiento accidental de la información en un cache. En una petición esta directiva prohíbe que el requerimiento o respuesta sean almacenadas en cualquier cache. La intención de esta directiva es prevenir actualizaciones accidentales o retención de información sensible (por ejemplo, sobre cintas de backup).

Esta directiva es especialmente útil para prevenir que los cache de los navegadores, los cuales frecuentemente almacenan información privada en los discos locales, almacenen información sensible.

Las directivas max-age y min-fresh permite a un cliente imponer requisitos más estrictos sobre un proxy que está procesando el requerimiento de un cliente. La directiva max-age indica que un cliente está dispuesto a aceptar una respuesta almacenada (cacheada) solo si el esta edad no excede el valor especificado en segundos; objetos más antiguos no pueden ser utilizados para cumplir el requerimiento aun si esos objetos no han expirado.

La directiva min-fresh específica que el cliente está dispuesto a aceptar una respuesta

desde un cache solo si la respuesta no espirara por al menos en el tiempo especificado como el argumento.

Por lo contrario, la directiva max-stale relaja la consistencia del política normalmente utilizada por un proxy. Esta directiva indica la voluntad del cliente de aceptar una respuesta expirada, siempre que esta respuesta no ha expirada mas allá de un número determinado de segundos atrás.

La directiva no-transform previene a un proxy intermediario de almacenar la representación del objeto. Por ejemplo, algunos prosees que sirven como dispositivos wireless degradan la calidad de los gráficos para reducir su tamaño. Esta directiva previene este tipo de degradación

Finalmente, el cliente utiliza la directiva only-if-cached para indicar que el requerimiento puede ser ejecutado solo si el primer proxy que recibió el requerimiento tiene el objeto solicitado en su cache. Esta directiva es útil, por ejemplo, si el ancho de banda entre el proxy y el resto de interne es limitado, utilizando esta directiva, los clientes no descargaran información no critica a menos que la información ya esta almacenada en el proxy.

b. Directivas de cabecera cache-control en las respuestas

Una variedad de directivas cache-control pueden ser utilizadas en una respuesta. Esas directivas son mostradas en la Tabla 1.6. Dos de ellas, no-store y no-transform, son idénticas a las mismas directivas en un requerimiento. El resto de las directivas son específicas a las respuestas o tienen diferente significado de las directivas similares utilizadas en un requerimiento.

La directiva no-cache previene el almacenamiento de la respuesta. La directiva prívate permite el almacenamiento (caching) pero indica que la respuesta es direccionada sólo para el cliente que lo pidió. Esta directiva impide que otro cliente almacene la información. La directiva public indica que la respuesta puede ser almacenada por cualquier cache y compartida por diferentes clientes.

La directiva must-revalidate permite a un servidor a especificar que un cache debe siempre validar el objeto almacenado antes de utilizarlo; la validación se realiza utilizando requerimientos condicionales al servidor origen. Esta directiva garantiza que los usuarios tendrán la versión actualizada del objeto.

Esta garantía tiene como consecuencia tener que acceder al servidor origen para cada requerimiento. La directiva proxy-revalidation es similar, excepto que los caches simples de los clientes, como los encontrados en los web browser, no tienen que desarrollar la revalidación.

Si la directiva max-age está presente en una respuesta, el objeto se puede guardar

(es cacheable) y se considera que expira una vez que su edad excede el número de segundos especificados por la directiva max-age. La directiva s-maxage es idéntica a la directiva max-age, excepto que esta se aplica sólo si la respuesta esta almacenada en un cache compartido. En este caso esta directiva sobrescribe la directiva max-age.

Tabla 1.6 Directivas cache-control usadas en una respuesta HTTP

Directiva	Valor	Significado
no-cache	ninguno	La respuesta no puede ser cacheada
no-store	ninguno	La respuesta no puede ser almacenada en ningún cliente (Proxy o navegador)
private	ninguno	La respuesta puede ser usada únicamente por el cliente que originalmente lo solicitó
public	ninguno	La respuesta debe ser cacheada y compartida a diversos clientes.
must-revalidate	ninguno	El cache del objeto debe sr siempre válidado antes de ser usado
proxy-revalidation	ninguno	Lo mismo que must-revalidate pero aplica a cache proxy solamente
max-age	Segs.	Un cache debe validar el objeto antes de servirlo a cualquier cliente una vez que el objeto alcanza el valor de edad especificado
s-maxage	ninguno	Lo mismo que max-age pero aplica sólo a proxies
no-transform	ninguno	Únicamente la respuesta precisa tal cómo fue proporcionada por el servidor de origen puede ser usada

c. Ejemplo de la cabecera cache-control

Para ilustrar alguno de los usos de la cabecera cache-control, se considerará un sitio Web hipotético, bigmoney.com. Este sitio tiene dos objetos Web que quiere entregar. El primer objeto es una página estática de bienvenida, welcome.html; el segundo objeto, mysecret.html, es generado dinámicamente por cada usuario individual. Dos usuarios, Bill y Bob, se conectan a bigmoney.com a través de un proxy cache compartido. La cabecera cache-control puede controlar el cache del navegador en los navegadores web de Bill y Bob y el cache compartido sobre el proxy.

Se considera las decisiones de almacenamiento (caching) realizadas por todas las partes involucradas. Durante la configuración de bigmoney.com, el diseñador web debe decidir que objetos pueden ser almacenados, donde y por cuánto tiempo. El objeto wellcome.html es la misma página para todos los usuarios y puede ser almacenada por el proxy compartido o por el navegador privado de cada usuario.

Ya que el caching compartido está permitido por default en HTTP 1.1, no se necesita especificar ninguna cabecera de respuesta para esta página. Si el diseñador web quiere ser extremadamente cuidadoso y evitar la posibilidad de que otra cabecera HTTP sobrescriba esta página default y prevenir que welcome.html sea almacenada en cache,

el diseñador puede explícitamente especificar el objeto como público utilizando la cabecera cache-control.

Por otro lado, la pagina mysecrets.html generada dinámicamente es destinada solamente a un simple usuario. Así, el diseñador web puede tener que incluir la directiva private en la cabecera cache-control del objeto. Sin embargo, esta directiva permite aun que tanto el navegador como el proxy almacenen en cache el objeto. Además, la página puede ser almacenada sin querer sobre el disco o backupeada en cache. Asumiendo que la página contiene información confidencial que debería mantenerse en secreto, el diseñador web debe agregar la directiva no-store que previene que los objetos sean almacenados en cache o almacenados sin querer por otro componente.

Después de decidir que objetos pueden ser almacenados en cache, el diseñador web debe determinar cuánto tiempo pueden permanecer en cache. El único objeto cacheable en nuestro ejemplo es la página welcome.html. Asumiendo que esta página contiene descuentos promocionales que cambian cada hora.

El diseñador web debe especificar que la directiva max-age debe ser configurada para permanecer el tiempo de la promoción. Además, suponiendo que bigmoney.com decide que ellos pueden agregar promociones extras para usuarios nuevos que acceden por primera vez al sitio web. Ya que estas promociones pueden ser en intervalos aleatorios, el diseñador web decide que los caches compartidos siempre deberían validar la página welcome.html antes de entregarla desde el cache. Esta validación es forzada especificando la directiva proxy-revalidation en la respuesta.

En este punto el sitio web bigmoney.com está listo. Los usuarios típicamente confían en su navegador para configurar correctamente la directiva cache-control. Sin embargo, para los propósitos de este ejemplo, asumiendo que el usuario configura personalmente esa directiva.

Suponiendo que Bill se conecta a la Internet vía un gran ancho de banda. Él especifica la directiva no-transform sobre el requerimiento, para asegurarse que el tendrá la pagina con la mejor calidad. Tampoco le preocupa el ancho de banda consumido asé es que quiere recibir el contenido mas actualizado. El elige aceptar solo el contenido que tiene menos de un minuto de antigüedad y para esto configura la cabecera max-age a un minuto.

Como resultado, un cache utilizara su copia almacenada de la página welcome.html para el requerimiento de Bill solo si la pagina fue recibida desde el servidor origen hace menos de un minuto. Este comportamiento se aplica así el cache considere o no que la copia de su página es válida.

Por otro lado, Bob, utiliza una palm con wireless para accede a la Internet. El no

puede gastar mucho ancho de banda. Así, el no específica la directiva no-transform, esperando que el proxy cache compartido reduzca el tamaño del objeto lo más que sea posible. Él también especifica una directiva max-stale de un día, reduciendo el ancho de banda requerido para actualizar objetos pasados pero incrementando la probabilidad de ver contenido desactualizado.

Tabla 1.7 Cabeceras cache-control en el hipotético Web Site bigmoney.com

Mensaje HTTP	Cabecera cache-control
Petición de Bill	cache-control: no-transform,max-age=60
Petición de Bob	cache-control:max-stale=86400
Respuestas de bigmoney.com conteniendo welcome.html	cache-control:public,max-age=t, proxy-revalidation
Respuestyas de bigmoney.com conteniendo mysecrets.html	cache-control:no-store

La Tabla 1.7 resume las cabeceras cache-control en los requerimientos realizados por los dos usuarios y en la respuesta de bigmoney.com (t en segundos). Este pequeño ejemplo nos da una muestra de la complejidad de directivas HTTP y sus interacciones.

1.6.6 Cookies

HTTP permite a los cliente Web y a los servidores a introducir nuevas cabeceras que no están especificadas en el estándar, siempre que estas cabeceras sigan la estructura general "keyword - value".

HTTP estipula que los clientes y servidores que no comprenden las nuevas cabeceras deben ignorarlas. Esta provisión para nuevas cabeceras ha sido un gran facilitador para la innovación, permitiendo le introducción de nuevas funcionalidades para HTTP. El mecanismo cookie, el cual no es parte del estándar HTTP, es un gran ejemplo de esta innovación. Este proporciona un mecanismo elegante para almacenar estados dentro de las interacciones web mientras se mantiene la naturaleza "stateless" de los servidores HTTP. Por ejemplo, un cookie puede registrar el estado de una compra sobre un sitio de comercio electrónico o un perfil de usuario que almacena sus preferencias tal como el lenguaje a utilizar.

Si el servidor necesita guardar algún estado para una interacción futura, este puede retornar su estado al cliente en una cabecera de respuesta set-cookies. El cliente nunca mira su estado (llamado cookie) simplemente lo almacena localmente. La próxima vez que el cliente envíe un requerimiento a este servidor, incluirá una cabecera de requerimiento "cookie" conteniendo su estado. De esta manera el servidor tendrá el estado deseado sin tener que mantenerlo.

Los atributos especificados en la cabecera set-cookies incluyen (entre otros campos) un nombre de cookie, un valor de cookie (el estado que se está guardando), un valor de timeout, una ruta, y un dominio. El nombre de cookie y el valor serán retornados en los

requerimientos HTTP posteriores en una cabecera de requerimiento "cookie" si el URL solicitado coincide con la ruta y el dominio especificado en la cabecera set-cookie. El timeout es utilizado para limitar el tiempo durante el cual el cliente debe almacenar el cookie. La posibilidad para configurar un cookie selectivamente para URLs que coinciden con un determinado patrón proporciona mucha flexibilidad al sitio web.

Por ejemplo, el sitio bigmoney.com puede incluir una cabecera "set-cookie:name=profile,domain=www.bigmoney.com, path=mysecrets.html, cookie=day-trader" con la respuesta a Bill. Entonces, cualquier requerimiento futuro desde el navegador de Bill para mysecrets.html incluirá una cabecera "cookie:name=profile, value=day-trader", el cual el servidor puede utilizar para personalizar la página mysecrets.html. Los requerimientos de Bills para welcome.htmll, sin embargo no incluirán un cookie porque no es necesario por ser una página genérica. Es más, los requerimientos para cualquier objeto embebido dentro de msecrets.html no incluirá un cookie.

Los atributos ruta y el dominio de un cookie también permiten diferentes aplicaciones sobre el mismo servidor para almacenar diferentes cookies sobre un mismo cliente. Por ejemplo, si el sitio firm-x.com contiene las aplicaciones auction y stock-quote que utilizan los dominios auction.firm-x.com y quote.firm-x.com respectivamente, ambas aplicaciones pueden mantener cookies independientes especificando su respectivo dominio en su cabecera de respuesta set-cookies.

Además de permitir que un servidor stateless almacene el estado sobre los clientes, otro uso típico de los cookies es poder registrar la identidad del usuario. En este caso el servidor mantiene el estado del usuario, y este emplea un cookie para evitar pedir al usuario su identificación varias veces. Utilizando la dirección IP del navegador del usuario para identificar el usuario no es suficiente: el usuario puede utilizar diferentes navegadores en casa y en el trabajo, y navegadores dial-up pueden tener diferentes IPs cada vez que se conectan a Internet.

En el primer caso, el sitio web debería identificar el usuario almacenando el mismo cookie sobre todos los navegadores, y debe preguntar la identificación del usuario solo cuando un nuevo navegador es utilizado por primera vez. En el segundo caso, el navegador envía el mismo cookie que identifica al usuario sin importar la dirección IP del navegador.

Ya que los servidores utilizan los cookies para personalizar el contenido para un usuario en particular, el problema obvio con los cookies, en el contexto de caching, es que las respuestas con la cabecera set-cookies no pueden ser almacenadas en un cache compartido y los requerimientos con la cabecera cookie no pueden ser proporcionados

desde un cache compartido.

Sin embargo, HTTP 1.1 permite caching de respuestas conteniendo set-cookies incluyendo este mismo la cabecera set-cookie. Así, estrictamente hablando, un sitio web puede utilizar explícitamente la cabecera cache-control para prevenir el caching de esas respuestas. En la práctica, muchos sitios no lo hacen, y los proxies a menudo no almacenan esas respuestas como precaución.

1.6.7 Soporte para servidores compartidos

La línea de requerimientos HTTP contienen solo la parte de la ruta de la URL solicitada. Un servidor HTTP 0.9 no podría saber a cual dominio pertenece la URL. Lo que asumieron los diseñadores de HTTP 0.9 fue que un servidor siempre comparte el nombre de dominio con todas las URLs y por tanto enviar el nombre de dominio de la URL era redundante. Sin embargo, la aparición de compañías de web hosting ha invalidado esta asunción. Estas compañías despliegan servidores compartidos que almacenan sitios web con diferentes nombres de dominios, una práctica conocida como hosting virtual.

Diferentes sitios web a menudo tienen URLs con porciones de rutas comunes, tales como www.firm-x.com/home.html y www.firm-y.com/home.html. Cuando no hay información del nombre de dominio, las compañías de hosting utilizan múltiples direcciones IPs a un servidor compartido, una por sitio web hosteado. De esta manera el servidor puede determinar a qué sitio web pertenece la URL analizando el IP destino del requerimiento. Obviamente, esto es un gasto de las direcciones IPs.

Consecuentemente, HTTP 1.1 define una cabecera de requerimiento de host obligatoria que contiene el nombre del host y parte de la URL. Esta información, combinada con la ruta especificada en la línea de requerimiento, permite a un servidor determinar la URL completa, aun si los nombres de múltiples dominios son mapeados a una misma dirección IP. La cabecera de host es también utilizada en algunos despliegues de forward proxies y sustitutos cuando están compartidos con múltiples sitios web.

1.6.8 Identificadores de objetos expandido

Ya que los objetos son identificados por sus URLs, los caches normalmente utilizan URLs para identificar objetos almacenados. Sin embargo, los servidores algunas veces entregan diferentes contenidos para un determinado URL basado en el valor de ciertas cabeceras de requerimientos.

Ya se ha visto un ejemplo de este comportamiento, donde los servidores personalizan el contenido basado en los cookies. De una manera general, los servidores pueden utilizar otras cabeceras de requerimientos para personalizar el contenido o elegir entre diferentes representaciones de los objetos. De hecho, esas cabeceras forman parte del

objeto identifier. Para evitar servir representaciones de objetos erróneas, los proxies también deben utilizar los identificadores expandidos para los objetos que almacenan.

El servidor origen puede listar las cabeceras de requerimientos que deben ser utilizadas para identificar los objetos almacenados en la cabecera de respuesta vary. Estas cabeceras de requerimientos son llamadas cabeceras selecting. Un objeto almacenado en cache con un campo vary puede ser utilizado para un requerimiento solo si todas las cabeceras selecting en el requerimiento coinciden con las cabeceras correspondientes almacenadas desde el requerimiento original que coloco el objeto en el cache.

1.6.9 Aprendizaje de la cadena de proxy

Un cliente puede querer conocer la cadena de proxies que toma su requerimiento o su camino a los servidores origen. Un proxie en algunos casos afecta (o incluso interrumpe) el servicio recibido desde ciertos sitios web; es por tanto importante que un cliente sea consciente de la existencia del proxy, así el cliente puede utilizar cabeceras de requerimientos apropiadas para ir directamente al servidor origen en tales situaciones.

HTTP 1.1 proporciona una cabecera via y un método de requerimiento TRACE, los cuales juntos permite al cliente conocer si existe un proxy. La cabecera vía mantiene track de los proxies y versiones de los protocolos involucrados en transferir el requerimiento desde el cliente al servidor. Cada proxy que procesa el requerimiento debe agregar una entrada a la cabecera de requerimiento vía, incluyendo el nombre del proxy y el protocolo utilizado para recibir el requerimiento. Por ejemplo la Figura 1.5 muestra un navegador (browser) cuyo requerimiento atraviesa una cadena de dos proxies sobre su camino al servidor origen (server).

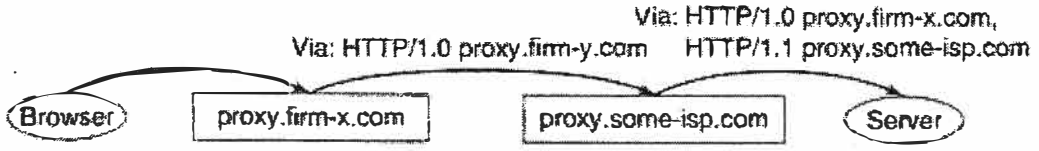

Figura 1.5 Ejemplo de cabeceras via

Si el navegador utilizar HTTP 1.0 para comunicarse al primer proxy y el segundo proxy utiliza HTTP 1.1 para comunicarse entre ellos, la Figura 1.5 muestra la cabecera de requerimiento vía en diferentes puntos sobre la ruta del requerimiento. El primer proxy inserta la cabecera via que especifica el nombre del proxy y HTTP 1.0, desde que es el protocolo por el cual el proxy recibe el requerimiento. El segundo proxy agrega una entrada a la cabecera que contiene su nombre y el protocolo utilizado para recibir el requerimiento (HTTP1.1).

El método de requerimiento TRACE estipula que el recipiente final de este requerimiento lo envíe de regreso al cliente en la entidad body de la respuesta. Las

respuestas al requerimiento TRACE nunca son almacenadas en cache, y por tanto el recipiente final típicamente es el servidor origen al cual es enviado el requerimiento. Ya que cada proxy sobre el camino al servidor inserta una cabecera vía, el requerimiento acumula la lista entera de proxies entre el cliente y el servidor. Este es el estado final del requerimiento que el servidor regresa al cliente, permitiendo que el cliente conozca la cadena de proxy.

El cliente puede limitar el número de proxies que el requerimiento TRACE atravesará incluyendo una cabecera max-forwards en el requerimiento. Esta cabecera contiene un entero que el cliente inicialmente configura con el máximo número de proxies permitidos.

Cada proxy decrementa este entero antes de enviarlo hacia adelante. Sin embargo, el proxy que recibe un requerimiento con un valor de max-forwards de 0no enviará el requerimiento; en su lugar, responderá el requerimiento el mismo como si fuera el destino final del requerimiento. Así, el cliente aprende solo la parte inicial de la cadena de proxy al servidor origen si este canal excede el valor original de la cabecera max-forwards.

La cabecera vía tiene muchos otros usos. Esto incluye las versiones de HTTP utilizadas por proxies intermedios. El conjunto de versiones de protocolos involucrados en la transferencia será importante si el servidor tiene la intención de utilizar funcionalidades específicas a HTTP 1.1, las cuales pueden ser deducidas de la cabecera vía del requerimiento.

La cabecera via también permite la detección de loop en el ruteo de los requerimientos, previniendo, de esta manera, una situación donde un requerimiento viaja indefinidamente a través de un loop de proxies.

1.6.10 Cacheabilidad de contenido web

Una respuesta es llamada “cacheable” (almacenada en cache) o “ uncacheable” (no cacheable) dependiendo si esta puede o no ser almacenada en un cache y utilizada para satisfacer requerimientos futuros.

Similarmente, se llama a un requerimiento cacheable si este puede ser atendido por una respuesta cacheada, y no cacheable si no es así. En particular, un requerimiento a un objeto uncacheable es siempre uncacheable.

Con algunas excepciones, HTTP 1.1 proporciona un regla simple para decidir si un requerimiento o respuesta es cacheable o no: el protocolo permite a cualquier respuesta a ser almacenada en cache y a cualquier requerimiento a ser atendido desde una respuesta almacenada en cache, a menos que este expresamente prohibido por las cabeceras HTTP. En la práctica, sin embargo, los requerimientos y respuestas no siempre incluyen esas cabeceras, y los proxies y otros caches utilizan la heurística, además de las cabeceras HTTP, para identificar el contenido que no debería ser

almacenado en cache.

Las siguientes son las reglas principales y heurísticas que los proxies utilizan para decidir la cacheabilidad de los requerimientos y respuestas:

a) Muchos proxies consideran que los requerimientos son no cacheables si las URLs requeridas contienen la cadena "cgi-bin". Esta heurística está basada en el hecho de que las URLs de esta forma usualmente identifican los objetos generados dinámicamente. Los objetos dinámicos a menudo son específicos a un requerimiento particular, o los requerimientos para esos objetos tienen efectos secundarios más allá de generar una respuesta. En este caso, es esencial que el requerimiento llegue al servidor origen. En otros casos, sin embargo – por ejemplo, búsquedas a un diccionario o un mapa – puede ser perfectamente seguro almacenar en cache respuestas dinámicas, y esta heurística puede ser demasiado restrictiva. Además, no todos los objetos dinámicos se pueden identificar por esta heurística, como se verá más adelante.

b) Los requerimientos con una cabecera cookie y las respuestas con una cabecera set-cookie son típicamente consideradas no cacheables por proxies. Esto es, otra vez, sólo heurística. Su razonamiento es que las cookies se utilizan a menudo para personalizar un objeto web para un cliente en particular o incluso a una solicitud en particular. Así, el objeto enviado en respuesta a un requerimiento puede no ser reutilizado para otro requerimiento.

c) Los proxies usualmente no almacenan en cache los requerimientos que utilizan métodos diferentes a GET o HEAD. De hecho, HTTP 1.1 proporciona reglas elaboradas que especifican cuando un requerimiento que utiliza otros métodos puede ser cacheable, pero tales situaciones son infrecuentes que implementar esas reglas no vale la pena.

d) Los proxies usualmente consideran los requerimientos con ciertas cabeceras, tal como la cabecera authorization, y respuestas con ciertos códigos de respuestas, tal como "307 Temporary redirect" a ser uncacheable. Otra vez, HTTP 1.1 especifica ciertas condiciones bajo las cuales esos requerimientos y respuestas podrían aun ser almacenadas en cache, pero en general se puede asumir que son no cacheables.

El contenido uncacheable o no cacheable (no se puede realizar cache) reduce extremadamente los beneficios de Web Caching (Almacenamiento en cache).

1.7 Reglas de comportamiento web para el caching

Cualquier estudio serio de las técnicas para Web Caching y replicación debe empezar con algún entendimiento de las propiedades y usos de los objetos almacenados en cache. ¿Cuán grande son los objetos? ¿Con que frecuencia serán reutilizados? ¿Con que frecuencia cambiaran? Todos esos temas tienen un profundo efecto sobre las políticas y sobre el diseño de los caches y mirrors. [9][10][11]

Desafortunadamente, es muy difícil caracterizar la Web. En primer lugar, estudiar la Web es como tratar de perseguir un objetivo en movimiento. Los resultados a menudo son obsoletos por el tiempo en que son publicados. Esto hace del comportamiento de la Web (o, en un término mas científico, caracterización) una gran área para escribir artículos, pero a la vez frustrante en términos de resultados duraderos. Segundo, porque debido al gran tamaño de la Web, es difícil obtener muestras significativas. Esas dificultades han originado muchos informes contradictorios sobre características de la Web. Aquí algunos ejemplos:

a) "Nosotros observamos que los documentos que son remotamente (que son requerimientos de clientes remotos) populares y globalmente populares son actualizados con muy poca frecuencia"

b) "Considerando todos los tipos de data, el acceso a los recursos con mayor frecuencia son claramente más actualizados que los recursos accedidos con menos frecuencia"

c) "No hay una evidencia sólida de que los documentos populares cambien con menos frecuencia. En el diseño de cualquier esquema consistente de Web cache o cualquier modelo de acceso Web, tal vez sea mejor asumir que no existe una correlación entre la popularidad de los documentos y su frecuencia de cambio "

Esas afirmaciones cubren todas las posibles respuestas a la pregunta de la relación entre la popularidad de los objetos y la tasa de modificación. Algunas de esas diferencias pueden ser debido a las diferencias en los ambientes estudiados: Algunos estudios están en relación a un único Servidor Web de una universidad, mientras otros estudios examinan el log de acceso de un servidor proxy que procesa todos los requerimientos desde un grupo de clientes. Sin embargo, ambos utilizan los logs de acceso del proxy y aún así llegan a diferentes conclusiones. En cualquier caso esas diferencias demuestran la dificultad de organizar un experimento representativo y obtener conclusiones significativas a nivel mundial.

Esta sección resume el poco conocimiento sobre el comportamiento de Internet que es aceptado como importante y más o menos estable, con lo cual nos referimos como "reglas de oro" del comportamiento Web. Se prestará especial atención a aspectos que están más relacionados a caching y replicación.

1.7.1 Tamaño del objeto

La primera propiedad Web de importancia es el típico tamaño de los objetos. Los objetos pueden tener tamaños drásticamente diferentes. Imágenes de iconos, banners de publicidad, y objetos de texto que son típicamente muy pequeños, cerca de 1 Kbyte.

Como ejemplos de objetos grandes se tiene a los archivos de audio y video, los cuales puedes fácilmente alcanzar múltiples megabytes. Conocer el tamaño de los

objetos es útil para muchas decisiones de diseño, por ejemplo, para decidir sobre la capacidad de almacenamiento de un cache o el ancho de banda de una conexión.

Hay una diferencia entre el tamaño de los objetos que existen en la Web y el tamaño de los objetos transferidos desde el servidor a los clientes.

1. El primero refleja el tamaño de los diferentes objetos, independientemente del número de veces que el objeto es descargado.
2. El segundo cuenta cada transferencia, aun si el mismo objeto es descargado varias veces.

Los estudios sobre el tamaño de los objetos son generalmente vagos sobre cuál de esas dos características es la que miden.

En general los siguientes valores parecen ser los supuestos más seguros, tanto para los tamaños de los diferentes objetos y el tamaño de los objetos descargados:

a) El promedio del tamaño de los objetos recuperados por los clientes es del orden de 10 a 15 KBytes.

b) La media del tamaño de los objetos es mucho menor que el promedio, alrededor de 2 a 4 KBytes.

Estos números muestran que la mayoría de los accesos son a objetos pequeños. De hecho, se informa que el 75 % de todos los objetos retribuidos son significativamente más pequeños que el tamaño medio. Esto concluye que los objetos pequeños son accedidos con mucha más frecuencia.

Por un lado, hay solo mucho más objetos pequeños en la Web. Sin embargo, el hecho de que el tamaño promedio y la media de las descargas son más pequeños que los distintos objetos indica que los documentos pequeños son también desproporcionalmente más populares que algunos grandes, aunque este sesgo no es muy pronunciado.

Por otro lado, el hecho que unos pocos objetos grandes pueden afectar significativamente el tamaño promedio significa que los pocos objetos grandes son realmente grandes. De hecho, se pueden encontrar objetos de decenas de megabytes.

Un gran número de objetos muy pequeños podría ser explicado por la proliferación de iconos pequeños e imágenes de publicidad. Los objetos grandes típicamente contienen PostScript y archivos PDF, imágenes grandes, o especialmente, archivos de video, audio o paquetes de distribución de software. Como se verá en la siguiente sección, muy pocos de estos objetos son visitados. Sin embargo, se puede esperar que el número de peticiones de objetos grandes se incremente en el futuro a medida que más objetos multimedia sean publicados en la web y que mejore el ancho de banda haciendo que las descargas de estos objetos sean menos dolorosas.

La concentración muy desigual de los accesos a los objetos pequeños crea un fenómeno interesante para las políticas de caching. Un cache puede incrementar su hit rate favoreciendo a los objetos pequeños (por ejemplo, no reteniendo objetos grandes en cache). Sin embargo los beneficios de un cache hit son más grandes cuando el objeto almacenado en cache es grande. Por lo tanto los beneficios del cache global pueden superiores en un cache con un hit rate bajo.

1.7.2 Tipos de objeto y cacheabilidad

Antiguamente (hace sólo unos pocos años) sólo habían unos pocos tipos de objetos Web: HTML estáticos, imágenes y documentos generados dinámicamente por script CGI. Hoy, se puede acceder a toda una gama de tipos de objetos a través de un navegador Web estándar.

En la actualidad, una gran mayoría de las descargas, en términos de objetos accedidos y bytes descargados, incluye paginas HTML e imágenes, con las imágenes superando a las paginas HTML. Juntos son cerca de 90% de todos los accesos Web y, típicamente, el 70% de bytes descargados, aunque el número de bytes varía de 60% a 80%. en los servidores individuales.

Esos números podrían variar grandemente, sin embargo, el interés de este trabajo están en las estadísticas que reflejan los Web típicos utilizados por los clientes. Esta perspectiva es mejor capturada por proxy traces, ya que ellos reflejan los accesos de los clientes a todos los servidores. Todos los estudios citados aquí utilizan proxy trace en sus análisis).

Los objetos multimedia son accedidos muy pocas veces pero representan una buena fracción de bytes descargados: 14% fueron observados por Wolman, 24 % por Arlitt y hasta 6% por Douglis.

Otra observación interesante es que el número de accesos a los objetos generados dinámicamente es más bien pequeño, con valores reportados desde menos de 1% a 10 a 20%. Este pequeño número de objetos web dinámicos contradice la intuición sobre la naturaleza dinámica del contenido Web. Sin embargo, debe recordarse que un documento web típicamente consiste de una página contenedora identificada por el documento URL y un número de objetos embebidos dentro de la página contenedora.

En el caso de documentos dinámicos, usualmente solo la página contenedora es dinámica, mientras que los objetos embebidos son estáticos (típicamente imágenes). Un documento contiene un promedio de 10 imágenes embebidas. Así el porcentaje de accesos a documentos dinámicos, entre todos los accesos a documentos, puede ser tanto como 10 veces mayor que la de los accesos a los objetos dinámicos entre todos los accesos a objetos.

Un tema de particular importancia para caching es qué con frecuencia se producen peticiones que pueden utilizar un cache. Como se discutió anteriormente, un requerimiento puede ser no cacheable ya sea por ciertas características en el requerimiento mismo, o porque el requerimiento es por un objeto no cacheable. La Tabla 1.8 resume las principales características de un requerimientos o requerimiento de objetos que lo hacen no cacheable bajo las reglas de HTTP o heurísticos más utilizados por los proxies.

El factor más importante que afecta el almacenamiento en cache es la presencia de un cookie. Se encontró que los cookies están presentes en 30% de los requerimientos desde usuarios residenciales y en cerca de 20% desde una comunidad de científicos de la computación. Se observó también que los sitios Web comerciales utilizan cookies cerca de cuatro veces más que los sitios educativos y los sitios americanos lo utilizan cerca de tres veces más que los sitios alemanes. Un uso amplio de cookies es a veces una herramienta de diseño descuidada de un sitio web, con los desarrolladores Web utilizando cookies sin darse cuenta del efecto en caching.

Tabla 1.8 Características de requerimientos de objetos que lo hacen uncacheable

Característica no cacheable	Frecuencia de ocurrencia (%)
Cabeceras cache-control/pragma	9–15
Cabeceras cookie/set-cookie	19–30
Cabecera authentication	1–1.7
No es un método GET o HEAD	1.4–2
"cgi-bin" o "?" en un URL	1–20
Código de respuesta no cacheable	22.8
Total de peticiones no cacheables	37–43

En conjunto, alrededor del 40% de todos los requerimientos son no cacheables. Esto representa claramente un gran obstáculo para caching. Es necesario una mejor tecnología (así como un mayor nivel de conciencia de los desarrolladores web en temas de cache) para abordar este problema.

1.7.3 Popularidad del objeto

La Web muestra la popularidad desigual de los objetos: Muchos accesos son para, relativamente, un pequeño conjunto de objetos, mientras la gran mayoría de objetos casi nunca son accedidos. Se analizaron seis diferentes sitios Web y se encontraron que 10% de los objetos representaban un 80% a 95% de todos los requerimientos recibidos para cada sitio web.

Muchos estudios (Glassman 1994, Ameida 1996, Chnha 1996, Barford 199) observaron que la popularidad de los objetos se ajusta a la distribución Zipf-like (Zipf 1949). Aplicado a la Web, el estado de distribución Zipf-like establece que la popularidad

(que es, la frecuencia de accesos) del i-ésimo objeto más popular es proporcional a 1/ia, para algunos una constante entre 0 y 1. En particular, para a=1, el objeto Web más popular es dos veces tan popular como el segundo objeto más popular. Esta distribución de popularidad de los objetos es conocida como ley Zipf.

La popularidad de todo sitio Web es tan desigual como la popularidad de los objetos individuales. Los traces de los requerimientos a muchos proxies muestran que el 25% de los sitios Web son responsables por el 90% de todos los accesos y del 90% de bytes descargados (Abdulla 1998). Williams (1996) encontró que la distribución de la popularidad de los sitios se ajusta a la distribución Zipf-like.

La actual concentración de sitios y objetos populares es probablemente aún más alta, ya que los accesos repetidos al mismo recurso desde el mismo usuario son a menudo entregados desde el cache del navegador del usuario y no son contabilizados en los estudios mencionados aquí. Una alta concentración de accesos, como explicaremos en la siguiente sección, es una buena noticia para caching y replicación.

1.7.4 Localidad de referencia

La distribución desigual es una manifestación de la localidad de referencia. La Localidad de referencia caracteriza la posibilidad para predecir los futuros accesos a los objetos desde accesos pasados. Hay dos tipos principales de localidad: 1) temporal y 2) espacial.

La localidad temporal se refiere a accesos repetidos al mismo objeto dentro de cortos periodos de tiempo. Una localidad temporal alta implica que los objetos accedidos recientemente serán probablemente accedidos otra vez en el futuro.

La localidad espacial se refiere a patrones de acceso donde los accesos a algunos objetos frecuentemente implican accesos a otros objetos. Por ejemplo, un acceso a una página HTML puede implicar accesos a objetos embebidos o a hiperenlaces ubicados en esta página. Una localidad espacial alta implica que las referencias a algunos objetos pueden ser un predictor de futuras referencias a otros objetos.

La localidad de referencia es aprovechada para formular políticas de caching, tal como decidir cuales objetos mantener o remover desde el cache y cuales objetos son prefetch, esto es, cargados dentro del cache antes que ellos sean accedidos. Aunque no se han publicado suficientes resultados sobre localidad de referencia para agregarlos a las reglas empíricas, esto es útil para comprender las metodologías para caracterización (que es, definiendo una medida de) de este aspecto importante del comportamiento Web. Estas metodologías son el centro del resto de esta sección.

a. Localidad Temporal

Una herramienta importante en la caracterización de la localidad temporal es el

concepto de stack distance. Esto se puede explicar al considerar un stack de profundidad infinita, con elementos correspondientes a objetos. Cuando un objeto es requerido, este es movido a la parte superior del stack, dejando su antigua posición en el stack (si este es una referencia repetida) y colocando otros objetos bajo el stack. El stack distance del objeto de referencia es igual a la posición de este objeto en el stack en el momento del acceso. Cómo un ejemplo, la Figura 1.6 da un posible estado del stack antes y después de que el requerimiento al objeto X11 ocurra. El stack distance para este acceso es 4.

b. Localidad espacial

Es difícil definir una medida que pueda caracterizar la localidad espacial. Un método, descrito por Almeida (1996), observa que si la referencia a objetos puede predecir referencias a otros objetos, entonces la traza completa de referencias debe exhibir un gran número de secuencias repetidas de objetos referencias. Por ejemplo, si un acceso a una página A, a menudo, implica el acceso a una página B, entonces la secuencia de acceso A,B ocurrirá repetidamente. Por el contrario, una traza contendrá menos secuencias de referencias únicas que una permutación aleatoria de la misma traza. Así, la relación del número de secuencias de referencias única en una traza a esa en una permutación aleatoria de la misma traza indica el nivel de localidad espacial: más baja esta relación, más alta la localidad espacial de la traza.

Obtener el número total de secuencias únicas en la traza es computacionalmente muy caro, se debería normalmente limitar la longitud máxima de las secuencias que están siendo consideradas. Considerando la traza de accesos a su servidor Web departamental, se encontró que la traza original tiene un número significativamente menor de secuencias únicas que una permutación aleatoria de la traza. Así, al menos ese sitio web particular exhibió un apreciable nivel de localidad espacial.

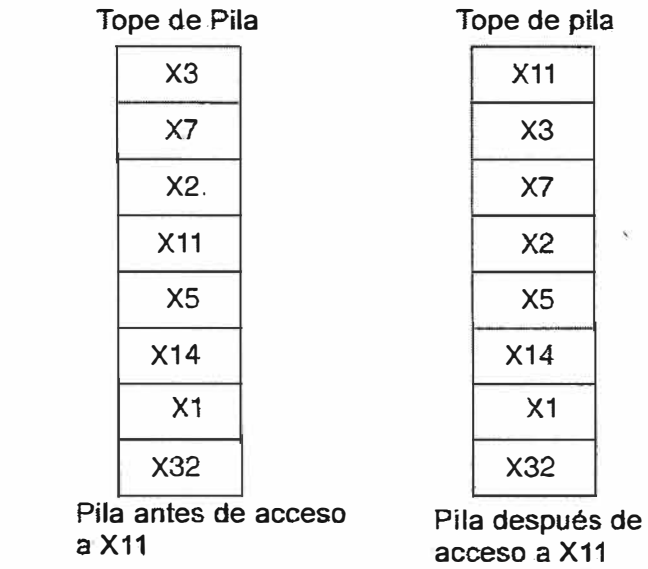

Figura 1.6 Ejemplo de transformación de pila

1.7.5 Tasa de modificaciones de objetos

Otra característica Web sobre la cual la utilidad de caching y replicación dependen es la tasa de actualización de los objetos web. Las tasas de actualización son importantes porque ellos afectan los mecanismos para asegurar que un cache no proporciona contenido obsoleto a los clientes.

Douglis (1997) reportó algunos resultados basados en accesos desde varios cientos de trabajadores de laboratorios de investigación. Aunque no está claro el grado de representatividad esta data es de uso general, el resultado aun da una idea aproximada de las tendencias.

El estudio encontró que muchos de los accesos buscan objetos cuya edad es de más de una semana, donde la edad es definida como la diferencia entre la fecha del acceso y la última modificación del timestamp. De hecho, la edad más frecuente de los objetos fue de más de un mes.

Para caracterizar el efecto del cambio del objeto sobre caching, Douglis estudió la tasa de cambio, la fracción de accesos que regreso un objeto que ha sido modificado desde un acceso previo. Entre todas las respuestas sobre las cuales fue posible determinar si sus objetos fueron o no modificados, la tasa de cambio fue alrededor de 15%. En otras palabras, el 15% de requerimientos repetidos no podrían ser atendidos por el cache desde las copias antiguas de objetos almacenados porque los objetos han cambiado.

La tasa de cambio que afecta actualmente a los caches debería ser algo más baja, porque el número incluye páginas generadas dinámicamente que no son cacheables de ninguna manera. Los caches con mayor población de usuarios deberían entregar aún más hits entre modificaciones de objetos debido a las tasas más altas de requerimientos relativos a la tasa de cambios de objetos.

En general, estas cifras indican que la tasa de modificaciones de objetos es lo suficientemente alta para causar preocupación sobre la consistencia del cache y replica pero no lo suficientemente alta para negar los beneficios de caching y replicación.

CAPÍTULO II
OPTIMIZACIÓN DE LA ACCESIBILIDAD A LOS RID

El presente capítulo describe la metodología para optimizar el acceso de recursos de información digital pública en alto tráfico de internet, en base a lo expuesto en el capitulo marco teórico conceptual.

2.1 Ubicación del problema en un escenario inicial

Como un escenario inicial se va a proponer una empresa que sea un periódico a nivel nacional: El Heraldo Perú. Esta empresa aparte de dedicarse a la venta de periódicos impresos, vende publicidad a través de su página Web, por lo que le interesa que su página Web este siempre disponible y que las vistas a sus noticias Web que son actualizadas consecutivamente no presenten tiempo prolongados en descargarse.

Los servidores de su página Web se encuentran en su local principal contratando a un proveedor de servicio de Internet para hacer pública su página Web. (Figura 2.1).

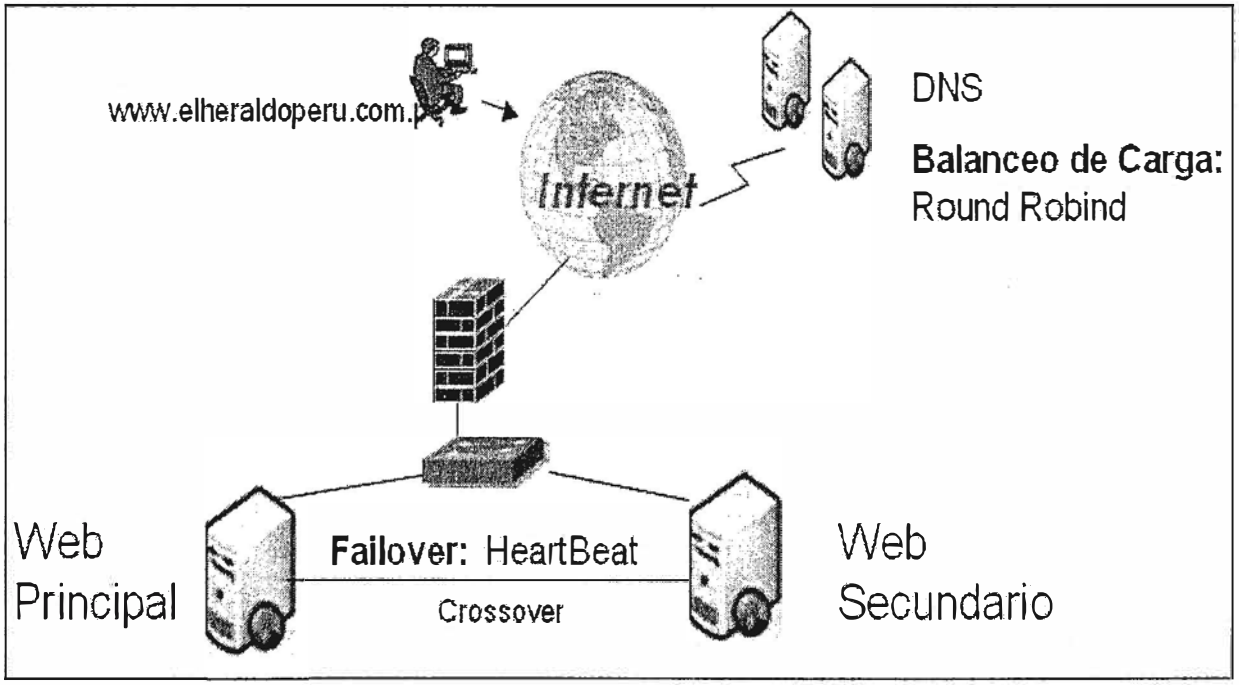

Figura 2.1 Escenario inicial

2.2 Análisis del problema

Viendo esta empresa que sus visitas a sus páginas Web van creciendo en número y en la cantidad, calidad y diversidad de contenido, el ancho de banda de su local se ve saturado, siendo este un problema a larga. Asimismo se observa que ningún servidor se encuentra en alta disponibilidad y que el local donde se encuentran los equipos no brinda

las garantías necesarias en infraestructura eléctrica y ambiente acondicionado para que el hardware funcione correctamente las 24 horas.

2.3 Planteamiento de una solución al problema

Una vez definido el problema y ubicado en un escenario inicial pasamos a plantear una solución teniendo en cuenta el marco teórico contemplado en un inicio. Se ve necesario establecer un procedimiento para plantear una adecuada solución al problema.

Este procedimiento estará comprendido por cuatro etapas: en la primera etapa se procederá con el levantamiento de información relevante al problema, en la segunda etapa se precisan los requerimientos necesarios para la solución, en la tercera etapa se definirá la arquitectura de la solución.

Luego de realizado todo este análisis se pasará a ver el diseño propio de la solución y finalmente su implementación y sometimiento a pruebas.

a. Levantamiento de información

La Figura 2.1 muestra el escenario inicial con la información relevante sobre la cual se efectuará el diseño de la solución.

Para iniciar el planteamiento de una solución se requiere en primer lugar obtener la mayor cantidad de información relevante al problema en cuestión.

Así, si bien ya se ha mencionado algunas características de los tres problemas en mención, se procede a describir todo el estado actual de la red que tenga relación al problema en discusión.

En primer lugar se tiene lentitud en acceder a las páginas web (200ms). La configuración de los servidores que sirven las páginas web se caracteriza por ser sencillo y útil en pequeñas empresas que tienen bajo tráfico de peticiones al servidor web (4000 peticiones por segundo) su alta disponibilidad está basado en otro servidor de similares características conectado a través de un cable cruzado que asume toda la carga de visitas a la página web ante la eventualidad de que el otro equipo falle.

Esta solución por ser simple y de fácil implementación no cuenta con un balanceador de carga inteligente. El balanceo de carga de visitas a las páginas web a los servidores es realizado por medio de un balanceo DNS (round robin). Sin embargo, en un entorno en el cual las visitas a las páginas web son más 40000 peticiones por segundo, la respuesta de estos servicios repercute en su desempeño, con dos servidores que sirvan las páginas web.

Puede concluirse que es inapropiado mantener un esquema de este tipo. Se requiere implementar mecanismos que permitan un mejor desempeño de los equipos a través de técnicas modernas como almacenamiento en cache y balanceo de carga.

.b. Requerimientos para la solución

Los requerimientos planteados son los siguientes: Que la solución permita almacenar en caché el 95% de los objetos estáticos (audio, video, páginas web, etc.), mejorando así los tiempos de respuesta en más de 50%.

c. Arquitectura a utilizar en la solución

En la Figura 2.2 se muestra la arquitectura propuesta. Esta consiste de la colocación de tres servidores que guarden en memoria caché la información que será utilizada por los usuarios. Todas las consultas hechas a los RID serán repartidas por un servidor balanceador a los servidores caché los cuales contendrán la información digital en su memoria. Si el servidor caché no tuviese la información requerida entonces esta será extraída del servidor real para tenerlo en memoria caché para las nuevas consultas que se den. Para lograr la alta disponibilidad se hace necesario poner otro servidor balanceador que realice la misma tarea.

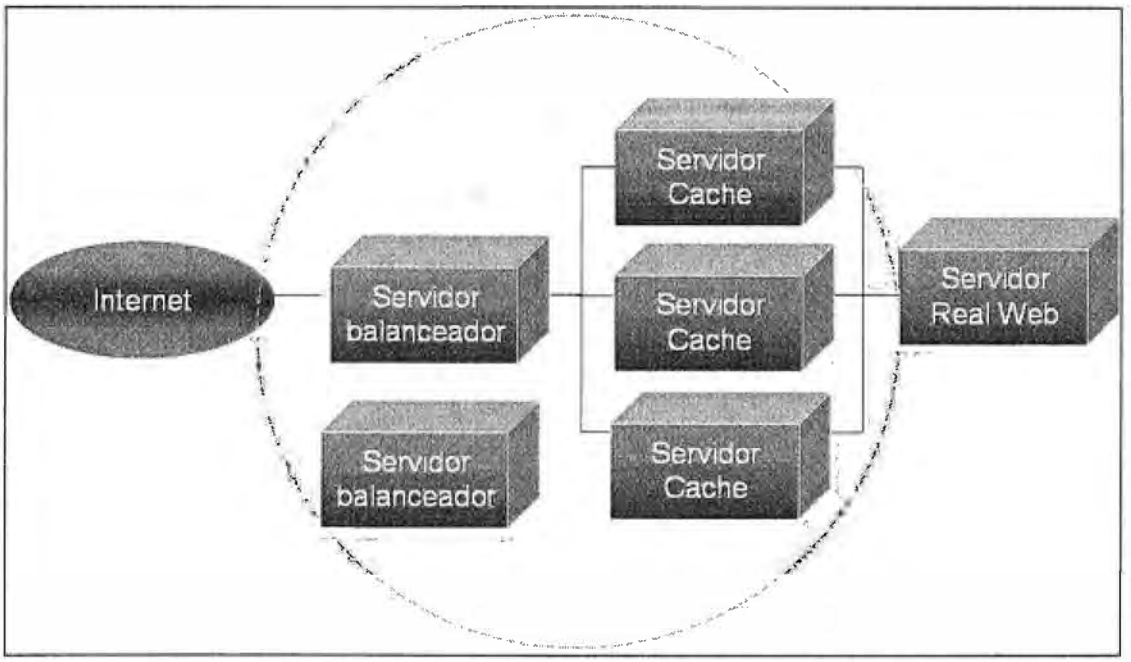

Figura 2.2 Arquitectura propuesta

2.4 Selección de equipos

Para poder lograr dar solución al problema plantead en base al marco teórico descrito, se selecciona los siguientes equipos:

2.4.1 El LVS - Linux Virtual Server

Con este equipo podremos balancear todo el tráfico web a los servidores cache, el servidor empleado es un LVS con Pirahna que estará en alta disponibilidad con otro servidor LVS de las mismas características de hardware y software. Con este equipo podremos lograr:

a. Soporta failover, un LVS principal se encuentra activo, si este falla el LVS secundario asume el papel.

b. Usar los 4 métodos de balanceo conocidos cuando sean necesarios.
c. Poder incorporar hasta 200 nodos.
d. Poder administrar el equipo por una interfaz web.
e. inscribir scripts para el monitoreo de la conexión hacia los nodos.

2.4.2 El Servidor cache

El servidor cache permite almacenar en memoria los recursos de información digital más solicitados para que sean entregados al cliente de forma rápida, el servido cache evita que el servidor real se sature, se usara el nginx como software en los equipos de los servidores cache, entre las principales características del nginx se tiene:

a. Almacena en cache el 95% de los objetos más consultados.
b. Tiempo de respuesta Hit menor a 50ms.
c. Manejo de los timeout de las conexiones TCP.
d. Permite configurar el tiempo de expiración de los objetos.
e. Menor uso de ancho de banda activando mecanismos de compresión.

2.4.3 El Servidor real

El servidor real es un servidor web de aplicaciones que contiene toda la información que será presentada a los clientes, siendo de este modo se usara un servidor web estable como el apache web server. Entres sus características principales tenemos:

a. Está diseñado para funcionar en múltiples plataformas de sistemas operativos y arquitecturas de hardware.
b. Se acomoda a una gran variedad de entornos de aplicaciones a través de su diseño modular.
c. Mejora su desempeño usando solos módulos necesarios.
d. Soporta comprensión de contenido antes de enviarlo al cliente.

Genera cabeceras de expiración y cache-control.

CAPÍTULO III
RESULTADOS

El presente capitulo describe los resultados logrados en el diseño de la solución planteada. En la Tabla 3.1 se describe brevemente los resultados logrados en la implementación de la solución.

Tabla 3.1 Resultados de la solución

Característica	Antes de la solución	Después de la solución
Escalabilidad	No existe escalabilidad.	La escalabilidad esta basado en LVS pirahna.
Alta disponibilidad	Ineficiente	Basado en LVS pirahna
Balanceo de carga web	Se usa registros DNS para el balanceo.	Se usa LVS pirahna
Tiempo de descarga de objetos	200ms.	50ms.
Almacenamiento de objetos en cache	No existe.	Almacena en cache hasta el 95% de objetos.
Número de peticiones por segundo	Soporta hasta 4000 peticiones por segundo.	Soporta más de 40000 peticiones por segundo.
Tiempo de expiración de objetos	No existe.	Se encuentra activo 30 días para imágenes y audio..
Compresión de objetos	No existe.	Se encuentra activo.
Saturación de Servidor real	Saturado en uso de recursos de hardware	Poco uso de recursos de hardware.

3.1 Escalabilidad

En la ubicación del problema en un escenario inicial cuando los recursos de información digital van creciendo progresivamente y se hace necesario contar con mas equipos para responder a la demanda de los usuarios, la configuración y adaptación de estos equipos en tal escenario se vuelve obsoleta por la demora en la configuración de los equipos y en la adaptación del esquema.

En la solución se propone un balanceador de carga web que permite agregar más equipos de forma sencilla, respondiendo de esta manera a la carga progresiva en peticiones a los RID.

3.2 Alta disponibilidad

En el escenario inicial no se contempla alta disponibilidad de los equipos que brindan

los RID, si uno falla toda la carga la asume el otro nodo, siendo las peticiones a los RID más lento debido a que toda la carga la sume un solo equipo. En la solución se contempla alta disponibilidad en el balanceador de carga web por si llegase a fallar asumiendo su papel el balanceador de carga secundario, asimismo si un nodo fallase el balanceador de forma inteligente se da cuenta y deja de enviarle peticiones al nodo hasta que este sea reparado.

3.3 Balanceo de carga web

En el escenario inicial se observa un balanceo de carga web por DNS, siendo este ineficiente debido a que los servidores DNS no pueden saber sin un nodo se encuentra caído o funcionando, asimismo no reparte de manera equitativa todas las peticiones de los clientes a los RID. En la solución se ha colocado un balanceador de carga web que reparte de forma inteligente las peticiones de los clientes a los RID, asimismo el balanceador puede saber si un nodo se encuentra caído o funcionando antes de enviar las peticiones a los RID.

3.4 Tiempo de descarga de objetos

En el escenario inicial todas las peticiones a los RID son directas al servidor web, conllevando un tiempo de procesamiento en el servidor web para que los RID sean entregados a los clientes. En la solución se ha colocado servidores cache que almacena en memoria los RID, de este modo se evita el tiempo de procesamiento del servidor web, logrando reducir el tiempo de respuesta en la descarga de objetos de los RID.

3.5 Almacenamiento de objetos en cache

En el escenario inicial no existe almacenamiento de información en cache siendo todas las peticiones a los RID directas al servidor web. En la solución se cuenta con servidores cache que pueden almacenar en cache hasta el 95% de objetos de los RID.

3.6 Número de peticiones por segundo

En el escenario inicial todas peticiones de los objetos son atendidas por los servidores web hasta 4000 peticiones por segundo por características de hardware y aplicación de software. En la solución al tener servidores cache que no incurre en procesamiento, permite mayor cantidad de conexiones por segundo logrando hasta 40000 conexiones por segundo con los tres servidores cache.

3.7 Tiempo de expiración de objetos

En el escenario inicial no cuenta con características de configuración de expiración de objetos. En la solución se ha configurado que el tiempo de expiración de objetos sea de 30 días para las imágenes y video, de este modo si el cliente vuelve a consultar la misma imagen o video, las imágenes que se mostraran serán las que se encuentran almacenadas en el propio computador del cliente.

3.8 Compresión de objetos

En el escenario inicial, siendo esta solución antigua no explota las nuevas características de los browsers de los clientes que es la de descomprimir la data que se descarga de los servidores que almacena los RID. En la solución se tiene activa esta opción para los objetos tipo texto, de este modo cada vez que el cliente accede a los RID todo los que son tipo texto serán enviados comprimidos, y serán descomprimidos por el browser del cliente, esto permite ahorrar ancho de banda y agilizar la descarga de los RID.

3.9 Servidor real

En el escenario inicial todas la peticiones a los RID son realizadas directamente a los servidores web (Servidor real), sobrecargando los equipos en el uso de los recursos de hardware, llegando en casos a inhibir el servidor. En la solución el que atiende la mayor cantidad de peticiones a los RID son los servidores cache, evitando de este modo que el servidor real se sobrecargue en consumo de recursos de hardware

CONCLUSIONES Y RECOMENDACIONES

1. Si bien existen varias alternativas de solución para el problema planteado se opto por esta porque es la que más se adecua al tráfico de internet generado por las empresas nacionales que brindan acceso a los recursos de información digital.
2. Con la solución propuesta se ha podido mejorar el tiempo de respuesta en consultas a los recursos de información digital en más del 50%.
3. De acuerdo al diagrama de la arquitectura se aprecia que el diseño es escalable y de alta disponibilidad.
4. Con la solución propuesta se ha podido almacenar en cache más del 95% de objetos web.
5. Con la solución propuesta se ha podido liberar el alto consumo de recursos de hardware en el servidor real.
6. Con la solución propuesta se ha logrado incrementar el número de peticiones concurrentes en más de 40000 peticiones por segundo.
7. En los balanceadores de carga el punto de enlace para el hearbeat debe ser un enlace directo Ethernet, de esta manera nos aseguramos que el tráfico que monitoreo el estado de los balanceadores no se vea afectado por el tráfico propio de los balanceadores a los servidores cache.
8. Es recomendable monitorear permanentemente el estado de uso de los recursos de hardware de los servidores, como también el consumo de ancho de banda que utiliza el balanceador.
9. Es recomendable mantener desactivado la opción de generación de logs en los servidores cache para un mejor uso de los recursos de hardware de los servidores.
10. Es recomendable tener actualizado y parchado el sistema operativo y las aplicaciones en los servidores.
11. Es recomendable que los equipos de hardware tenga contrato de soporte permanente con la empresa manufacturera.

ANEXO A
INSTALACIONES Y CONFIGURACIONES

En este anexo se describe la instalación y configuración de los diversos componentes de la solución: 1) El servidor balanceador, 2) El servidor caché, y 3) El Servidor real

A.1 El Servidor balanceador

Se describe la instalación y configuración.

a. Instalación

Para instalar el balanceador LVS se realizara del siguiente modo:

#yum groupinstall “Clustering”

b. Configuración

Los pasos son los siguientes:

1. Antes de proceder a usar la herramienta web de configuración se procede a poner clave a la interfaz web para la cuenta piranha del siguiente modo:

#piranha-passwd

2. Luego se levanta el servicio de la siguiente manera:

#service piranha-gui start

3. Agregar la IP primaria de nuestro cluster en la imagen se muestra la 192.160.2.100. Figura A.1.
4. Agregar la IP del Failover de nuestro cluster. Figura A.2
5. Agregar nuestro servidor virtual en este ejemplo vamos a utilizar el puerto 42201 para la entrada al balanceo y utilizaremos el algoritmo de balanceo Round Robin. Figura A.3
6. Se agrega las direcciones IP de los servidores cache, lo dejamos activo. Figura A.4
7. Por último la pantalla de Monitoring Scripts se queda como estaba por default. Figura A.5
8. Por último se tendrá que modificar el archivo /etc/sysctl.conf, dejando el parámetro de redirección de este modo net.ipv4.ip_forward = 1, para activar los cambios digitamos el comando # sysctl -p

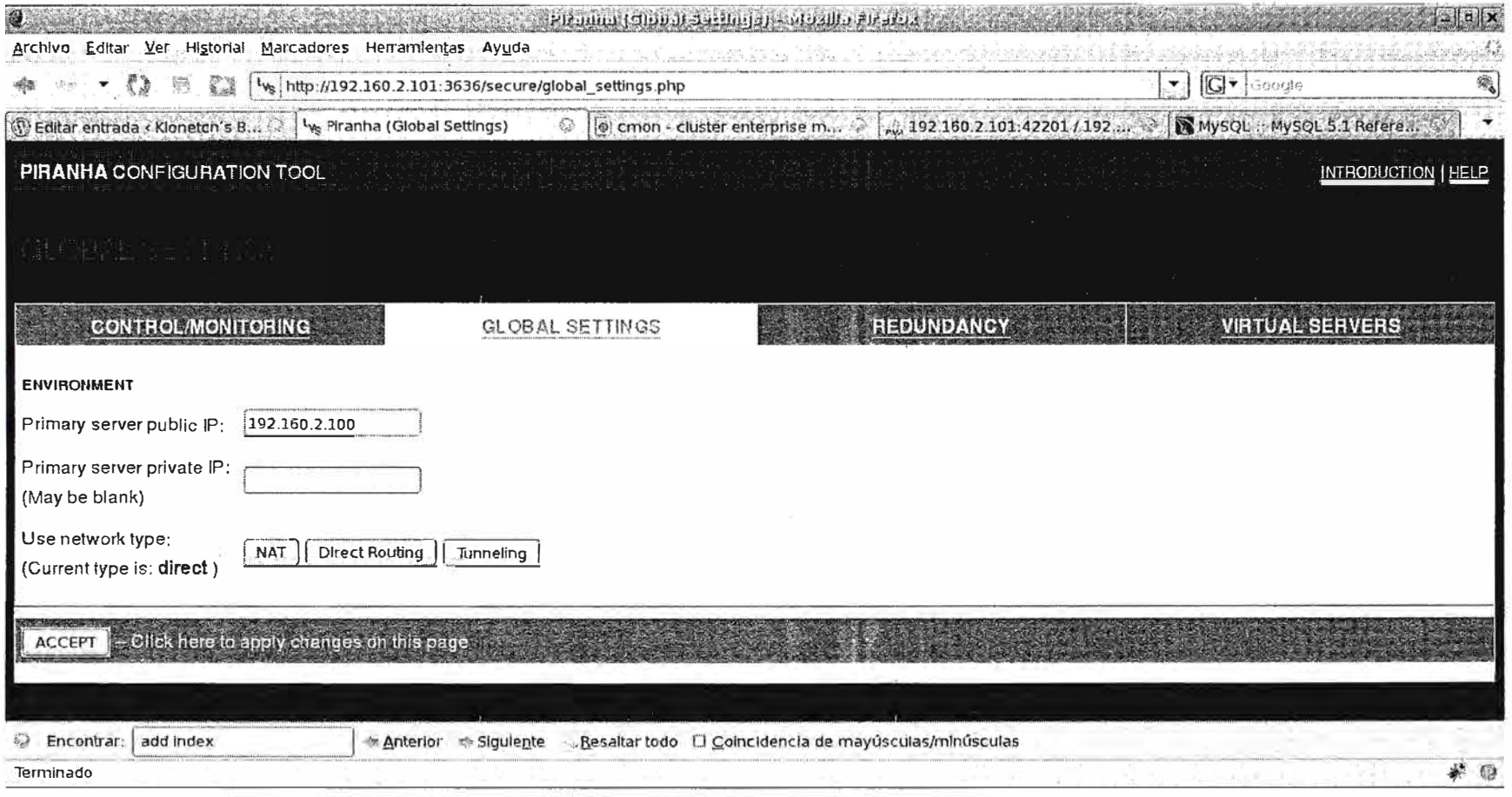

Figura A.1 Configuración global

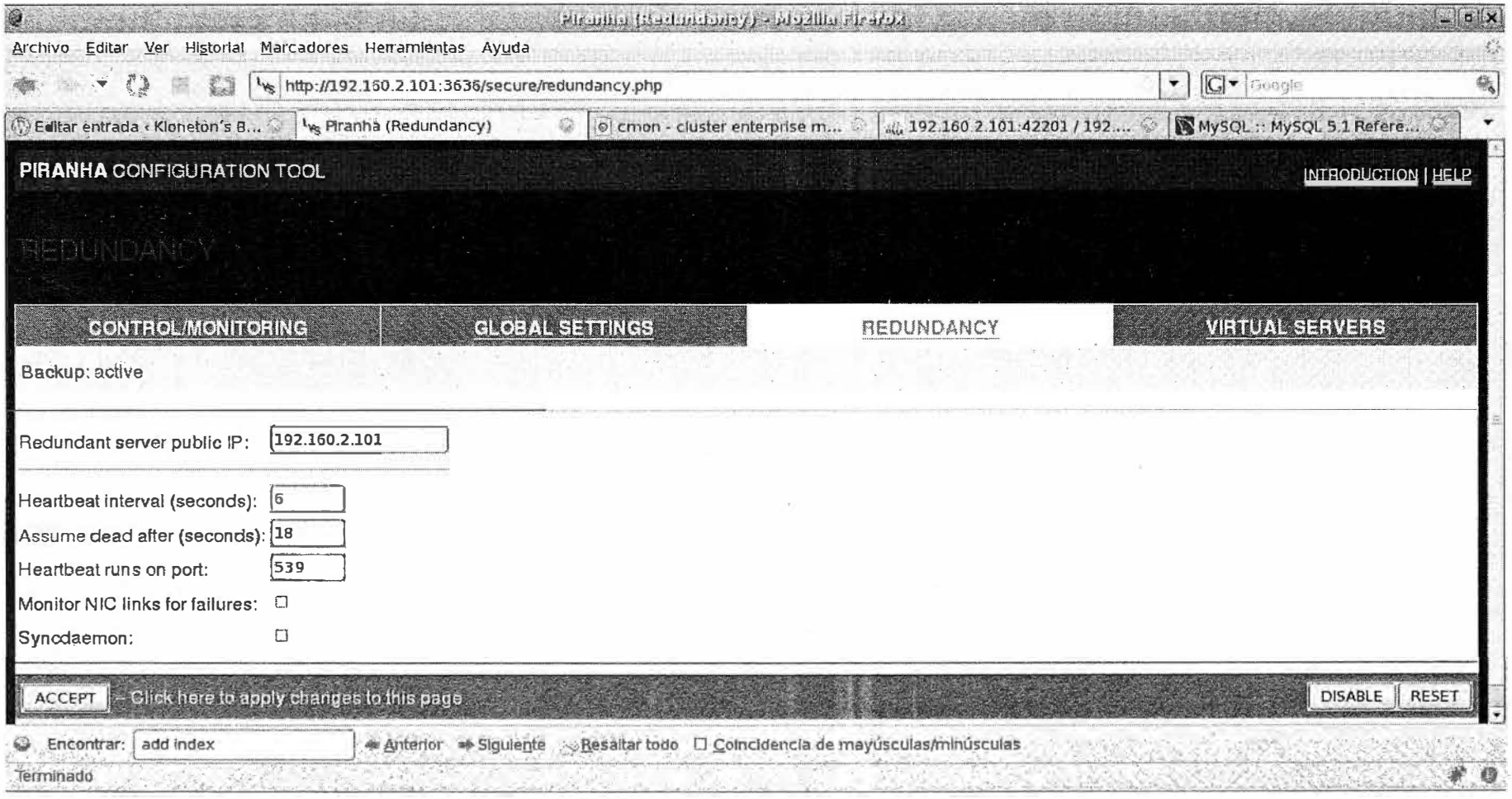

Figura A.2 Configuración de redundancia (fail over)

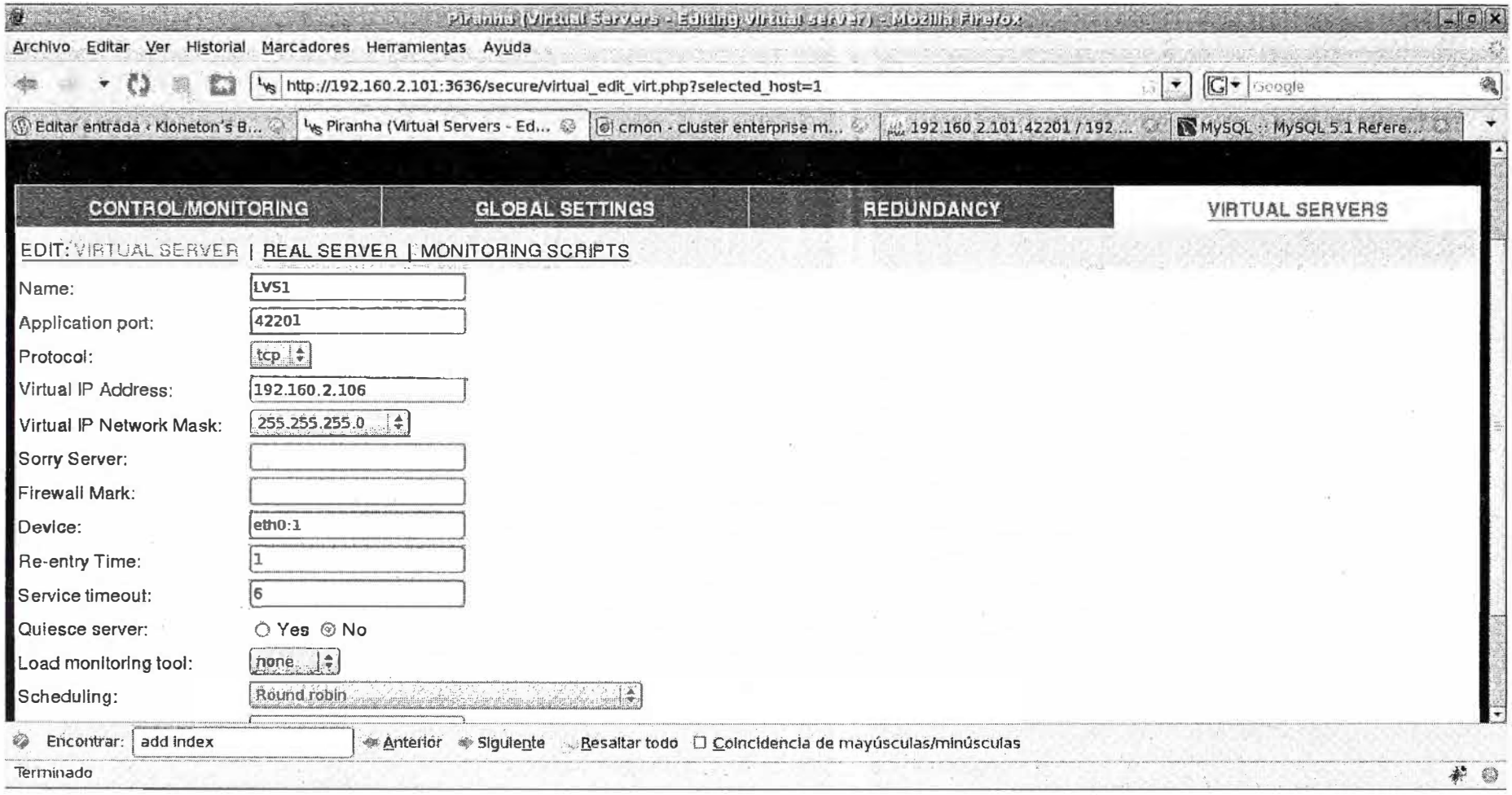

Figura A.3 Configuración del balanceo

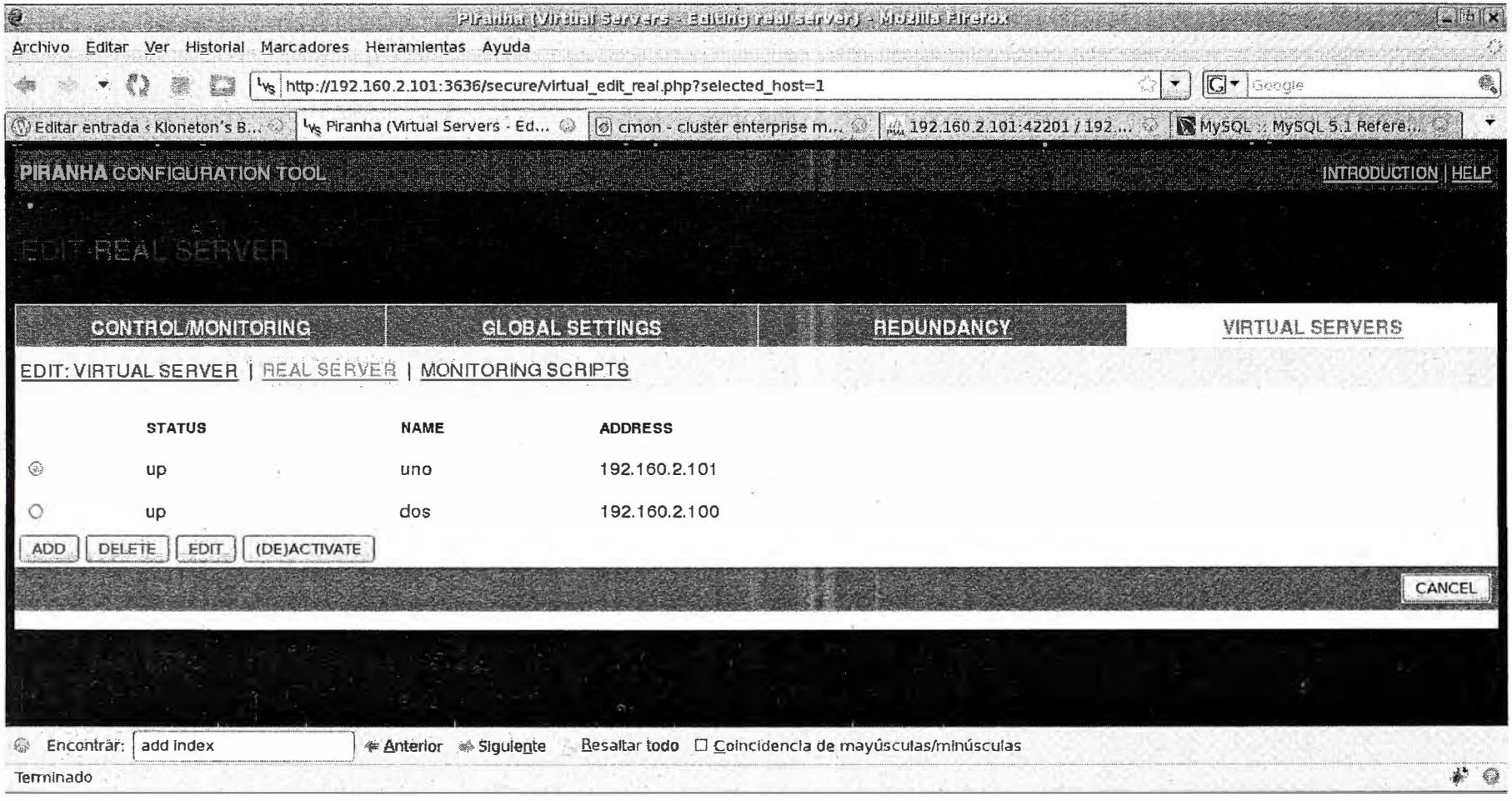

Figura A.4 Activación de los nodos

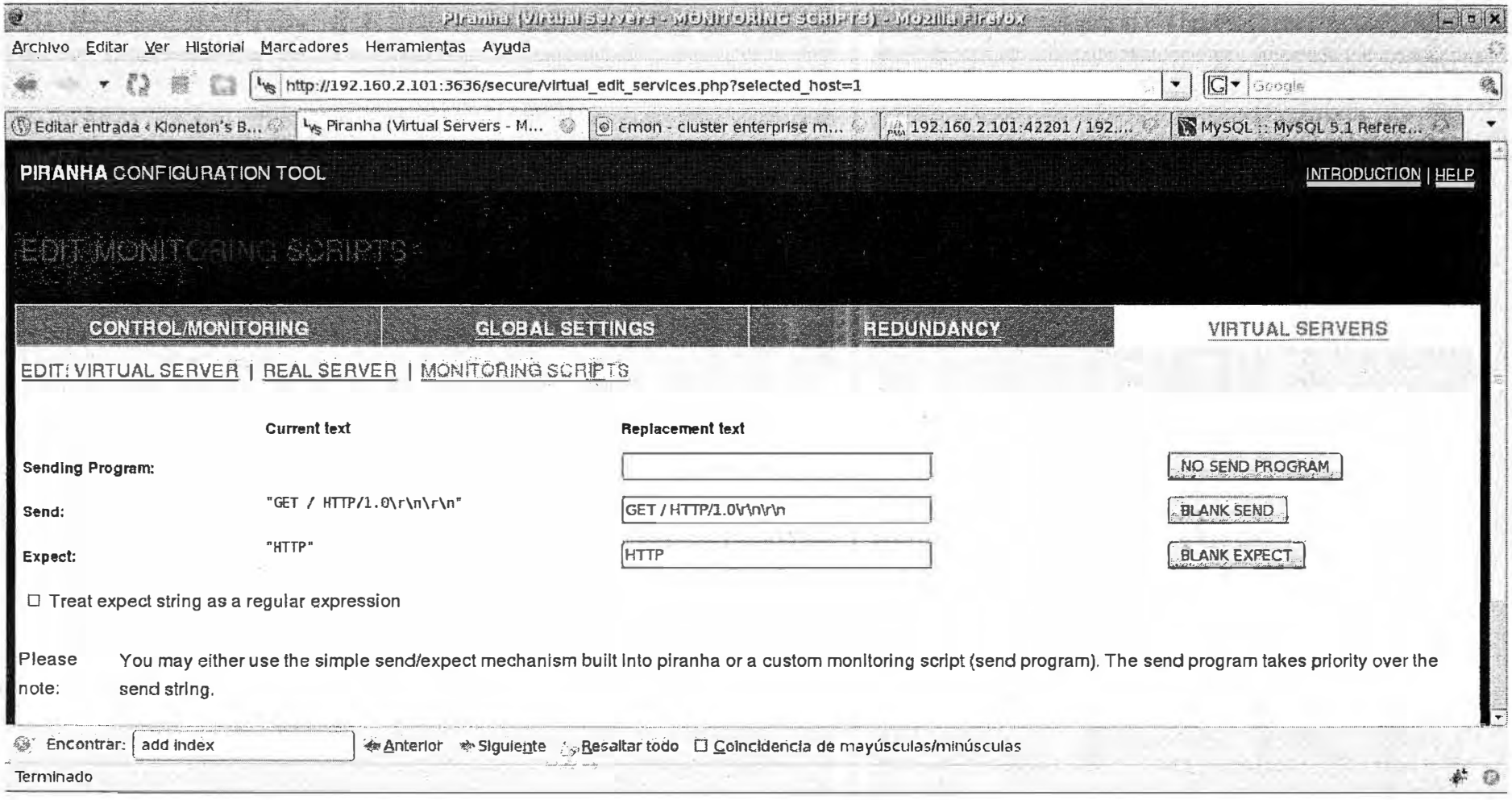

Figura A.5 Configuración de monitoreo

A.2 El servidor caché

Se desarrolla la metodología para instalar y configurar el servidor caché.

a. Instalación

Para la instalación del nginx se ha seguido los siguientes pasos:

1. Descargar de la página principal el código fuente estable del nginx:

   http://nginx.org/download/nginx-0.7.67.tar.gz

2. Crear la cuenta nginx en el sistema operativo

   #adduser nginx –d /var/lib/nginx –s /bin/false -c ”Nginx user”

3. Instalar paquetes necesarios para la compilación del código fuente

   yum install gcc make pcre-devel openssl-devel

4. Desempaquetar el codigo fuente nginx y poceder a su compilación y instalación

```
#tar -xzvf nginx-0.7.67.tar.gz
#./configure --user=nginx --group=nginx --prefix=/usr/share/nginx --sbin-
path=/usr/sbin/nginx --conf-path=/etc/nginx/nginx.conf --error-log-
path=/var/log/nginx/error.log --http-log-path=/var/log/nginx/access.log --http-client-body-
temp-path=/var/lib/nginx/tmp/client_body --http-proxy-temp-path=/var/lib/nginx/tmp/proxy -
-http-fastcgi-temp-path=/var/lib/nginx/tmp/fastcgi --pid-path=/var/run/nginx.pid --lock-
path=/var/lock/subsys/nginx --with-http_ssl_module --with-http_realip_module --with-
http_addition_module --with-http_sub_module --with-http_dav_module --with-
http_flv_module --with-http_gzip_static_module --with-http_stub_status_module --with-
http_perl_module --with-mail --with-mail_ssl_module
#make
#make install
```

5. Crear archivos necesarios con sus respectivos permisos

```
#mkdir -p /var/lib/nginx/tmp
#chown -R nginx:nginx /var/lib/nginx
```

6. Creando script de parada y inicio del nginx

```
#vi /etc/init.d/nginx
#!/bin/sh
#
# nginx - this script starts and stops the nginx daemin
#
# chkconfig:   - 85 15
# description:  Nginx is an HTTP(S) server, HTTP(S) reverse \
#               proxy and IMAP/POP3 proxy server
# processname: nginx
# config:      /etc/nginx/nginx.conf
# config:      /etc/sysconfig/nginx
# pidfile:     /var/run/nginx.pid
# Source function library.
. /etc/rc.d/init.d/functions
# Source networking configuration.
. /etc/sysconfig/network
# Check that networking is up.
```

```
[ "$NETWORKING" = "no" ] && exit 0
nginx="/usr/sbin/nginx"
prog=$(basename $nginx)
NGINX_CONF_FILE="/etc/nginx/nginx.conf"
[ -f /etc/sysconfig/nginx ] && . /etc/sysconfig/nginx
lockfile=/var/lock/subsys/nginx
start() {
 [ -x $nginx ] || exit 5
 [ -f $NGINX_CONF_FILE ] || exit 6
echo -n $"Starting $prog: "
daemon $nginx -c $NGINX_CONF_FILE
retval=$?
echo
 [ $retval -eq 0 ] && touch $lockfile
return $retval
}
stop() {
echo -n $"Stopping $prog: "
killproc $prog
retval=$?
echo
[ $retval -eq 0 ] && rm -f $lockfile
return $retval
}
restart() {
configtest || return $?
stop
start
}
reload() {
configtest || return $?
echo -n $"Reloading $prog: "
killproc $nginx -HUP
RETVAL=$?
echo
}
force_reload() {
restart
}
configtest() {
$nginx -t -c $NGINX_CONF_FILE
}
rh_status() {
status $prog
}
rh_status_q() {
rh_status >/dev/null 2>&1
}
case "$1" in
start)
rh_status_q && exit 0
$1
;;
stop)
```

```
rh_status_q || exit 0
$1
;;
restart|configtest)
$1
;;
reload)
rh_status_q || exit 7
$1
;;
force-reload)
force_reload
;;
status)
rh_status
;;
condrestart|try-restart)
rh_status_q || exit 0
;;
*)
echo $"Usage: $0 {start|stop|status|restart|condrestart|try-restart|reload|force-
reload|configtest}"
exit 2
esac
#chkconfig --add nginx
#chkconfig nginx on
```

b. Configuración

El siguiente el código del archivo de configuración:

```
user             nginx;
worker_processes  2;
error_log         /var/log/nginx/error.log;
pid               /var/run/nginx.pid;
events {
   worker_connections  1024;
}
http {
include       /etc/nginx/mime.types;
default_type  application/octet-stream;
log_format main '$remote_addr - $remote_user [$time_local] '
'"$request" $status $body_bytes_sent '
'"$http_referer" "$http_user_agent"';
access_log       /var/log/nginx/access.log  main;
## Timeouts
client_body_timeout   60;
client_header_timeout 60;

#expires              24h;
keepalive_timeout     60 60;
send_timeout          60;
## General Options
ignore_invalid_headers   on;
keepalive_requests      100;
```

```
limit_zone gulag $binary_remote_addr 5m;
recursive_error_pages    on;
sendfile                 on;
server_name_in_redirect off;
server_tokens            off;
## TCP options
tcp_nodelay on;
tcp_nopush  on;
## Compression
gzip on;
gzip_buffers 16 8k;
gzip_comp_level   9;
gzip_http_version 1.0;
gzip_min_length   0;
gzip_proxied      any;
gzip_types            text/plain image/x-icon image/png image/jpg image/jpeg text/js text/php
text/css application/xml application/xml+rss text/javascript;
gzip_vary         on;
port_in_redirect    off;
proxy_buffering              on;
proxy_cache_min_uses      3;
proxy_cache_path          /var/lib/nginx/tmp/proxy levels=1:2 keys_zone=cache:10m
inactive=30m max_size=5000M;
proxy_cache_valid         any 15m;
proxy_ignore_client_abort off;
proxy_intercept_errors    off;
proxy_next_upstream       error timeout invalid_header;
proxy_redirect            off;
proxy_set_header          X-Forwarded-For $remote_addr;
proxy_connect_timeout     60;
proxy_send_timeout        60;
proxy_read_timeout        60;
server {
listen       80;
server_name  heraldoperu.com.pe www.elheraldoperu.com.pe;
#access_log  /var/log/nginx/heraldo_access.log  main;
access_log  off;
error_log   /var/log/nginx/heraldo_error.log;
location /administrator/ {
proxy_pass      http://www.heraldoperu.com.pe/administrator/;
}
location ~ ^/(images|javascripts|stylesheets)/ {
proxy_cache           cache;
proxy_cache_valid      200 1d;
proxy_cache_valid      403 30m;
proxy_cache_valid      404 30m;
proxy_cache_use_stale  error timeout invalid_header updating http_500 http_502
http_503 http_504;
proxy_ignore_headers   Expires Cache-Control;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header Host $host;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
}
error_page  404              /index.php;
```

```
location = /404.html {
root   /var/www/sites/;
}
error_page   500 502 503 504  /50x.html;
location = /50x.html {
root   /usr/share/nginx/html;
}
}
}
```

A.3 El servidor real

Se desarrolla la metodología para instalar y configurar el servidor real.

a. Instalación

Para el servidor real se utilizara el apache web server, su instalación se muestra a continuación

b. Configuración

Se desarrollan las siguientes configuraciones: 1) Configurar el servidor web httpd.conf, 2) Configurando un host virtual, 3) Configurando modulo deflate, 4) Configurando modulo expires

b.1 Configurar el servidor web httpd.conf

Una vez instalado se procede a configurar el servidor editando el archivo /etc/httpd/httpd.conf:

```
ServerTokens Prod
ServerRoot "/etc/httpd"
PidFile run/httpd.pid
FileETag MTime Size
Timeout 45
KeepAlive Off
MaxKeepAliveRequests 400
KeepAliveTimeout 6
<IfModule prefork.c>
StartServers       8
MinSpareServers    5
MaxSpareServers   20
ServerLimit      200
MaxClients       200
MaxRequestsPerChild  800
</IfModule>
Listen 80
LoadModule auth_basic_module modules/mod_auth_basic.so
#LoadModule auth_digest_module modules/mod_auth_digest.so
LoadModule authn_file_module modules/mod_authn_file.so
#LoadModule authn_alias_module modules/mod_authn_alias.so
#LoadModule authn_dbm_module modules/mod_authn_dbm.so
#LoadModule authn_default_module modules/mod_authn_default.so
LoadModule authz_host_module modules/mod_authz_host.so
LoadModule authz_user_module modules/mod_authz_user.so
#LoadModule authz_owner_module modules/mod_authz_owner.so
#LoadModule authz_groupfile_module modules/mod_authz_groupfile.so
```

```
#LoadModule authz_dbm_module modules/mod_authz_dbm.so
#LoadModule authz_default_module modules/mod_authz_default.so
#LoadModule ldap_module modules/mod_ldap.so
LoadModule include_module modules/mod_include.so
LoadModule log_config_module modules/mod_log_config.so
LoadModule logio_module modules/mod_logio.so
LoadModule env_module modules/mod_env.so
LoadModule ext_filter_module modules/mod_ext_filter.so
#LoadModule mime_magic_module modules/mod_mime_magic.so
LoadModule expires_module modules/mod_expires.so
LoadModule deflate_module modules/mod_deflate.so
LoadModule headers_module modules/mod_headers.so
LoadModule usertrack_module modules/mod_usertrack.so
LoadModule setenvif_module modules/mod_setenvif.so
LoadModule mime_module modules/mod_mime.so
LoadModule autoindex_module modules/mod_autoindex.so
LoadModule info_module modules/mod_info.so
LoadModule vhost_alias_module modules/mod_vhost_alias.so
LoadModule negotiation_module modules/mod_negotiation.so
LoadModule dir_module modules/mod_dir.so
LoadModule actions_module modules/mod_actions.so
LoadModule speling_module modules/mod_speling.so
LoadModule userdir_module modules/mod_userdir.so
LoadModule alias_module modules/mod_alias.so
LoadModule rewrite_module modules/mod_rewrite.so
LoadModule cache_module modules/mod_cache.so
LoadModule suexec_module modules/mod_suexec.so
LoadModule disk_cache_module modules/mod_disk_cache.so
LoadModule file_cache_module modules/mod_file_cache.so
Include conf.d/*.conf
User apache
Group apache
ServerAdmin root@localhost
ServerName www.elheraldo.com.pe
UseCanonicalName Off
DocumentRoot "/var/www/html"
<Directory />
Options FollowSymLinks
AllowOverride None
</Directory>
<Directory "/var/www/html">
Options Indexes FollowSymLinks
AllowOverride None
Order allow,deny
Allow from all
</Directory>
<IfModule mod_userdir.c>
UserDir disable
</IfModule>
DirectoryIndex index.php index.html index.html.var
AccessFileName .htaccess
<Files ~ "^\.ht">
Order allow,deny
Deny from all
```

```
</Files>
TypesConfig /etc/mime.types
DefaultType text/plain
<IfModule mod_mime_magic.c>
MIMEMagicFile conf/magic
</IfModule>
HostnameLookups Off
ErrorLog logs/error_log
LogLevel warn
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
LogFormat "%h %l %u %t \"%r\" %>s %b" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent
LogFormat "%{X-Forwarded-For}i %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-
Agent}i\"" nginxcombined
CustomLog logs/access_log combined
ServerSignature Off
IndexOptions FancyIndexing VersionSort NameWidth=* HTMLTable
AddIconByEncoding (CMP,/icons/compressed.gif) x-compress x-gzip
AddIconByType (TXT,/icons/text.gif) text/*
AddIconByType (IMG,/icons/image2.gif) image/*
AddIconByType (SND,/icons/sound2.gif) audio/*
AddIconByType (VID,/icons/movie.gif) video/*
AddIcon /icons/binary.gif .bin .exe
AddIcon /icons/binhex.gif .hqx
AddIcon /icons/tar.gif .tar
AddIcon /icons/world2.gif .wrl .wrl.gz .vrml .vrm .iv
AddIcon /icons/compressed.gif .Z .z .tgz .gz .zip
AddIcon /icons/a.gif .ps .ai .eps
AddIcon /icons/layout.gif .html .shtml .htm .pdf
AddIcon /icons/text.gif .txt
AddIcon /icons/c.gif .c
AddIcon /icons/p.gif .pl .py
AddIcon /icons/f.gif .for
AddIcon /icons/dvi.gif .dvi
AddIcon /icons/uuencoded.gif .uu
AddIcon /icons/script.gif .conf .sh .shar .csh .ksh .tcl
AddIcon /icons/tex.gif .tex
AddIcon /icons/bomb.gif core
AddIcon /icons/back.gif ..
AddIcon /icons/hand.right.gif README
AddIcon /icons/folder.gif ^^DIRECTORY^^
AddIcon /icons/blank.gif ^^BLANKICON^^
DefaultIcon /icons/unknown.gif
ReadmeName README.html
HeaderName HEADER.html
AddLanguage ca .ca
AddLanguage cs .cz .cs
AddLanguage da .dk
AddLanguage de .de
AddLanguage el .el
AddLanguage en .en
AddLanguage eo .eo
AddLanguage es .es
```

```
AddLanguage et .et
AddLanguage fr .fr
AddLanguage he .he
AddLanguage hr .hr
AddLanguage it .it
AddLanguage ja .ja
AddLanguage ko .ko
AddLanguage ltz .ltz
AddLanguage nl .nl
AddLanguage nn .nn
AddLanguage no .no
AddLanguage pl .po
AddLanguage pt .pt
AddLanguage pt-BR .pt-br
AddLanguage ru .ru
AddLanguage sv .sv
AddLanguage zh-CN .zh-cn
AddLanguage zh-TW .zh-tw
LanguagePriority en ca cs da de el eo es et fr he hr it ja ko ltz nl nn no pl pt pt-BR ru sv
zh-CN zh-TW
ForceLanguagePriority Prefer Fallback
#AddDefaultCharset UTF-8
AddType application/x-compress .Z
AddType application/x-gzip .gz .tgz
AddHandler type-map var
AddType text/html .shtml
AddOutputFilter INCLUDES .shtml
Alias /error/ "/var/www/error/"
<IfModule mod_negotiation.c>
<IfModule mod_include.c>
<Directory "/var/www/error">
AllowOverride None
Options IncludesNoExec
AddOutputFilter Includes html
AddHandler type-map var
Order allow,deny
Allow from all
LanguagePriority en es de fr
ForceLanguagePriority Prefer Fallback
</Directory>
</IfModule>
BrowserMatch "Mozilla/2" nokeepalive
BrowserMatch "MSIE 4\.0b2;" nokeepalive downgrade-1.0 force-response-1.0
BrowserMatch "RealPlayer 4\.0" force-response-1.0
BrowserMatch "Java/1\.0" force-response-1.0
BrowserMatch "JDK/1\.0" force-response-1.0
BrowserMatch "Microsoft Data Access Internet Publishing Provider" redirect-carefully
BrowserMatch "MS FrontPage" redirect-carefully
BrowserMatch "^WebDrive" redirect-carefully
BrowserMatch "^WebDAVFS/1.[0123]" redirect-carefully
BrowserMatch "^gnome-vfs/1.0" redirect-carefully
BrowserMatch "^XML Spy" redirect-carefully
BrowserMatch "^Dreamweaver-WebDAV-SCM1" redirect-carefully
NameVirtualHost *:80
```

TraceEnable Off

b.2 Configurando un host virtual

Crear el archivo sites.conf en /etc/httpd/conf.d con el siguiente contenido:

<VirtualHost *:80>
ServerAdmin webmaster@elheraldo.com.pe
DocumentRoot /var/www/sites
ServerName www.elheraldo.com.pe
ServerAlias elheraldo.com.pe
ErrorLog logs/elheraldo.com.pe-error_log
CustomLog logs/elheraldo.com.pe-access_log nginxcombined
DirectoryIndex index.html index.php index.htm index.shtml index.php4 index.php3 index.phtml index.cgi
<Directory "/var/www/sites">
Options -Indexes FollowSymlinks MultiViews
AllowOverride All
Order allow,deny
Allow from all
</Directory>
</VirtualHost>

b.3 Configurando modulo deflate

Crear el archivo deflates.conf en /etc/httpd/conf.d con el siguiente contenido:

<IfModule mod_deflate.c>
AddOutputFilterByType DEFLATE text/html text/plain text/xml text/css text/javascript application/x-javascript
</IfModule>

b.4 Configurando modulo expires

Crear el archivo expires.conf en /etc/httpd/conf.d con el siguiente contenido:

ExpiresActive On
ExpiresByType image/jpeg "access plus 1 month"
ExpiresByType image/gif "access plus 1 month"
ExpiresByType image/jpg "access plus 1 month"
ExpiresByType image/png "access plus 1 month"
ExpiresByType text/css "access plus 1 month"
ExpiresByType text/javascript "access plus 1 month"
ExpiresByType application/x-javascript "access plus 1 month"
ExpiresByType application/x-shockwave-flash "access plus 1 month"
ExpiresByType image/swf "access plus 1 month"

ANEXO B
GLOSARIO DE TÉRMINOS

AS	Sistema autónomo
ANSI	American National Standards Institute
BIOS	Sistema básico de entrada/salida de datos
Caché	Sistema especial de almacenamiento de alta velocidad.
CIDR	Classless Interdomain Routing
CNAME	Regsitro DNA que permite tener un alias en otro dominio.
CPU	Unidad de proceso central
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
Firewall	Sistema diseñado para bloquear el acceso no autorizado
Firmware	Es un bloque de instrucciones de programa para propósitos específicos
Ethernet	Estándar de redes de computadoras de área local con CSMA/CD
Heartbeat	Programa que proporciona infraestructura de cluster.
Host	dispositivo terminal en una red de datos
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IP	Protocolo de Internet.
LAN	Local Area Networks
Nodo	Equipo que recibe las peticiones de un balanceador.
ISO	International Organization for Standardization
LVS	Linux Virtual Servel
NAT	Network address translation (NAT)
Networking	Interconexión de cualquier grupo de dispositivos de red
ROUTER	Dispositivo para interconexión de redes de computadoras.
Switch	Dispositivo digital de lógica de interconexión entre redes
TCP	Protocolo de Control de Transmisión.
TIC	Tecnologías de la Información y la Comunicación.
UDP	Protocolo de datagramas de usuario
UPS	Uninterruptible Power Supply, o Sistema de potencia ininterrumpida.
URL	Uniform Resource Locator, o localizador uniforme de recurso.
VLAN	Red de área local virtual.
VPN	Red privada virtual
Verisign	Proveedor de confianza de servicios de infraestructura de Internet

BIBLIOGRAFÍA

[1] A Larry L. Peterson, Bruce S. Davie "Computer Networks, 4th Edition: A Systems Approach" 2007, ISBN 978-0-12-370548-8.

[2] Kopparapu, Chandra "Load Balancing Servers, Firewalls, and Caches", Wiley Computer Publishing 2002, ISBN 0-471-41550-2

[3] Tony Bourke "Server Load Balancing" O'Reilly 2001, ISBN 0-596-00050-2

[4] http://www.linux-ha.org "The High Availability Linux Project", última actualización 24 abril del 2007.

[5] David Gourley "HTTP the guide definitive" O'Reilly Media Inc. 2002, ISBN 1-56592-2

[6] Balachander Krishnamurthy, Jennifer Rexford "Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement " Adisson Wesley Pearson Education 2001, ISBN 0-201-71088-9

[7] Rabinovich, Michael, et al "Web Caching and Replication", Pearson Education 2001. ISBN 0-201-61570-3.

[8] Duane Wessels "Web Caching" O'Reilly Internet Series 2001, ISBN 1-56592-536

[9] Henderson, Cal "Building Scalable Web Sites", O'Reilly Media Inc. 2006, ISBN 978-0-59-610235-7

[10] Souders, Steve "High Performance Web Sites", O'Reilly Media Inc.2007, ISBN 978-0-596-52930-7

[11] Patrick Killelea, "Web Performance Tuning, 2nd Edition" O'Reilly Internet 2002, ISBN 0-596-00172.