Universidad Nacional de La Pampa
Facultad de Ingeniería

PROYECTO FINAL

HACIA UN BROKER UNIVERSAL PARA LA DISTRIBUCIÓN EFICIENTE DE CARGA EN SERVIDORES DISTRIBUIDOS HETEROGENEOS

AUTOR: BORGEAUD, MATIAS
DIRECTOR: MARTIN, MARIA DE LOS ANGELES

2008

A Laura,

Por todo el amor que me das, y por ser la luz de cada uno de mis días. La mayor satisfacción es tenerte para compartir tantas cosas lindas. Te lo dedico a vos, mi vida.

INDICE

PREFACIO

Debemos aprender con el tiempo a disfrutar de los desafíos. No digo que se deba estar permanentemente en busca de ellos, pero sí creo que tenemos que ser capaces de aceptarlos y encararlos con un cierto disfrute y hasta en ocasiones, proponernos alguno.

Este proyecto no es sólo el trabajo final de mi carrera de grado, sino también un desafío al cual disfrutaré muchísimo compartir con ustedes.

Entre los motivos que me han llevado a este desarrollo, debo resaltar uno en particular: el problema que significa lograr una distribución eficiente de carga para utilizar en forma óptima los recursos computacionales de un conjunto de servidores interconectados, o lo que comúnmente denominamos *balance de carga.* Este problema es considerado uno de los temas más interesantes dentro del área de estudio de los sistemas distribuidos, para el cual se han propuesto soluciones de lo más variadas [1] [2] [3]. Sin embargo creo que todavía falta andar más camino.

Por otro lado, el avance de la tecnología ha solucionado muchos problemas que de otro modo no hubiesen podido resolverse. Lo que requiere que nosotros, los profesionales del software, equiparemos este avance. Además, “muchos” no significa todos y así es como nacen propuestas como ésta.

El objetivo que persigue esta propuesta es obtener una solución por software al problema del balance de carga en servidores distribuidos, atendiendo la necesidad de mayores y mejores recursos con bajos costos, requerida por el mercado en constante evolución.

La organización de este trabajo es como sigue: la primera parte trata sobre el problema de distribución de carga desde un enfoque teórico y

deja sentadas las bases conceptuales para enfrentar la búsqueda de soluciones. En esta sección, el capítulo I es una introducción al tema, en el que se plantea la evolución de los sistemas distribuidos y las tendencias actuales que reflejan la necesidad concreta de distribución de carga. El capítulo II hace un tratamiento especial sobre las formas de distribución, las soluciones tempranas en este sentido y el estado del arte. La segunda parte es la aplicación práctica del trabajo, en la cual se presenta la solución propuesta, se avanza sobre características técnicas de implementación y se analizan resultados importantes de las pruebas realizadas. En esta sección, el capítulo III presenta la solución propuesta, *ublobs*, su aporte y las directivas para su instalación y puesta en funcionamiento. Sobre el final de este capítulo, se presentan pruebas de performance y de stress sobre ésta y otras soluciones, para justificar las mejoras pretendidas. El capítulo IV desarrolla las conclusiones, a partir de los resultados anteriores, dejando planteadas tareas pendientes y direcciones para desarrollos futuros.

Este trabajo está dirigido a profesionales, formados o en formación en el área de la informática, y especialmente a aquellos que se perfilen como administradores de red o avocados a la planificación de servicios. Si bien la problemática planteada en el comienzo de este trabajo final será resuelta con el avance de los capítulos, el tema en sí no tiene un punto final. Crecerá, cambiará, habrá otras propuestas. Y cuento con ello.

RECONOCIMIENTOS

Quiero agradecer a mi madre, mi continua fuente de inspiración y modelo de persona, por el amor que me ha brindado y el apoyo incondicional. A Lino, por ser una gran persona, por su enorme sabiduría y sentido común paradójicamente poco común hoy día. A mis hermanos Esteban y Celina, a los que veo poco y extraño mucho, más que gracias debiera ir un pedido de perdón. A Ester y Juan, por quererme y malcriarme como un hijo. A mi jefe, aunque prefiero más compañero de trabajo y amigo, José Luis por confiar en mí y dejarme ser. A todos los profesores que estuvieron para ayudarme, enseñarme, moldearme, y con los que compartí momentos inolvidables. En especial a “Peti”, mi tutora, por ser una guía imprescindible en este trabajo.

No quiero dejar pasar por alto mi eterno agradecimiento a la Facultad de Ingeniería de la UNLPam toda, y a las personas que la hacen mi segundo hogar. Ha sido un honor para mí haber encontrado el tesoro más grande que uno puede hallar: la plena felicidad. Y gran parte de ella se la debo a mi “facu”.

A mis compañeros, por compartir tantas cosas. En especial a Alina, Carlos, Claudio y Javier, a los que extrañaré enormemente y lamentaré perderlos como compañeros pero más disfrutaré de tenerlos como amigos.

Finalmente a Laura, a quien le he dedicado este trabajo y mi vida, por ser el sostén de mi peor yo, y por ser quien provoca a mi yo mejor. Sin ella, nada de esto tendría sentido.

ABREVIATURAS

Las abreviaturas y definiciones que aquí se presentan pueden consultarse y ampliarse en http://whatis.techtarget.com/definitionsAlpha/

[DBMS] *Database Management System*.

[DNS] *Domain Name Service*.

[FTP] *File Transfer Protocol*.

[HLD] *Hardware Load-balancing Device*.

[HTML] *Hyper-Text Markup Language*.

[HTTP] *Hyper-Text Transfer Protocol*.

[IMAP] *Internet Message Access Protocol*.

[LAN] *Local Area Network*.

[LRU] *Least Recently Used*.

[LAN] *Local Area Network*.

[MAN] *Metropolitan Area Network*.

[P2P] *Peer-to-peer.*

[POP] *Post Office Protocol*.

[RRDNS] *Round Robin DNS*.

[SHTTP] *Secure HTTP*.

[SMTP] *Simple Mail Transfer Protocol*.

[SSL] *Secure Sockets Layer*.

[TCP] *Transmission Control Protocol*.

[TCP/IP] *Transmission Control Protocol/Internet Procotol*.

[TTL] *Time-To-Live*.

[UDP] *User Datagram Protocol*.

[URL] *Uniform Resource Locator*.

[VRID] *Virtual Router IDentifier*.

[VRRP] *Virtual Router Redundancy Protocol*.

[VRRPd] Virtual Router Redundancy Protocol daemon.

[WAN] Wide Area Network.

[WWW] World Wide Web.

PARTE I

CAPITULO 1
TEORIA DE LA EVOLUCION

> *El problema de la distribución eficiente de carga para lograr la utilización óptima de los recursos computacionales de un conjunto de servidores interconectados mediante una red de computadoras, o lo que comúnmente denominamos balance de carga, es considerado uno de los temas más interesantes dentro del área de estudio de los sistemas distribuidos.*
>
> Andrew S. Tanembaum

Desde las compañías productoras de hardware se han propuesto soluciones al problema de la distribución de carga (ver *Estado del Arte*, pág. 25-26). Algunas con altos costos y otras muy complejas de implementar y que difícilmente se adaptan a los sistemas ya existentes o plantean nuevos inconvenientes, y terminan no siendo soluciones.

También la industria del software ha propuesto soluciones (ver *Estado del Arte*, pág. 25-26). Se trata, usualmente, de algoritmos estadísticos que estudian el estado de los servidores y determinan cuál de ellos tiene mejor capacidad para resolver una solicitud de servicio dada, en un determinado momento. Lo que, evidentemente, requiere de un agente que se encargue de obtener el estado general de sistema, tarea no

menor, que puede involucrar más tiempo del razonable para la atención del servicio. A esto se le suma otro inconveniente: después de un cierto tiempo incluso mínimo, cuando se tiene el estado general de todos los servidores, ¿aún seguirá siendo el mismo? Lamentablemente no podemos afirmarlo.

Un desacierto, a mi criterio, ha sido buscar una solución "personalizada" para los distintos servicios como si en el fondo éstos no fueran lo mismo. Por ejemplo, existen balanceadores de carga para servidores web y por separado para servidores de correo electrónico. Ha habido intentos, pero aún no se ha propuesto una solución con "bombos y platillos" para un servicio genérico, si este puede ser balanceado.

Sin embargo, es posible hallar esa solución. Y más aún si tratamos con servicios replicables. Un servicio es replicable, si se puede implementar un conjunto de servidores que brinden este servicio –por lo que además se requiere manejen el mismo contenido[1]- y así, independientemente de a cual se acceda, se obtenga el mismo resultado. Por ello, podemos afirmar que los servicios replicables son susceptibles de ser balanceados. A partir de ello, dejo planteada la tarea: construir una solución para balancear servicios replicables.

Partiendo de este concepto, si se implementa un agente o *broker* para distribuir la carga siguiendo algún criterio, independientemente del sistema operativo subyacente y del servidor que atenderá el servicio, mediante algún algoritmo eficiente de carga compartida, se habrá llegado a una alternativa más simple, genérica, transparente y poco acoplada en todo sentido. En esta dirección, existe un trabajo previo de Ulric Eriksson llamado *pen* [4], descontinuado en 2004 pero con actualizaciones de terceros hasta 2007, que se ha propuesto como una solución inicial a este problema y la cual usaré como base para este proyecto.

[1] En este sentido, comúnmente se utiliza NFS o *rsync*.

Desde los sistemas centralizados a los distribuidos

En el comienzo de la era de la computación, las computadoras eran grandes y caras. La mayor parte de las organizaciones tenía tan solo un puñado de computadoras y por carecer de una forma para conectarlas, estas operaban por lo general en forma independiente entre sí. [5] [6]

A partir de la mitad de la década del 80, dos avances tecnológicos comenzaron a cambiar esta situación. El primero fue el desarrollo de poderosos microprocesadores con el poder de cómputo de una computadora mainframe de tamaño respetable, pero por una fracción de su precio y tamaño. El segundo desarrollo fue la invención de las LAN. Las redes de área local permiten conectar docenas e incluso cientos de máquinas dentro de un edificio, de tal forma que se puede transferir información entre ellas en tiempos muy menores. Estos dos hitos marcaron el comienzo de la era moderna de la computación. [5]

No faltaba mucho para la aparición de los sistemas distribuidos, que son una realidad inherente a la distribución física de las organizaciones. Un sistema distribuido se forma mediante la interconexión de un conjunto de computadoras autónomas, las cuales soportan el almacenamiento de datos y la ejecución de procesos que interactúan con un fin común.

Los microprocesadores de 8 bits pasaron a ser de 16, 32 hasta 64 bits en la actualidad. Y hoy incluso, las computadoras cuentan con más de uno. Es más, su costo ha hecho esta tecnología accesible y popular. Sumado a esto, las redes de área amplia (WAN) permiten que millones de máquinas en toda la Tierra se conecten con velocidades que van de los Kbps. a los Gbps.

De la evolución vertiginosa de la computación en las últimas décadas resulta que hoy en día no sólo es posible, sino fácil, reunir sistemas de cómputo compuestos por un gran número de CPUs, conectados

mediante una red de alta velocidad. Estos sistemas categorizados con el nombre genérico de sistemas distribuidos contrastan con los sistemas centralizados anteriores. La problemática aparece ahora en el software, ya que los sistemas distribuidos requieren un software radicalmente distinto al de los centralizados. En particular los sistemas operativos distribuidos están apenas en la etapa de surgimiento. [6]

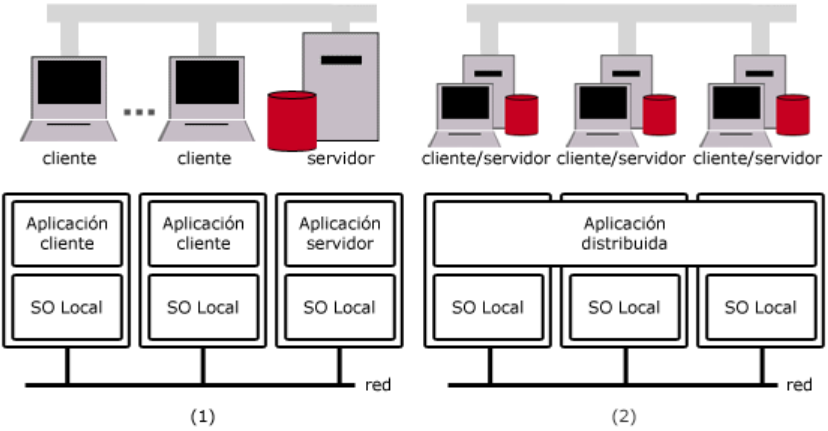


Figura 1. Sistemas centralizados (1), y no centralizados (2)

Como muestra la figura 1, la diferencia entre los sistemas centralizados y los no centralizados es bastante evidente: en los primeros (centralizados), el funcionamiento completo del conjunto depende de un único servidor, y el rol de cada elemento está muy marcado, mientras que en el segundo caso (distribuidos), los roles pueden compartirse, y un cliente podría ser además servidor de otro, así como las aplicaciones podrían estar totalmente diseminadas en los nodos pero con la apariencia (transparente) de estar en uno único. Sin embargo el contraste entre los sistemas de red y los verdaderamente distribuidos puede resultar mucho más sutil [5]. He aquí una clasificación extra [6] que puede clarificar los conceptos:

- Un sistema centralizado concentra todos los servicios en un servidor al que acceden todos los usuarios.
- Un sistema de red típico, es una colección de sistemas operativos locales, acompañado de servidores (por ejemplo de impresión o de archivos), conectados por medio de una red.
- Un sistema distribuido es un sistema cuyos componentes están expandidos en toda la red, pero visto como un sistema único para todos los usuarios del mismo. En éstos, a diferencia de los anteriores, también la lógica funcional del sistema podría estar dispersa entre los distintos componentes y uno o más procesos emplean la red como medio de comunicación e interacción.

El auge de las granjas de servidores

Es evidente que la digitalización e informatización de la información en todos los ámbitos plantea la necesidad de mayores capacidades computacionales, no solo de almacenamiento sino también de procesamiento.

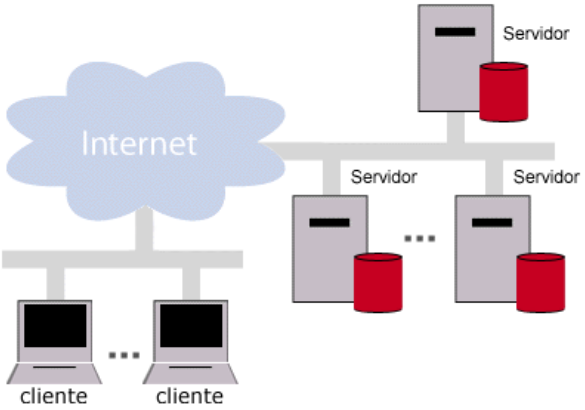


Figura 2. Una granja de servidores

La tecnología ha probado resolver los problemas de almacenamiento de formas muy económicas y eficaces. Hoy no es sorprendente hablar de unidades de almacenamiento con capacidades del orden del terabyte o el petabyte. Sin embargo para el segundo problema, el procesamiento, no hay mucho camino andado aún. Aquí es donde esta propuesta puede echar un poco de luz.

La mayoría de los proveedores de servicios de datos (web, email, ftp, etc.) han comenzado a migrar las soluciones implementadas en servidores centralizados, que requieren de enormes capacidades y que por lo tanto son antieconómicos, a soluciones basadas en varios servidores con menores capacidades individuales pero con similares capacidades en conjunto, distribuyendo la carga entre ellos, descentralizando las peticiones, aumentando el rendimiento (evitando cuellos de botella) y apuntando a la alta disponibilidad.

Un sistema bajo el enfoque de carga compartida aplicado en ambientes de hardware y software débilmente acoplados y heterogéneos, es comúnmente conocido como *farm* o granja de servidores (ver figura 2). Una granja de servidores es un grupo de computadoras que actúan como servidores que están alojados en conjunto y en una misma ubicación no solo física sino lógica. A menudo se usa el término *cluster de servidores* como sinónimo.

Las granjas permiten crear potentes servidores virtuales a base de unir varios servidores físicos, con ciertas ventajas:

- Transparencia de acceso (una misma dirección IP virtual para todos)
- Replicación (alta disponibilidad mediante redundancia)
- Balance de carga (mejora el rendimiento y evita los cuellos de botella)
- Alta disponibilidad (tolerancia a fallos mediante comprobación activa de disponibilidad)

Tendencias actuales

Seguramente las nuevas supercomputadoras ya no serán contundentes mainframes que ocupen habitaciones completas, sin posibilidad de actualización y cuyo costo es digno de inventar una nueva moneda para evitar los excesivos ceros. Por el contrario, serán clusters formados de miles de computadoras, incluso, físicamente distribuidas alrededor del globo. La superioridad para ese entonces, la tendrá quien logre no solo agrupar la mayor cantidad de nodos, sino hacer que éstos se integren de tal forma que cooperen hasta para optimizar el consumo de energía.

Si bien con la aparición de las computadoras multiprocesador, los problemas propios de la interconexión (retrasos de red, limitaciones de conectividad, fallos de comunicación), y los problemas de energía y refrigeración tienden a desaparecer, mejorar el rendimiento pasará por otras cuestiones inherentes a la distribución de la carga de trabajo en el cluster, y será entonces cuando los problemas que plantea este trabajo, y fundamentalmente las soluciones, serán el factor limitante.

El caso más resonante es el de Google (http://www.google.com), que se ha posicionado como el buscador más utilizado por su velocidad y alta disponibilidad debido al uso de clusters para brindar el servicio.

No falta mucho para que nuestras propias computadoras personales integren clusters. Incluso lo hacen, tal vez sin que nos demos cuenta, cuando usamos sistemas de intercambio de archivos P2P, como eDonkey o KaZaa, que han popularizado las redes formadas de computadoras hogareñas para compartir grandes volúmenes de datos. Y llegaremos más allá. No será extraño en un futuro cercano, correr aplicaciones distribuidas de forma remota, con una potencia de cálculo y procesamiento formado por cientos de procesadores en un cluster. Es solo cuestión de tiempo.

CAPITULO 2
DIVIDE Y VENCERÁS

> *Divide et vinces (divide y vencerás) nos conduce a una buena estrategia para resolver problemas que se puedan dividir en sub-problemas más sencillos.*
>
> *Julio César*

En la cultura popular, divide y vencerás hace referencia a un refrán que implica resolver un problema difícil, dividiéndolo en partes más simples tantas veces como sea necesario, hasta que la resolución de las partes se torna obvia. La solución del problema principal se construye con las soluciones encontradas.

El término, en su acepción más amplia es algo más que una técnica de diseño de algoritmos. De hecho, suele ser considerada una filosofía general para resolver problemas y de aquí que su nombre no sólo forma parte del vocabulario informático, sino que también se utiliza en muchos otros ámbitos.

En nuestro contexto, divide y vencerás se aplica al hecho de sacar ventaja dividiendo la carga de trabajo entre un conjunto de servidores, para aumentar el rendimiento total, eliminar un cuello de botella, mejorar significativamente la disponibilidad y, por ende, lograr casi de manera gratuita la tan buscada tolerancia a fallos.

Balance de carga, carga compartida y distribución de carga.

El balance de carga en servidores es una técnica de bajo costo y fácil implementación, que permite aumentar el rendimiento general de los sistemas. Supongamos que hay un solo servidor respondiendo a todos los requerimientos HTTP entrantes para un sitio web. Podría suceder que, de pronto, el sitio gane popularidad, y que la capacidad del servidor no pueda atender tanto volumen de tráfico. A medida que esto sucede, las páginas cargarán cada vez más lentamente y los usuarios tendrán que esperar más tiempo a que el servidor se libere para atender sus pedidos. Este incremento de tráfico y conexiones puede llevar a un punto en que actualizar el hardware no será ni efectivo ni costeable. Esta situación se puede repetir para servicios FTP, STMP, POP, etc.

Para lograr la escalabilidad de un servicio, se pueden agregar más servidores y distribuir la carga entre ellos. La distribución de carga entre estos servidores es conocida como balance de carga. El balance o balanceo de carga es un concepto usado en informática que se refiere a la técnica usada para compartir el trabajo a realizar entre varios procesos, servidores, discos u otros recursos. Está íntimamente ligado a los sistemas de multiprocesamiento, o que hacen uso de más de una unidad de procesamiento para realizar labores útiles.

El balance de carga se mantiene gracias a un algoritmo [7] que divide de la manera más equitativa posible el trabajo, evitando los cuellos de botella, que es el principal objetivo del multiprocesamiento. En el CD-ROM que acompaña a este trabajo se incluye documentación descriptiva de los algoritmos más comúnmente utilizados.

En este proceso, los servidores que colaboran para brindar el servicio, deben aparecer como un único servidor al cliente. El mecanismo, comúnmente recibe el nombre de *Network dispatch* o *IP Spraying*. El dispositivo encargado del balance es llamado *load dispatcher* o simplemente balanceador de carga (LBS).

Como ejemplo tomemos un *web farm*. El balanceador interceptará los requisitos HTTP, y los redirigirá a algún servidor dentro del cluster. Dependiendo del mecanismo de distribución implementada, la arquitectura puede proveer escalabilidad, balance de carga, y tolerancia a fallos.

Ahora bien, se debe asumir que pasar de un servicio centralizado a un cluster de servidores no será automático y sobre todo no será gratis en términos de trabajo de adaptación. Más aún, para ser exitosa, la forma en la que se implemente la distribución y los algoritmos de balance de carga elegidos, resultarán de principal importancia en este proceso de migración, pues será la base del funcionamiento del sistema distribuido. De otra forma, tales sistemas no tendrían ningún sentido.

En este punto es una buena idea hacer un paréntesis para aclarar algunos términos: la distribución de carga se refiere a una técnica para repartir la carga de trabajo entre un conjunto de *workers* (los servidores pares que componen la granja o cluster). Este proceso se hace de forma dinámica, y puede llevarse a cabo mediante balance de carga o carga compartida. Es sutil, pero hay una diferencia entre estos términos.

La carga compartida se produce interceptando una solicitud de servicio, seleccionando según un cierto algoritmo al worker más adecuado para atenderlo, y redirigiendo la solicitud hacia él. El balance de carga, es un caso particular de distribución, que intenta mantener a todos los workers con el mismo grado de carga; he ahí el verdadero balance. El procedimiento se inicia como en la carga compartida, pero el balance se logra, por ejemplo, permitiendo que los workers se comuniquen entre sí y se distribuyan la carga entre ellos bajo demanda y en cualquier momento de la ejecución.

Más allá de la diferencia semántica de los términos y por cuestiones de simplificación de conceptos, se tratarán en adelante como términos intercambiables, sin embargo el significado que se pretende será el del primero (carga compartida). Y esto es fácilmente justificable: el balance

de carga (en el sentido más estricto de balance), usualmente se abandona en pos del aumento de performance, así cayendo en algoritmos de distribución de carga por carga compartida, que son menos exhaustivos y más sencillos de implementar.

Es tan importante el rol que juega el algoritmo de distribución de carga, que es en este punto en donde muchas de las soluciones existentes fallan. Generalmente, los algoritmos que se utilizan, se basan en una actividad denominada *polling*. El polling consiste de una toma, a intervalos muy estrechos de tiempo, de un cierto valor. En este caso, este valor es el *estado de carga* de un servidor dado. Este procedimiento debe hacerse para todos los workers que integran el cluster. Y la escalabilidad, al tacho.

Entre las desventajas del polling, la más evidente tiene que ver con el tiempo que toma completar la obtención del estado general del sistema. Pero, menos evidente e indirecto aún, está el siguiente inconveniente: luego de haberlo obtenido, ¿el estado individual de los servidores se mantiene? Indudablemente, el grado de actualidad del estado dependerá del tiempo en tomarlo. Generalmente, este tiempo no es menor. De ahí que me aventuro en afirmar que se pueden usar algoritmos más eficientes para la distribución de carga alternativos a los basados en polling.

Soluciones tempranas

El balance de carga de los servidores mediante un IP sprayer se puede implementar de varias maneras [8]. Por ejemplo, a través de un DNS, un servidor proxy, un dispositivo hardware o un servidor específico con un software para tal fin. También son varias las alternativas de algoritmos para lograr este balance. Desde los más tempranos a los más actuales, podemos encontrar al *random allocation*, *round-robin allocation*, *weighted allocation* y los algoritmos basados en colas.

Random Allocation. La selección aleatoria puede resultar la solución más natural: las solicitudes de servicio son asignadas a cualquier servidor elegido al azar de entre un conjunto de servidores. En tal caso, a uno de los servidores podrían serle asignados varios requerimientos, mientras que otros podrían permanecer ociosos. Sin embargo, por la probabilidad de los grandes números, se puede aventuradamente pronosticar que con el paso del tiempo y a lo largo de los múltiples requerimientos, todos recibirán la misma cantidad de solicitudes de servicio. Claramente, a la ventaja de su sencilla implementación se opone la desventaja de la posible situación de sobrecarga de algún servidor cuando otros están relativamente poco cargados (figura 3).

Este tipo de algoritmo se ha usado en dispositivos hardware de ruteo para ofrecer balance de carga de una forma muy económica.

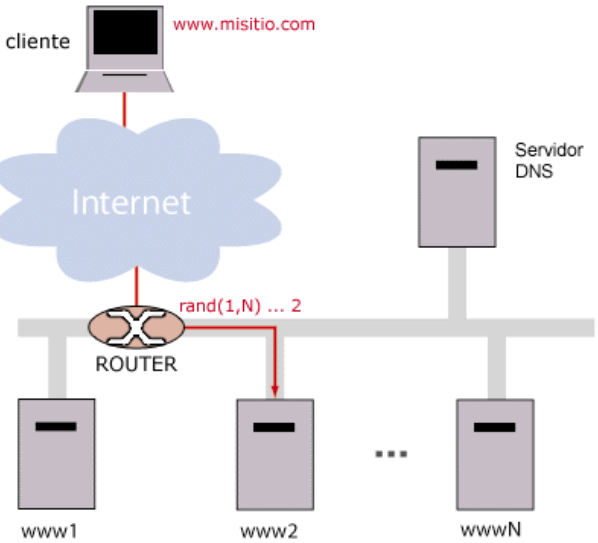


Figura 3. Random Allocation Router

Round Robin allocation. Otras soluciones tempranas de balance de carga se han implementado a través de un servidor DNS que reparte los requerimientos siguiendo un algoritmo *Round Robin* (figura 4). Esta técnica es conocida como Round Robin DNS (RRDNS), y funciona como se detalla a continuación.

Supongamos que existen n servidores $S_0, S_1, \ldots, S_{n-1}$ con $n>0$ en donde el valor de i indica el último servidor seleccionado inicializado en -1. El algoritmo es como se muestra en el listado 1:

```
j = i;
do {
    j = (j + 1) mod n;
    if (Available(Sj)) {
        i = j;
        return Si;
    }
} while (j != i);
return NULL;
```

Listado 1. Algoritmo round-robin

Para un conjunto de n servidores web *www*1,… *www*N, se puede implementar un servidor RRDNS para redirigir las solicitudes HTTP correspondientes a un dominio www.misitio.com hacia los n servidores, de forma tal, que el balance se logra a través de un truco en la traducción URL a IP.

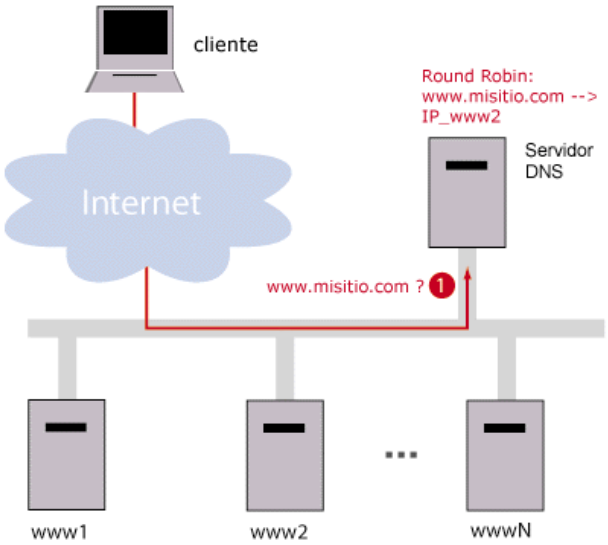


Figura 4. Round Robin DNS (1)

1) La figura 4 muestra un caso típico de requerimiento HTTP a un servidor web. El cliente escribe la URL del sitio en un navegador y para ser direccionada deberá traducirse en una dirección IP. Esta tarea es realizada por un DNS, al que el sistema operativo del cliente consulta para tal fin.

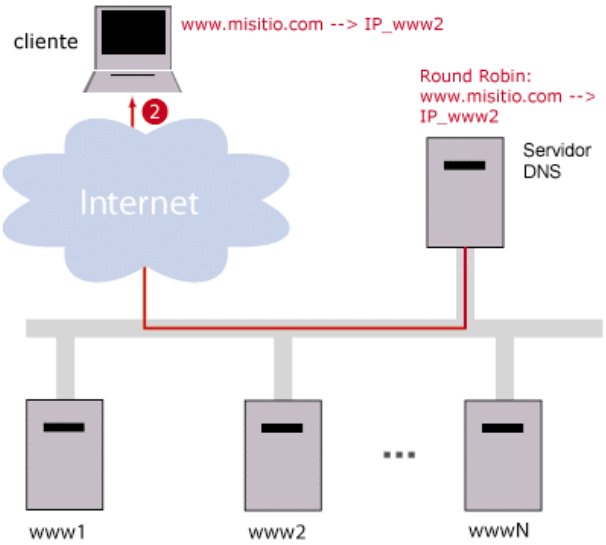


Figura 5. Round-Robin DNS (2)

2) La figura 5 muestra como el servidor DNS (en este caso un RRDNS) recibe la solicitud de traducción y siguiendo el algoritmo round robin (uno a uno y de forma cíclica) selecciona y retorna como respuesta la dirección IP de alguno de los servidores del cluster para atender al servicio solicitado.

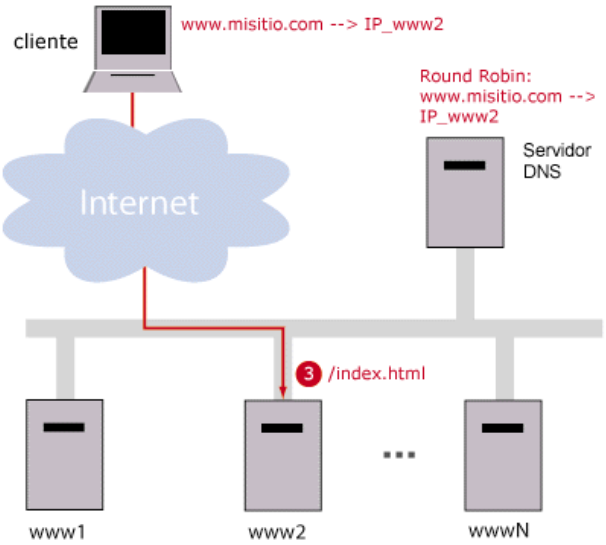


Figura 6. Round-Robin DNS (3)

3) En la figura 6 puede verse cómo el cliente recibe la dirección IP y accede al servidor asignado de manera transparente, asumiendo que ese servidor tiene la dirección IP que corresponde a la URL solicitada.

Así, las solicitudes llegarán al RRDNS y serán asignadas de manera aleatoria a los servidores del cluster. He aquí el balance de la carga.

La forma en la que se implementa un RRDNS es muy sencilla y básicamente consiste en configurar adecuadamente un servidor DNS para tal fin. Para el ejemplo anterior, se deberá:

a) configurar el archivo named.conf

```
;; named.conf
;;type    domain           source-file
primary   misitio.com      db.misitio
primary   rr.misitio.com   db.misitio.rr
```

Listado 2. Archivo /var/named/named.conf

b) configurar el archivo de zona para el dominio misitio.com.

```
;; db.misitio
IN SOA misitio.com. root.misitio.com. (
     1998021502 ; SERIAL
     604800     ; REFRESH: 1 week
     3600       ; RETRY:   1 hour
     604800     ; EXPIRE:  Maximum TTL is 1 week
     86400      ; MINTTL:  Minimum TTL is 1 day
)
IN NS ns.misitio.com.
;;
;;  the resource record for www.foo.dom which
;;  maps to the Round-Robin domain
;;
www   IN  CNAME   www.rr.misitio.com.
```

Listado 3. Archivo /var/named/db.misitio

c) configurar el archivo de zona para el dominio rr.misitio.com.

```
;; db.misitio.rr
;;
IN SOA rr.misitio.com. root.rr.misitio.com. (
     1998021502 ; SERIAL
     3600       ; REFRESH: 1 hour
     600        ; RETRY:   10 minutes
```

```
     3600       ; EXPIRE:  Maximum TTL is 1 hour
     1800       ; MINTTL:  Minimum TTL is 30 min
)
IN NS ns.misitio.com.
;;
;;  multiple CNAME resource record
;;  for BIND's Round-Robin (RR) feature
;;
www      IN  CNAME   www1.rr.misitio.com.
         IN  CNAME   www2.rr.misitio.com.
         IN  CNAME   www3.rr.misitio.com.
         IN  CNAME   www4.rr.misitio.com.
         IN  CNAME   www5.rr.misitio.com.
;;
;;  the address (A) resource records for the
;;  final NAME -> IP mapping
;;
www1     IN  A       192.168.1.1
www2     IN  A       192.168.1.2
www3     IN  A       192.168.1.3
www4     IN  A       192.168.1.4
www5     IN  A       192.168.1.5
```

Listado 4. Archivo /var/named/db.misitio.rr

Un RRDNS tiene como ventaja la transparencia que ofrece a clientes y servidores. Además interviene muy poco en todo el proceso: sólo se consulta al comienzo de la operación. Desafortunadamente, no siempre funciona adecuadamente, ya que los DNS intermedios y el software de los clientes (navegadores), suelen almacenar en cache las direcciones IP retornadas por el DNS ignorando el valor TTL (time-to-live) y usando siempre la misma ruta al servidor una vez obtenida, salteando al balanceador y por lo que el proceso de balanceo fracasa.

Proxy en Reversa. Soluciones posteriores apuntaron a un servidor Proxy configurado “en reversa”. La idea consiste en utilizar un servidor proxy que opera en la dirección contraria a su uso normal. El proxy se hace pasar como el servidor final (www.misitio.com) y traduce el URL

relativo recibido en un URL absoluto dirigido a uno de los servidores en el cluster.

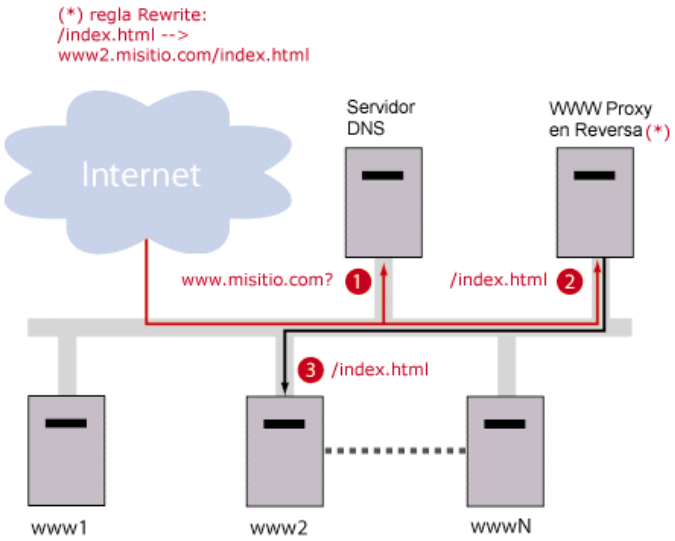


Figura 7. Proxy en Reversa

En la figura 7 se puede observar como el proxy en reversa aparece como el servidor final para los clientes, pero en vez de servir la solicitud él mismo, determina el servidor que va a responder, envía la solicitud y después encamina la respuesta. Además, como se muestra en la figura 8 y con una modificación sencilla, los servidores en el cluster pueden enmascararse en una subred detrás del servidor proxy en reversa y no se hacen necesarios trucos con el DNS.

La forma en la que se implementa un proxy en reversa con Apache (desde 1.3.0) es también bastante sencilla. Básicamente consiste en:

a) aplicar un parche para habilitar las características del proxy en reversa (el archivo se incluye en el CD-ROM que acompaña este proyecto, bajo el nombre apache-rproxy.mk)

b) especificar separadamente los servidores de contenido dinámico de los de contenido estático en el archivo de configuración:

```
# apache-rproxy.conf-servers
#
# servers for static pages
static www1.misitio.com|www2.misitio.com|www3...
# servers for dynamic pages
dynamic www4.misitio.com|www5.misitio.com
```

Listado 5. Archivo apache-rproxy.conf-servers

c) configurar apache para que actúe como proxy en reversa (el archivo se incluye en el CD-ROM que acompaña este proyecto, como apache-rproxy.conf)

```
# apache-rproxy.conf
…
RewriteMap server rnd:/path/to/apache-rproxy
RewriteRule ^/rproxy-status.*      -    [L]
RewriteRule ^(http|ftp)://.*       -    [F]
RewriteRule ^/(.*\.(cgi|shtml))$ to://${server:dy
RewriteRule ^/(.*)$              to://${server:st
RewriteRule ^to://([^/]+)/(.*) http://$1/$2 [E=SE
RewriteRule .*                     -    [F]
```

Listado 6. Archivo apache-rproxy.conf

En el Apéndice II se incluyen los cálculos necesarios para configurar el número de servidores (NOS) y el tamaño óptimo de memoria RAM (MBR), que se referencian en el archivo apache-rproxy.conf

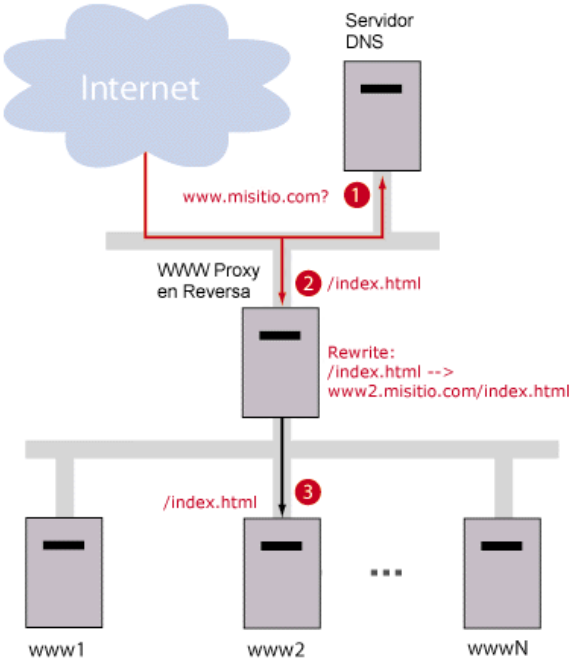


Figura 8. Proxy en reversa delante de una subred de servidores

La desventaja de este enfoque está dada por el cuello de botella, en este caso, en el servidor proxy. Más aún, el rendimiento de un proxy en reversa en general es inferior al de un RRDNS.

Un proyecto que hace uso de proxy en reversa es Resin, de Caucho Technology. Se puede acceder a la documentación y descargas en el sitio de Caucho Tech. http://www.caucho.com/resin-3.1/index.xtp

Load Balancing Servers. Los intentos posteriores sobre el tema, han sido destinados a eliminar las limitaciones en las propuestas anteriores. Para ello, se propone colocar por separado un servidor para balance de carga (LBS) que implementa algoritmos de balance de carga más eficientes, delante de un cluster de servidores.

Un enfoque de balance mediante LBS (ilustrado en la figura 9) funciona de esta manera: 1) los requerimientos de resolución de nombres son resueltos por un DNS, 2) los clientes hacen pedidos al LBS (a través de la IP devuelta por el DNS), y 3) el LBS selecciona al servidor para atender al pedido y redirige la solicitud de forma transparente.

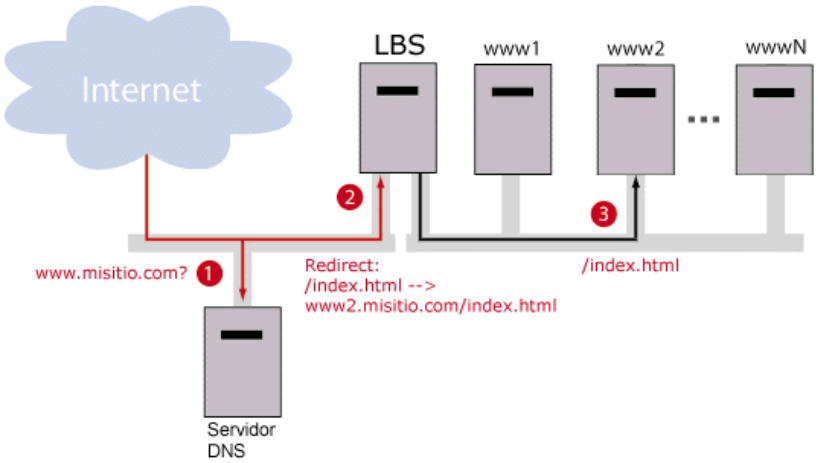


Figura 9. Load Balancing Server

Un LBS permite que la granja tenga la apariencia de una única IP, eliminando cualquier dependencia del DNS para lograr el balance de carga. Esta dirección IP es denominada comúnmente como “la dirección IP del clúster”. Los clientes y aplicaciones hacen los requerimientos a

esta dirección IP y una vez que el LBS recibe un requerimiento, hay varias formas de encaminarlo al servidor apropiado. La forma más simple de hacerlo es utilizando la función de redirección del HTTP http-redirect. [9]

Respecto del algoritmo de balance de carga que implementa un LBS, comúnmente se utiliza alguna variación del Round Robin más equitativa y que tenga en cuenta las capacidades individuales de los servidores en el cluster.

Weighted allocation. El algoritmo Round Robin con pesos es una versión avanzada de Round Robin, que elimina algunas de sus deficiencias. Propone asignar un valor de peso a cada servidor en el grupo, de tal forma que si un servidor es capaz de manejar el doble de carga que otro, este servidor más “poderoso” obtenga un valor de peso igual a 2, mientras que el otro, el más “débil”, obtenga un valor de peso igual a la unidad. El LBS, en esa situación, asignará dos requerimientos al servidor con más capacidades por cada requerimiento asignado al de menor potencia. Sin embargo, como desventaja, el algoritmo no considera los tiempos de procesamiento de los requerimientos individuales.

Estado del arte

En las últimas dos décadas se ha venido usado hardware muy contundente respecto de capacidades de cómputo y procesamiento para las aplicaciones paralelas. Sin embargo, últimamente éste ha sido reemplazado por clusters de computadoras que son más económicos y a menudo más poderosos.

Con esto se plantean nuevos desafíos. El más interesante, a mi criterio, es el modo en que la carga de trabajo es repartida entre los distintos elementos de procesamiento dentro de un cluster. Más interesante resulta aún, cuando consideramos que los elementos que conforman un cluster son tan diversos como las personas que conforman un grupo. He aquí la clave, la heterogeneidad.

Usualmente, los algoritmos de distribución de carga para sistemas heterogéneos son tratados como "estrategias centralizadas", en el sentido de que un procesador maestro mantiene información sobre todos los elementos de la red y toma todas las decisiones necesarias. Esto a veces es insuficiente debido al overhead que implica la actualización de esta información del estado global.

Otros enfoques usan políticas distribuidas, considerando propiedades de los sistemas que en la mayoría de los casos son estáticos (por ejemplo, basados en benchmarks). Otros consideran parámetros dinámicos de carga, como por ejemplo, la cantidad de procesos en la cola. [10]

Por nombrar alguna de las soluciones y proyectos existentes respecto de la problemática de balance de carga, encontramos:

- Linux Virtual Server Project: es quizás el más avanzado de los proyectos de servidor distribuido, es opensource e inicialmente sólo funciona para sistemas operativos Linux. Es una suite completa de herramientas de manejo y configuración de servidores. http://www.linuxvirtualserver.org/

- Eddie: es una herramienta para clusters de alta disponibilidad. Es opensource y promete ser una suite de administración automática de tráfico y configuración para un conjunto de sitios servidores distribuidos geográficamente. http://eddie.sourceforge.net/

- Piranha: Es un sistema para clusters que sólo soporta balance de carga en servicios web y ftp. Aunque está disponible para Linux únicamente, se han hecho algunos esfuerzos para portarlo a FreeBSD. http://freshmeat.net/projects/piranha/

- DC-Apache v1.0: propone desde su nombre (Distributed Cooperative Apache Web Server) ser una solución escalable para servicio web. Este módulo permite configurar varios servidores web Apache para que colaboren distribuyendo carga. http://www.cs.arizona.edu/dc-apache/

- XLB HTTP Load Balancer: es un balanceador de carga de alta performance opensource. http://sourceforge.net/projects/xlb

La industria del hardware también ha incursionado en soluciones embebidas para balance de carga de servidores, conocidas como *HLD* (Hardware Load-Balancing device). Respecto de este tópico se pueden consultar en internet (entre otros), los sitios web de:

- Local Director de Cisco Systems, en http://www.cisco.com/univercd/cc/td/doc/prod_cat/pclocald.htm

- Equalizer de Coyote Point Systems, en http://www.coyotepoint.com/equalizer.shtml

- 3Com, en http://lat.3com.com/

- HP, en http://www.hp.com/

Para concluir este capítulo, es necesario decir que la configuración de un software o hardware para balance de carga debe decidirse teniendo en cuenta el requerimiento en particular. Por ejemplo, si lo que se busca es balancear la carga de un servidor web de páginas HTML estáticas o

dinámicas pero livianas, los algoritmos Round Robin serán suficientes. En otros casos será necesario un algoritmo más eficiente. Pero más allá de lo importante que resulte la elección de uno u otro algoritmo –y he aquí el aporte de este trabajo– está la generalidad con la que debe tratarse el problema: primero, por diferentes que resulten los servicios que se pretendan balancear, existe una similitud fundamental entre ellos que permite una misma solución (intentaré demostrar esto en el capítulo siguiente) y segundo, la heterogeneidad que debe contemplar la solución no sólo corresponde al hardware de los distintos elementos del cluster, sino también al software base y al software del servidor que da soporte al servicio (o los servicios) a balancear.

CAPITULO 3
EN BUSCA DEL BROKER PERDIDO

> *Me doy paso hacia la búsqueda, pase lo que pase, será una de mis grandes promesas. Aunque el miedo me invada a veces es parte de esta historia, en la que estoy seguro de encontrar muchas más preguntas que respuestas, y donde hallaré aquellas inesperadas aventuras que me ayudarán a encontrar mi propio ser.*
>
> Anónimo

Al final del capítulo anterior se hizo referencia a un concepto interesante el cual me permito recordar aquí: "*más allá de lo importante que resulte la elección de uno u otro algoritmo está la generalidad con la que debe tratarse el problema: primero, por diferentes que resulten los servicios que se pretende balancear, existe una similitud fundamental entre ellos que permite una misma solución y segundo, la heterogeneidad que debe contemplar la solución no solo corresponde al hardware de los distintos elementos del cluster, sino también al software base y al software del servidor que da soporte al servicio*".

¿Existe tal similitud? Claro que sí. Aunque a veces, lo esencial sea invisible a los ojos.

ublobs

ublobs es acrónimo de Universal Broker for Load-Balancing Services y nace como resultado de la investigación de la problemática tratada en este proyecto, respecto del balance de carga en aplicaciones distribuidas en ambientes heterogéneos y poco (o débilmente) acoplados, o más comúnmente conocidos hoy como (pues así han sido popularizados) clusters, o granjas de servidores.

La idea detrás de ublobs, es la de presentar un balanceador de carga para servicios basados en el protocolo TCP. Esto cubre la mayoría de los servicios que pueden clasificarse como “replicables” o incluso “paralelizables”, como HTTP, SMTP, POP, IMAP, etc. dejando afuera otros servicios “no tan sencillos” como el de las bases de datos, que usan protocolos más complejos. Sin embargo, y es bueno subrayarlo, muchos DBMS, como es el caso de MySQL y PostgreSQL entre otros, funcionan perfectamente como ejemplos de servicios “no tan sencillos”, pero susceptibles de ser balanceados con ublobs.

Una de las principales premisas que ublobs persigue es la transparencia, en el sentido de permitir que un conjunto de servidores aparezcan como un único servidor hacia el exterior y además, detectar a los servidores cuando “se caen”, para distribuir la carga de los clientes solo entre aquellos que permanezcan disponibles. El primer hecho (enmascaramiento de un conjunto de servidores detrás de uno único) apunta a satisfacer la relación escalabilidad/performance, y el segundo (la detección de fallas) mueve a ublobs en la siguiente dirección: alta disponibilidad.

En relación a la importancia del algoritmo de balance o distribución de la carga de trabajo, ublobs implementa un algoritmo de distribución que rescata lo mejor de las alternativas mencionadas en el apartado *Soluciones Tempranas* del Capítulo 2.

En el CD-ROM que se adjunta con este trabajo, se incluye el código completo de ublobs con comentarios técnicos más puntuales respecto del algoritmo implementado. Sin embargo, a modo descriptivo y en forma sintética, el algoritmo mantiene una tabla de seguimiento que tratará de encaminar a un mismo cliente al último servidor que sirvió su requerimiento. Al llenarse (la tabla tiene un máximo de entradas especificado a través de un parámetro en la línea de comandos, por defecto 2048) se corre un algoritmo LRU para eliminar aquellas entradas menos recientemente usadas y hacer lugar a otras nuevas.

La elección de este tipo de algoritmo responde a la problemática presentada en el siguiente ejemplo: Un cliente accede a un sitio alojado en el server a través de su URL, supongamos http://www.misitio.com. El servidor recibe la solicitud y devuelve al cliente la página de inicio con un formato HTML. El navegador del cliente interpreta el código y si este incluye otros archivos (imágenes, sonido, videos, o incluso frames), al mismo tiempo que muestra la página, crea una conexión por cada uno y los solicita al servidor.

Claramente, al usar un balanceador que distribuya la carga por Round Robin, cada petición del cliente se enviaría a un servidor distinto. Esto no sería un problema si se tratara sólo con aplicaciones que no mantienen el estado de las conexiones en el servidor (por ejemplo un sitio estático), sin embargo una gran cantidad (por no decir la mayoría) de las aplicaciones web modernas mantienen “algo” de estado, por ejemplo, las sesiones web para mantener los datos de inicio de sesión de un usuario. Un balanceador Round Robin rompe este esquema de estado. Por otro lado, mantener una tabla para que las conexiones activas de un cliente sigan siendo hacia un mismo servidor, si bien atenta en alguna medida contra el balance, no tiene este inconveniente.

Otra cuestión es la tolerancia a fallos. ublobs también considera como premisa el hecho de que los dispositivos tienden a fallar, y las fallas, en tal caso, además de imprevisibles, son fatales. Por ejemplo, llevamos

descargado el 60% de una página web y el servidor deja de responder, por ejemplo, por problemas de sobre-temperatura. Si bien contemplar estos casos de falla es imposible, puede tomarse alguna medida en respuesta, como redirigir la solicitud a un servidor “réplica” continuando la descarga desde este otro. La forma de lograr esta tolerancia a fallos es también bastante sencilla: cuando ublobs detecta que un servidor no responde, lo considera “caído” y lo elimina de la lista de servidores disponibles para evitar redirigir nuevos pedidos hacia el. Las próximas solicitudes serán asignadas a un nuevo servidor. De este modo, se obtiene balance de carga con tolerancia a fallos “casi gratis”.

Incluso hay otras situaciones, además de las fallas, en las que se debe asegurar la alta disponibilidad de servicio aún cuando algún servidor individual dentro de la granja no este disponible: por ejemplo, cuando es apagado por mantenimiento o reconfiguración.

Otro punto de falla, es ublobs en sí mismo. Si el equipo en el que se ejecuta sufre una falla quedando no disponible, el sistema completo caería. Esta situación es típica de los servicios centralizados y a veces no es posible evitar caer en estos cuellos de botella. Afortunadamente, esto puede revertirse sin demasiado esfuerzo. Se hará a continuación una breve introducción a algunas ideas acerca de un protocolo de redundancia para alta disponibilidad que podría resultar útil en tal sentido.

Protocolo de redundancia para alta disponibilidad

VRRP especifica un protocolo de elección que dinámicamente otorga responsabilidad sobre una IP a un router virtual, de entre un conjunto de routers VRRP en una LAN. El router VRRP que tenga control sobre esta “dirección del cluster”, es llamado Master y su función es hacer reenvío

de los paquetes (forwarding) recibidos del exterior hacia adentro del cluster y viceversa (ver Apéndice I).

El proceso de elección provee tolerancia a fallos en la responsabilidad de forwarding cuando el router Master pasa a estar no disponible. Esto permite que cualquier otro router virtual en la LAN sea promovido a router principal, tomando el lugar del que ha fallado.

La ventaja que se saca de usar VRRP es una ruta por defecto de alta disponibilidad, sin requerir la configuración de enrutamiento dinámico, hardware adicional, o protocolos de descubrimiento en cada host. [11]

En resumen, VRRP es un protocolo que elige un servidor como "servidor maestro" en una LAN y le asigna una IP "virtual", a la que sólo el responderá. Si el servidor falla en algún momento, otro servidor de respaldo toma esa dirección IP y continúa con el trabajo. Esta virtualización a través de una dirección IP única para todos los nodos, permite que el proceso se realice de forma transparente hacia el exterior y como el protocolo trabaja con mensajes básicos UDP entre los servidores, el tiempo entre la detección de la "caída" del servidor maestro y en que un reemplazo toma su lugar ante la falla es mínimo y hasta será imperceptible.

Algunos ejemplos de proyectos y soluciones en este sentido son, por citar alguno:

- vrrpd: es una implementación de VRRPv2 como se especifica en la RFC 3768. Se ejecuta en Linux, como una aplicación en el espacio del usuario. Es un proyecto opensource mantenido por Jerome Etienne <jetienne@arobas.net>

- FreeVRRPd: otra implementación del protocolo, originalmente desarrollado para FreeBSD pero que luego ha sido portado a Unix/Linux. Documentación y descargas en el sitio del proyecto http://www.bsdshell.net/hut_fvrrpd.html

Usando entonces un protocolo de redundancia, la última cuestión que podría considerarse como punto de falla, el carácter centralizado de ublobs en sí mismo, estaría cubierta.

Un ejemplo que ilustra el comportamiento del protocolo de VRRP para asegurar alta disponibilidad es el mostrado en las figuras 10 a 13.

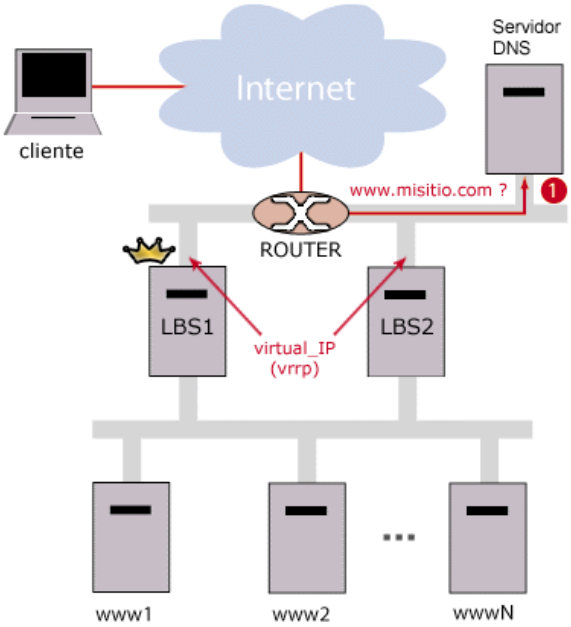


Figura 10. Protocolo de redundancia (1)

1) Un cliente solicita la resolución de nombre para un sitio cuya URL es www.misitio.com (figura 10)

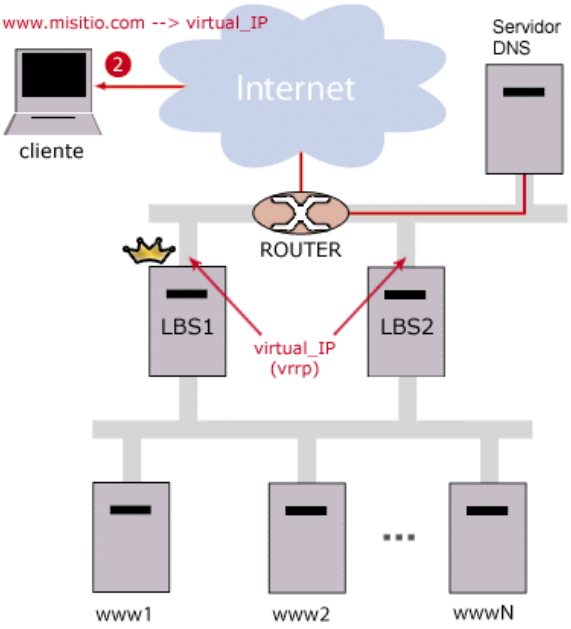


Figura 11. Protocolo de redundancia (2)

2) El servidor de nombres de dominio devuelve la dirección IP virtual del cluster. Suponiendo que todos los LBSs están disponibles y que el protocolo VRRP inicialmente eligió al LBS1 como Master, será LBS1 quien tendrá asignada esta dirección virtual_IP (figura 11).

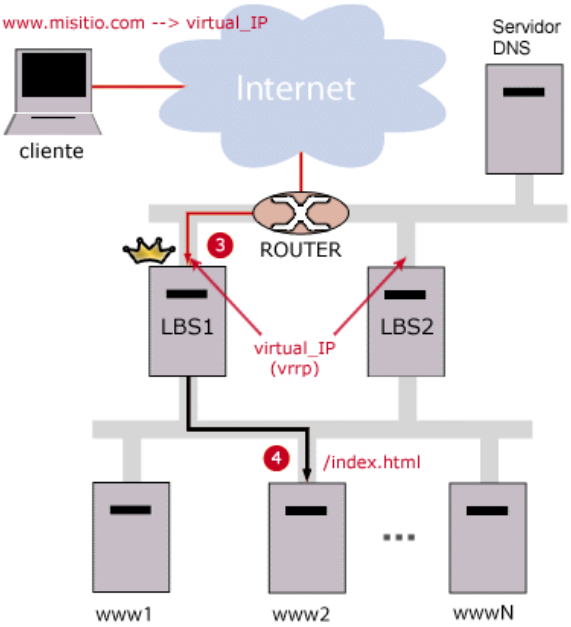


Figura 12. Protocolo de redundancia (3 y 4)

3) El cliente asocia la URL www.misitio.com a la dirección virtual_IP y envía el request,

4) el LBS1 recibe la solicitud y selecciona algún servidor, al cual redirigirá el pedido (figura 12).

Debido a que en la mayoría de los casos los servidores mantienen algún estado de la conexión, un cliente debiera (siempre que se pueda) seguir la misma ruta y ser atendido por el mismo servidor.

Propongo, a modo de ejemplo, suponer que en algún momento dado, el servidor LBS1 sobrepasa un cierto umbral de temperatura, se bloquea y queda fuera de alcance. La figura 13 ilustra esta situación.

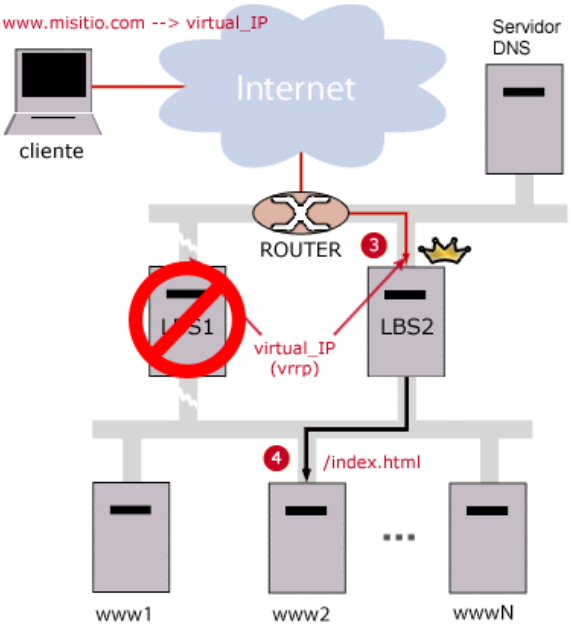


Figura 13. Protocolo de redundancia (3 y 4) cuando falla un LBS

5) El protocolo VRRP (también en ejecución en LBS2) detecta que LBS1 está caído y otorga a LBS2 la dirección virtual_IP, de modo que futuras peticiones serán atendidas por él, de forma transparente, como si lo siguiera haciendo LBS1 (figura 13).

En el CDROM que acompaña a este trabajo se incluye documentación de la RFC 3768 correspondiente al protocolo VRRPv2. La última versión del documento en línea se encuentra en http://tools.ietf.org/html/rfc3768.

En el Apéndice I se adjunta más información respecto de protocolos de redundancia, una guía de instalación y prueba de vrrpd, y se describen en profundidad los parámetros de línea de comandos.

El aporte de ublobs

El aporte real de ublobs está en lo simple que resulta implementarlo. En conjunto con VRRP, es una solución completa para balance de carga en un cluster de servidores de alta disponibilidad, transparente y que contempla tanto la heterogeneidad del hardware como la del software base y de servicio. Además, ublobs es una implementación desarrollada como software libre bajo licencia GPL, y es el resultado de una profunda investigación de la problemática.

Instalación y configuración

ublobs funciona tanto en *Unix/Linux* como en *Microsoft® Windows*. Para Unix/Linux se distribuye en formato tar.gz (Debian/Slackware) y como paquete rpm (Fedora/RedHat). Para Windows, como binario ejecutable (.exe) con sus respectivas librerías de portabilidad (cygwin.dll, etc.).

La instalación del archivo tar.gz no difiere del común. Se descomprime el archivo en algún directorio, y se ejecuta:

```
# tar -xjvf ublobs-0.1.tar.gz
# cd ublobs-0.1
# ./configure
# make && make install
```

Listado 7. Instalación de ublobs (tar.gz)

Por defecto, ublobs instalará en /usr/local/bin, /usr/local/man, etc. Se puede especificar un prefijo de instalación distinto a /usr/local agregando el parámetro “prefix” en la línea de configure:

```
# ./configure --prefix=PATH
```

Listado 8. Instalación personalizada de ublobs (tar.gz)

En el caso del paquete rpm (para distribuciones Redhat/Fedora), la instalación se hace directamente con:

```
# ./rpm -ivh ublobs-0.1-0.i386.rpm
```

Listado 9. Instalación de ublobs (rpm)

ublobs se configura desde la línea de comandos, a través de un conjunto de parámetros. He aquí la sintaxis y una descripción de las diferentes opciones:

```
# ./ublobs [ -a ] [ -A cacertdir ] [ -b sec ]
[ -c N ] [ -C port ] [ -d ] [ -e host:port ]
[ -E certfile ] [ -f ] [ -F cfgfile ]
[ -G cacertfile ] [ -h ] [ -H ] [ -j dir ]
[ -K keyfile ] [ -l logfile ] [ -L protocol ]
[ -n ] [ -p file ] [ -P ] [ -Q ] [ -r ] [ -R ]
[ -s ] [ -S N ] [ -t sec ] [ -T sec ] [ -u user ]
[ -w file ] [ -W ] [ -x N ] [ -X ] [ -Z ]
[host:]port h1[:p1[:maxc1]] … [hN[:pN[:maxcN]]]
```

Listado 10. Sintaxis de comandos para ublobs

```
-a            Junto con -dd produce la salida en ASCII
              (en lugar de hexadecimal)
-A cacertdir  Directorio de los certificados del CA
              (en formato hashed)
-b sec        Tiempo de exclusión del servidor de la
              lista de servidores elegibles en caso de
              falla, en segundos (por defecto 30)
-c N          Máximo número de clientes (tamaño de la
              tabla de asociaciones, por defecto 2048)
-C port       Puerto de control, donde el demonio del
              balanceador escucha peticiones
-d            Modo debugging. La salida se redirige a
              stderr si se ejecuta en primer plano con
              (-f), o al syslog en otro caso
-dd           Modo extra debugging
-e host:port  Dirección y puerto del servidor de
              emergencia para contactar si todos los
              servers regulares están no disponibles
-E certfile   Usar un certificado dado en formato PEM
-f            Ejecución en primer plano (foreground)
-F cfgfile    Archivo de configuración
-G cacertfileArchivo con el certificado del CA
-h            Selección inicial del servidor según
              hash de la dirección IP del cliente
-H            Agrega el encabezado "X-Forwarded-For" a
              los requests HTTP
-j dir        Ejecución en ambiente chroot (jailed)
-K keyfile    Usar una clave en formato PEM (puede
              estar contenida en el certificado)
-l file       Logging en el archivo especificado
-L protocol   Protocolo usado: ssl23 (por defecto),
              ssl2, ssl3 o tls1
host:port     Dirección local y puerto en el que se
              escuchan requerimientos de servicio (por
              defecto escucha en todas las direcciones
              locales)
-n            Ejecución no bloqueante (non-blocking)
-p file       Escribe el pid en el archivo file
```

```
-P              Usar poll() para notificación de eventos
-Q              Usar kqueue() para notificación de
                eventos (solo en BSD)
-r              Usar selección de servidores por round-
                robin sin hacer seguimiento de clientes
                (no mantiene el estado)
-R              Requerir un certificado válido
-s              Si el servidor asociado inicialmente al
                cliente falla, la conexión se cierra sin
                intentar otro servidor (stubborn server
                selection)
-S N            Número de servidores (por defecto 16)
-t sec          Timeout de conexión en segundo (por
                defecto 5)
-T sec          Tiempo de seguimiento de clientes
                (mantiene la última ruta del cliente al
                servidor, de modo que un cliente siempre
                acceda al mismo servidor para mantener
                el estado de la conexión), en segundos
                (por defecto 0, nunca expira)
-u user         Ejecución como un usuario diferente
-w file         Archivo de salida de los reportes de
                estado en formato HTML
-W              Usar el parámetro peso (weight) para la
                selección del servidor
-x N            Máximo número de conexiones simultáneas
                (por defecto 256)
-X              Agrega un comando de "exit" a la
                interface de control
-Z              Usar el modo de compatibilidad SSL.
hosti:porti:    Dirección, puerto y máximo número de
soft:hard       conexiones simultáneas a un servidor
                remoto i (por defecto, el puerto es el
                mismo que el puerto local y el límite
                débil en el número de conexiones es
                ilimitado. El límite duro se usa para
                clientes activos que ya han accedido al
                servidor)
```

El funcionamiento de ublobs

ublobs básicamente se inicia como un demonio y escucha peticiones de servicio en una dirección local y un puerto dados. Al recibir una solicitud de conexión, dependiendo del algoritmo de selección especificado por línea de comandos, elige un servidor del conjunto de servidores que integran el cluster para el cual ublobs balanceará la carga de trabajo. Estos servidores (sus direcciones y puertos de servicio) también le son pasados a ublobs mediante un parámetro de configuración por línea de comando, así como el peso relativo de cada uno de ellos (si es requerido por el algoritmo). Como algoritmo de selección de servidor, ublobs permite usar alternativamente, una tabla de seguimiento de conexiones, un hash sobre la dirección IP del cliente, según un cierto peso asignado a los servidores o simplemente por Round Robin.

Por ejemplo, la siguiente línea de comandos inicia ublobs en primer plano (-f), con capacidad para atender hasta 25 conexiones simultáneas (-x 25), en modo balance de servicio HTTP para los servidores 10.0.0.2 y 10.0.0.3 con un factor peso (-W) de 2 a 1 respectivamente.

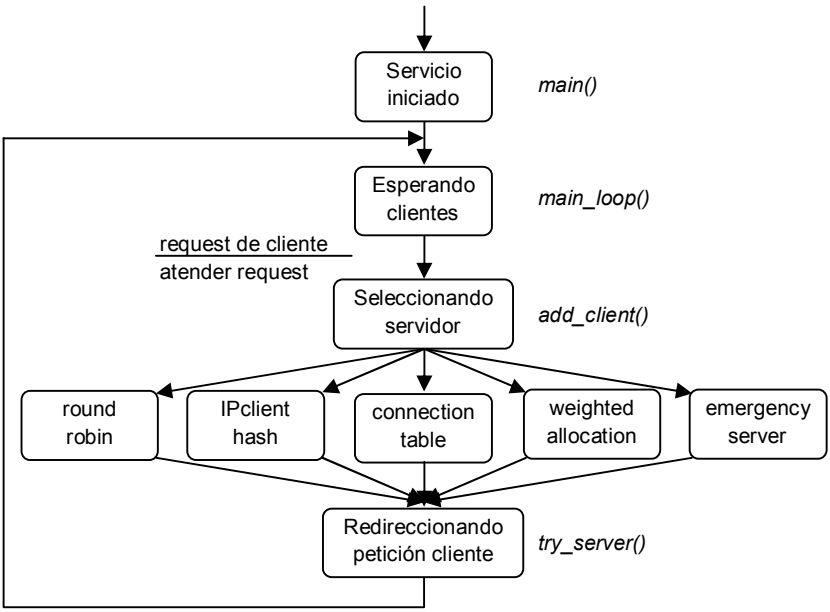
```
# ./ublobs -x 25 -f -W http 10.0.0.2 2 10.0.0.3 1
```

Listado 11. ublobs en modo selección por pesos

Si se utiliza una tabla de conexiones, cada cliente es redirigido por la misma ruta hacia el servidor que tenía asociado (se mantiene el estado de la conexión y el último servidor accedido). En otro caso, ublobs determina algún servidor disponible para atender al pedido. Para todos los casos, sea cual fuere el algoritmo de selección de servidor usado, ublobs redirige el request al servidor seleccionado. Este proceso aparece inadvertido por el cliente (transparencia).

Para asegurar la disponibilidad del servicio, aún en el caso de que todos los servidores del cluster estuvieran fuera de servicio, puede indicársele a ublobs un servidor de emergencia, que será utilizado en estos casos excepcionales.

Podemos esquematizar el funcionamiento de ublobs mediante el siguiente modelo, el cual no debe pensarse como una máquina de estados típica o un diagrama de transición de estados (por la similitud en la simbología), sino como un modelo híbrido, que representa el funcionamiento del sistema en sí.



Esquema 1. Funcionamiento de ublobs

A continuación, se muestran extractos de código pertenecientes a ublobs. El código completo con comentarios técnicos y puntuales, así como la documentación del mismo (*readme*, *howtos*, *changelog*, etc.) se encuentran disponibles para ser consultados en el CD-ROM que acompaña a este trabajo. En forma sintética y a modo de ejemplo, la implementación de ublobs define, entre otras, las siguientes funciones:

- La función principal *main* (listado 12), que inicia un escuchador de puertos para atender las peticiones, interpreta los parámetros de la línea de comandos, y dispara el bucle de ejecución dependiendo del algoritmo de selección con el que se inició ublobs.

```
int main(int argc, char **argv)
{
   listenport = argv[0];

   open_listener(listenport);

   int n = options(argc, argv);
   // control over params

   create_pidfile(pidfile);

   if (use_kqueue) mainloop_kqueue();
   if (use_poll) mainloop_poll();
   if (!use_kqueue && !use_poll) mainloop_select();

   //exiting procedure

   for (i = 0; i < connections_max; i++) close_conn(i);

   if (pidfile) unlink(pidfile);

   return 0;
}
```

Listado 12. La función *main*

- La función bucle de ejecución principal (listado 13), que recibe y atiende las peticiones de servicio de los clientes, disparando el algoritmo de selección de servidor. Si el sistema operativo proveyera los mecanismos, podrían usarse las funciones de sistema *kqueue* o *poll* como métodos de selección alternativos, mejorando los tiempos de respuesta del procedimiento de selección. Se puede consultar información adicional sobre los mecanismos de kqueue y poll en el CD-ROM que acompaña a este trabajo.

```
static void mainloop_select(void)
{
   while (loopflag) {

     // wait for client requests

     // then assign server

     add_client(..., &cli_addr);

     // ...

   }
}
```

Listado 13. La función *mainloop_select*

- La función *add_client* (listado 14), que elige un servidor para atender el requerimiento según el algoritmo de selección indicado: mediante round robin, hash sobre la dirección IP del cliente, según las capacidades del servidor –a partir del peso asignado-, o manteniendo la ruta hacia un servidor, si esta existía en la tabla.

```
static void add_client(..., cli_addr &cli)
{
   if (roundrobin)
   {
     index = current;
     do {
       index = (index + 1) % nservers;
       if (try_server(index, 0, cli) != -1) goto Ok;
     } while (index != current);
   }
   else if (hash)
   {
       index = cli % nservers;
       if (try_server(index, 0, cli) != -1) goto Ok;
   }
   else if ((clino = lookup_client(cli)) != -1)
   {
       // already a client?
       index = clients[clino].cno;
       if (try_server(index, 1, cli) != -1) goto Ok;
   }
   else if (weight)
   {
     index = server_by_weight();
     if (index != -1) {
       if (try_server(index, 0, cli) != -1) goto Ok;
     }
   }

   if (emerg_server != -1)
   {
     if(try_server(emerg_server, 0, cli)!= -1) goto Ok;
   }

Failure:
   // ...
   return;

Ok:
   n = store client(clino, cli, index);
   store_conn(..., n, index);
   return;
}
```

Listado 14. La función *add_client*

- La función de selección de servidor por peso (listado 15), que calcula la relación carga/capacidad para elegir el servidor más "liviano".

```
static int server_by_weight(void)
{
   for (i = 0; i < nservers; i++) {
          load = servers[i].c/servers[i].weight;
          if (best_server == -1 || load < best_load) {
                 best_load = load;
                 best_server = i;
          }
   }
   return best_server;
}
```

Listado 15. La función *server_by_weight*

- La función *try_server* (listado 16), que intenta contactar al servidor seleccionado, retornando un error si lo hubiera.

```
static int try_server(int index, ...)
{
   if (servers[index].port == 0) {
          // ... port unavailable
          return -1;
   }
   if (servers[index].c >= servers[index].maxc) {
          // ... server overloaded
          return -1;
   }

   // opens sock, on error
   //      return -1;

   return 0;
}
```

Listado 16. La función *try_server*

Pruebas de campo

Se propone en este punto un experimento, que consiste en un conjunto de pruebas de performance y stress, para mostrar las diferencias entre implementar un servicio web tradicional centralizado, y de hacerlo repartiendo la carga en varios servidores mediante un mecanismo de distribución de carga; por ejemplo, primero a través de un RRDNS y luego con un LBS ejecutando ublobs.

Para las pruebas que se realizarán será interesante recrear un ambiente de prueba. En el CD-ROM que acompaña a esta presentación se incluyen scripts y el material necesario para reproducir este ambiente usando un laboratorio virtual basado en UML, sin embargo, el ejemplo se enriquece al incorporar otras cuestiones, como la heterogeneidad, retrasos de red, etc., que solo aparecen al tratar con ambientes reales.

El laboratorio de pruebas propuesto es como el que se muestra en la figura 14.

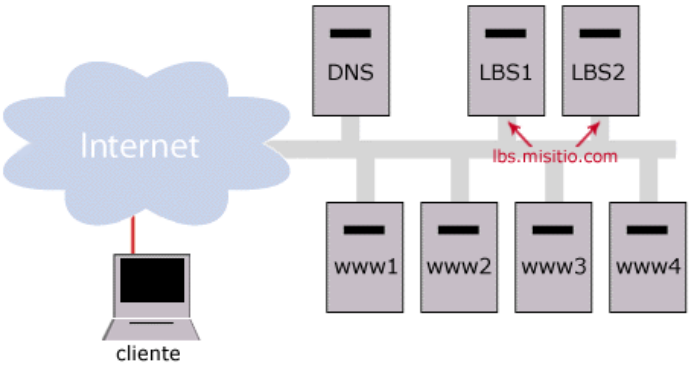


Figura 14. Laboratorio de prueba

Si bien los resultados enunciados a continuación corresponden a pruebas en un ambiente real, se pueden igualmente verificar en un escenario virtual.

Hardware utilizado:

- DNS Server (192.168.0.1, *wwwrr*): AMD Athlon XP 1200+ RAM 128 MB (Linux Fedora Core 5 Kernel 2.6.10)
- Load Balancer Server 1: (192.168.0.3, *LBS1*): Pentium MMX 233 Mhz 64 MB RAM (Linux RedHat 7.2 Kernel 2.4.20-8, i586)
- Load Balancer Server 2: (192.168.0.4, *LBS2*): ídem anterior.
- Web Server 1 (192.168.0.11, *www1*): ídem anterior.
- Web Server 2 (192.168.0.12, *www2*): Pentium MMX 233 Mhz 32 MB RAM (Linux RedHat 7.2 Kernel 2.4.7-10, i686)
- Web Server 3 (192.168.0.13, *www3*): Pentium Pro 200 Mhz 32 MB RAM (Linux RedHat 7 v1 Kernel 2.4.7-10, i586)
- Web Server 4 (192.168.0.14, *wwwc*): AMD Athlon XP 1200+ 128 MB RAM (Windows 2000)
- Cliente / Stress Server (192.168.0.20): AMD Turion X2 64 2 GHz 1 Gb RAM (Fedora Core 7)

NOTA: Para hacer más comparativo al experimento se propone elegir al servidor más potente para el servicio centralizado, y una combinación de servidores para el cluster, tal que en suma, se equiparen las capacidades del primero.

Software utilizado:

- Web Server 1: Apache 1.3.19 + php 4.0.6 + MySQL 3.11
- Web Server 2-3: Apache 2.2.4 + php 5.2.2 + MySQL 5.0.37
- Web Server 4: Apache 2.2.4 + php 5.2.2 + MySQL 5.0.37
- DNS Server: Bind 8.2.3-REL-NOESW
- Load Balancer: ublobs-0.1
- Stress Server (cliente): Cliente web + Apache ab + openload + httperf

NOTA: La elección adecuada de la cantidad de memoria de los LBS elimina el riesgo de que el experimento resulte en valores espúreos producto del overead del swap. Se recomienda asignar el equipo con mayor cantidad de RAM para funcionar como LBS.

Configuración inicial e inicialización.

1) Crear un Round Robin DNS "webrr.misitio.com" que apunte a las direcciones IP de www1, www2, y www3.

```
wwwrr.misito.com        IN      A     192.168.0.11
                        IN      A     192.168.0.12
                        IN      A     192.168.0.13
```

Listado 17. Extracto del archivo zone.misitio.com (1)

2) Crear una entrada DNS “wwwlb.misitio.com” para el balanceador, y otra para el servidor centralizado “wwwc.misitio.com”.

```
wwwc.misitio.com        IN      A     192.168.0.14
wwwlb.misitio.com       IN      A     192.168.0.2
```

Listado 18. Extracto del archivo zone.misitio.com (2)

3) Crear un archivo “index.php” en cada webserver con el siguiente contenido dinámico:

```
<?php
   $server_ip = $HTTP_SERVER_VARS['SERVER_ADDR'];
   $webserver = $HTTP_SERVER_VARS['SERVER_NAME'];
   echo "Soy $webserver en $server_ip</h3>";
?>
```

Listado 19. Archivo de prueba index.php (1)

4) Iniciar el balanceador de carga en LBS1 y LBS2.

```
# ublobs -r 80 www1 www2 www3
```

Listado 20. Línea de comandos para ublobs (1)

Una excelente herramienta para testeo de performance y stress de servidores, es *ab*, y se incluye en la versión del paquete Apache, llamada así por sus siglas en inglés que derivan de *Apache server Benchmarking tool*. Ente otras estadísticas importantes, muestra especialmente cuántos requerimientos por segundo es capaz de atender un servidor web y cuanto toma atender un requerimiento en promedio.

La sintaxis del comando ab es la siguiente:

```
ab [ -A auth-username:passw ] [ -c concurrency ]
[ -C cookie-name=value ] [ -d ] [ -e csv-file ]
[ -g gnuplot-file ] [ -h ] [ -H custom-header ]
[ -i ] [ -k ] [ -n requests ] [ -p POST-file ]
[ -P proxy-auth-username:passw ] [ -q ] [ -s ]
[ -S ] [ -t timelimit ] [ -T content-type ]
[ -v verbosity] [ -V ] [ -w ]
[ -x <table>-attributes ] [ -X proxy[:port] ]
[ -y <tr>-attributes ] [ -z <td>-attributes ]
[http://]hostname[:port]/path
```

Listado 21. Sintaxis de comandos de Apache ab

Para el test de stress, se propone usar la herramienta de medición de throughput *openload*, la cual resulta de especial utilidad descubriendo cuellos de botella en HTTP. Incluso se pueden corroborar los resultados

usando *httperf* [12], una aplicación para asistir en el "tuning" de los parámetros de HTTP respecto de la performance, así como la cantidad de usuarios concurrentes y los tiempos de respuesta.

La sintaxis de comandos de openload es la siguiente:

```
# openload [http://]URL[:port] [clients]
```

Listado 22. Sintaxis de comandos para openload

Pruebas.
(Desde el cliente/stress server)

B#1: Prueba de performance del servidor web centralizado.

```
# ab -n 300 -c 15 http://wwwc.misitio.com/
```

Listado 23. Prueba de performance con Apache ab (B#1)

B#2: Prueba de performance de la granja de servidores con balance de carga por RRDNS.

```
# ab -n 300 -c 15 http://wwwrr.misitio.com/
```

Listado 24. Prueba de performance con Apache ab (B#2)

B#3: Prueba de performance de la granja de servidores con balance de carga por un LBS con ublobs.

```
# ab -n 300 -c 15 http://wwwlb.misitio.com/
```

Listado 25. Prueba de performance con Apache ab (B#3)

B#4: Prueba de stress del servidor centralizado.

```
# openload http://wwwc.misitio.com 10
```

Listado 26. Prueba de stress con openload (B#4)

B#5: Prueba de stress de la granja de servidores con balance de carga por RRDNS.

```
# openload http://wwwrr.misitio.com 10
```

Listado 27. Prueba de stress con openload (B#5)

B#6: Prueba de stress de la granja de servidores con balance de carga por un LBS con ublobs.

```
# openload http://wwwlb.misitio.com 10
```

Listado 28. Prueba de stress con openload (B#6)

Resultados y Observaciones

Los resultados de las pruebas fueron los siguientes:

```
Time taken for tests:   311.448 seconds
Complete requests:      300
Failed requests:        0
Total transferred:      143700 bytes
HTML transferred:       94200 bytes
Requests per second:    0.96 [#/sec] (mean)
Time per request:       15572.40 [ms] (mean)
Transfer rate:          0.46 [Kbytes/sec] received

Connnection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0   140  763.4      0  8999
Processing: 15093 15163  279.9  15127 17669
Waiting:    15093 15163  279.9  15127 17669
Total:      15093 15303  936.6  15127 24144
```

Listado 29. Resultados de B#1

```
Time taken for tests:   311.558 seconds
Complete requests:      300
Failed requests:        0
Total transferred:      143700 bytes
HTML transferred:       94200 bytes
Requests per second:    0.96 [#/sec] (mean)
Time per request:       15577.90 [ms] (mean)
Transfer rate:          0.46 [Kbytes/sec] received

Connnection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0   139  760.0      0  8896
Processing: 15082 15152  201.0  15109 17712
Waiting:    15082 15152  200.9  15109 17713
Total:      15082 15291  860.2  15118 24129
```

Listado 30. Resultados de B#2

OBS#1. Una observación interesante, en relación a que los valores obtenidos en B#1 y B#2 son similares, se debe al problema de los RRDNS mencionado en el capítulo 2: "(…) *Desafortunadamente (…) los DNS intermedios y el software de los clientes (navegadores), suelen almacenar en cache las direcciones IP retornadas por el DNS (…) usando siempre la misma ruta al servidor una vez que la obtuvo, salteando el balanceador y por lo que el proceso de balanceo fracasa*". Para B#2, la herramienta Apache ab resolvió wwwrr.misitio.com a la dirección de www2 para el primer request, ésta fue cacheada y luego los siguientes 299 fueron igualmente redirigidos (esto se puede ver en el log de Apache en www2).

```
Time taken for tests:   303.120 seconds
Complete requests:      300
Failed requests:        0
Total transferred:      143700 bytes
HTML transferred:       94200 bytes
Requests per second:    0.99 [#/sec] (mean)
Time per request:       15156.00 [ms] (mean)
Transfer rate:          0.47 [Kbytes/sec] received

Connnection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0     0    0.4      0     4
Processing: 15100 15150   35.3  15141 15337
Waiting:    15100 15149   35.2  15141 15336
Total:      15100 15150   35.8  15141 15339
```

Listado 31. Resultados de B#3

OBS#2. Otra observación es un resultado lógicamente esperado: mientras que para el webserver centralizado el tiempo de conexión en promedio fue de 140 ms. (alcanzando un tiempo máximo de alrededor de 8000 ms.), para la granja de servidores con balance de carga, fue de 0.037 ms. Los tiempos de proceso y espera no varían mucho, sin embargo muestran una mejora al usar distribución de carga. Finalmente existe una relación de 1.55:1 (24129/15539 ≈ 1.55) en el tiempo máximo total de atención a favor de la granja de servidores (ver figura 15).

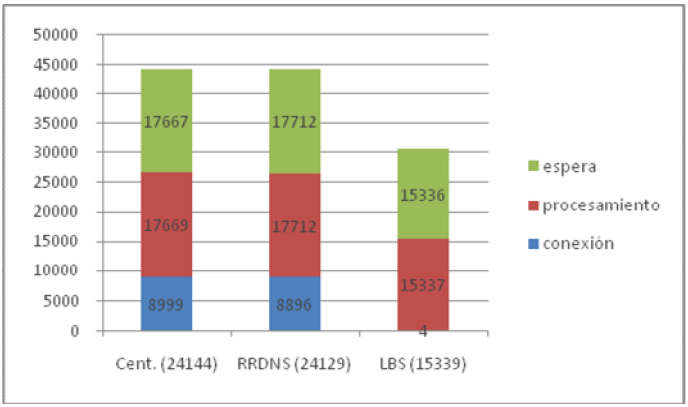


Figura 15. Comparativa de tiempos máximos de atención (en seg^{-3}.)

Las siglas de los listados 27, 28 y 29, a continuación, deben interpretarse como sigue:

- Tps: (Transacciones por segundo) Número de requests completados durante el intervalo.
- MaTps: Promedio de Tps (estimado).
- ResTime: Promedio de tiempos de respuesta (en seg.) para el intervalo.
- Err: Porcentaje de error (status code distinto a HTTP 200 OK).
- Count: Número de requests completados en total.
- Total Tps: Requests completados por Tiempo transcurrido.
- Avg. Response time: Prom. total de tiempo de respuesta (en seg).
- Max Response time: Mayor tiempo de respuesta de la corrida.

```
MaTps      Tps       ResTime   Err     Count
141,43     141,43    0,021       0%     142
144,49     172,00    0,008       0%     314
146,99     169,49    0,006       0%     484
133,99      16,99    0,225       0%     644
```

```
137,19     166,00    0,404       0%     810
124,88      14,03    0,101       0%     963
```

```
113,15       7,65    0,615       0%    1125
103,30      14,68    0,335       1%    1205
 92,99       0,11    9,001       0%    1206
 84,05       3,63    7,275       0%    1246
 75,94       2,95    6,685       6%    1261
 68,42       0,78    5,602      28%    1268
 61,68       1,00    7,776      12%    1276
 55,75       2,33    7,260     100%    1283
 50,24       0,67    3,755      87%    1291

Total TPS:  11.90
Avg. Response time:  0.645 sec
Max Response time:  46.199 sec
Total Requests:    1291
Total Errors:        19
```

Listado 32. Resultados de B#4

```
MaTps      Tps       ResTime  Err     Count
141,44     141,44    0,022      0%      148
142,27     155,59    0,016      0%      314
165,35     178,92    0,009      0%      484
140,28      22,12    0,114      0%      644
132,23     172,64    0,396      0%      810
127,16      28,52    0,254      0%      963
 90,65      11,18    0,760      0%     1125
101,67      13,12    0,854      0%     1205
 89,92       7,62    5,625      8%     1206
 78,55       5,37    8,455      0%     1246
 69,16       2,95    9,390      9%     1261
 68,88       2,40    8,602     31%     1268
 59,16       2,51    7,776     42%     1276
 55,30       2,33    7,260     95%     1283
 50,16       3,13    2,985     61%     1297

Total TPS:  12.07
Avg. Response time:  0.788 sec
Max Response time:  45.793 sec
Total Requests:    1297
Total Errors:        22
```

Listado 33. Resultados de B#5

```
MaTps      Tps     ResTime   Err    Count
207,00     207,00    0,014    0%      207
207,96     216,57    0,014    0%      424
209,49     223,33    0,013    0%      648
211,50     229,54    0,128    0%      878
213,20     228,54    0,026    0%     1107
204,21     123,27    0,040    0%     1258
184,26       4,68    1,192    0%     1263
165,92       0,87    2,959    0%     1264
149,38       0,54    2,996    0%     1265
148,27     138,32    0,023    0%     1426
155,66     222,11    0,004    0%     1649
162,16     220,68    0,089    0%     1871
168,82     228,77    0,028    0%     2100
174,57     226,32    0,224    0%     2327
167,39     102,73    0,031    0%     2523

Total TPS: 123.80
Avg. Response time:  0.059 sec
Max Response time:   9.434 sec
Total Requests:    2523
Total Errors:         0
```

Listado 34. Resultados de B#6

OBS#3. La última observación tiene que ver con el desempeño mostrado por las alternativas planteadas. El test de stress sobre el servidor centralizado (lo mismo para la granja con balance por RRDNS, por igual motivo al señalado en OBS#1) muestra que los tiempos de atención son en, en el extremo, 10 veces superiores a los de una granja con balance por LBS corriendo ublobs. Incluso el tiempo máximo de respuesta es de 5:1 a favor de ublobs: 45.793 seg. contra 9.434 seg. (ver figura 16).

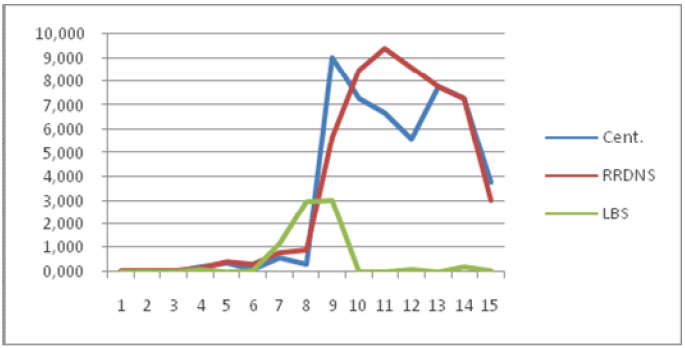


Figura 16. Comparativa de Tiempos de respuesta (en seg.)

Más allá de estos valores hay un resultado aún más contundente que no debemos pasar por alto: la evolución de la tasa error producto de la sobrecarga generada por la prueba de stress. Con el paso del tiempo y a medida que se saturaron las capacidades para atender requests, el error fue en aumento para el servidor centralizado (y lo mismo en el caso de balance por RRDNS).

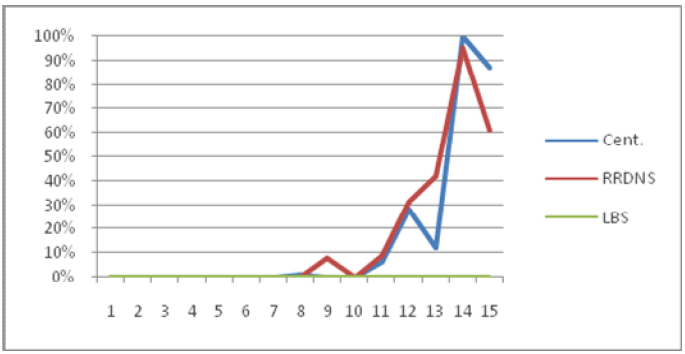


Figura 17. Comparativa de la tasa de error (en %)

La ventaja de un LBS aquí fue más que evidente, preservando la tan buscada alta disponibilidad.

CAPITULO 4
CONCLUSIÓN

> *El último capítulo representa el final del ciclo de la emoción. Es el momento en que, después de todo el viaje colmado de adrenalina que fue escribirlo, llega el relax y me permite prender un cigarro. A veces el final es un momento de menos fuerza, más descansado. Incluso más pelado a nivel poético, con vocablos más rudos. Es una poesía que, por el momento del viaje en el que llega, tiene un clima comparable a lo que pasa con las chicas lindas: no necesitan de mucho maquillaje.*
>
> Anónimo

En este trabajo, he intentado mostrar una solución generalizada para el problema del balance de carga en los servidores de un sistema distribuido heterogéneo. En primer lugar, definiendo el área de incumbencia del problema y luego, analizando soluciones existentes en busca de inconvenientes que justifiquen nuevos desarrollos, como éste. En segundo lugar, he propuesto una implementación sencilla de un protocolo de balance de carga, a partir de rescatar lo mejor de intentos anteriores y sumándole las ventajas de otras tecnologías, como por ejemplo el protocolo de redundancia VRRP para hacerlo más tolerante a fallos. El resultado de ello ha sido ublobs, al que de alguna forma he

“enfrentado” con otras soluciones existentes a través de una serie de pruebas propuestas para mostrar las mejoras en el rendimiento en general que se obtienen al usarlo como balanceador de carga.

Con respecto a los resultados de estas pruebas, se puede observar en ellos que el transfer rate, al igual que los request por segundo, son mejores al usar ublobs (ver listados 29 a 31). Una consecuencia de esto resultó en que los tiempos de prueba con este enfoque fueron sensiblemente menores. También las conexiones se abrieron más rápido y el tiempo de procesamiento fue menor en este caso. Comparado con el balance por RRDNS, aún presentando mejor rendimiento que por Reverse Proxy, e incluso haciendo que las traducciones DNS no permanezcan en caché, no evitamos el overhead debido a las sucesivas queries de traducción. A esto se le suma el hecho de que si un servidor no está disponible recibiría peticiones de todas formas. Finalmente, los tiempos totales de atención presentaron una relación de 5:1 a favor del servicio con el LBS. Bajo stress, solo se verificó alta disponibilidad en el caso del LBS: el RRDNS y el caso centralizado presentaron errores por sobrecarga, lo que derivó en interrupciones del servicio.

En definitiva, un LBS pareciera ser la mejor de las tres alternativas propuestas: ublobs permite distribuir de manera pareja la carga y resulta ser, además de mejor en cuanto a performance, más tolerante a fallos.

Con estas conclusiones puedo decir que, en principio, la solución para balance de carga de servicios replicables, sencilla, de fácil implementación y de bajo costo que buscaba, existe y ublobs es la prueba de ello.

Tareas pendientes y desarrollos futuros

Como tarea pendiente, queda un análisis más profundo de las estadísticas para hacer los resultados aún más gráficos. Incluso propongo usar herramientas complementarias como *autobench* [13] o *hammerhead* [14] para automatizar el proceso de testing. En el CD-ROM que acompaña a este trabajo, se incluyen los binarios, fuentes, documentación y enlaces de interés para estas herramientas.

Respecto de desarrollos futuros, creo que la punta de la madeja está en continuar la búsqueda de algoritmos más eficientes que los que se han implementado para este proyecto. Por ejemplo, un buen comienzo sería intentar la selección de servidores por less-used-server (LUS), o lowest-response-time (LRT). También se ha incluido mayor información y enlaces de interés en este sentido, en el CD-ROM que acompaña a este trabajo.

De todas formas y cualquiera sea el aporte, será siempre en pos de algo mejor que está por venir.

APÉNDICE I

IMPLEMENTANDO VRRP

La siguiente es una guía de instalación y prueba de laboratorio del protocolo de redundancia para brindar alta disponibilidad a una granja de servidores web.

Para configurar este ambiente de prueba, se puede usar un par de PCs como LBS (en ambas se deberá instalar e iniciar ublobs y vrrpd) y algunas otras como Web Servers (en las que se deberá instalar e iniciar Apache con PHP y una página dinámica de prueba). Un ejemplo es como el que se muestra en la figura siguiente:

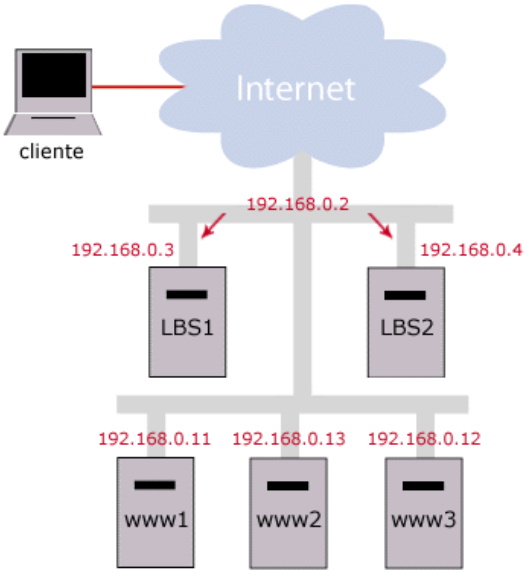


Figura 18. Un ambiente de prueba para vrrpd

Una página dinámica de prueba simple pero efectiva puede ser como la siguiente:

```
<?php
   $server_ip = $HTTP_SERVER_VARS['SERVER_ADDR'];
   $webserver = $HTTP_SERVER_VARS['SERVER_NAME'];
   echo "Respuesta desde:<br>";
   echo "<h3>$webserver en $server_ip</h3>";
?>
```

Listado 35. Archivo de prueba index.php (2)

En principio se debe verificar el acceso desde el cliente directamente a cada uno de los dos servidores web para comprobar que estén en correcto funcionamiento. Si todo marcha bien se verán las distintas respuestas de los servidores, y se podrá pasar a probar los LBSs. Luego de instalado, se debe iniciar el servicio ublobs de forma similar a la que se muestra en el ejemplo (incluyendo todos los servidores para los que se pretenda balancear carga):

```
# ublobs -r 80 192.168.0.11 192.168.0.12 192.168.0.13
```

Listado 36. Línea de comandos para ublobs (2)

Acto seguido, se debe acceder desde el cliente a los servidores web ahora a través de los LBSs para comprobar que estén funcionando. Se recomienda repetir el proceso varias veces hasta obtener las respuestas de los tres servidores a través de los dos LBS.

Por último, se debe instalar e iniciar el protocolo de redundancia para brindar alta disponibilidad al cluster web. En este sentido, existen varias alternativas, pero la mayoría son soluciones propietarias, entre ellas:

- HSRP (Hot Standby Routing Protocol). Solución de redundancia de routers propiedad de Cisco.
- (GLBP) Gateway Load Balancing Protocol - Solución de redundancia de routers con balanceo de carga propiedad de Cisco.
- (EAPS) Ethernet Automatic Protection Switching.
- (CARP) Common Address Redundancy Protocol. Alternativa a HSRP y VRRP no propietaria, libre de patentes y sin restricciones, pero obsoleta.
- (RSMT) Routed Split Multilink Trunking. Solución de redundancia de routers propiedad de Nortel

VRRP, por otro lado, es una versión estandarizada del HSRP de Cisco. Este protocolo de redundancia no propietario definido en la RFC 3768, fue diseñado para aumentar la disponibilidad de la puerta de enlace por defecto, dando servicio a máquinas en la misma subred. El aumento de fiabilidad se consigue mediante el anuncio de un router virtual como una puerta de enlace por defecto en lugar de un router físico. Dos o más routers físicos se configuran representando al router virtual, con sólo uno de ellos realizando realmente el enrutamiento. Si el router físico que está realizando el enrutamiento falla, el otro negocia la sustitución. Se denomina router maestro al router físico que realiza realmente el enrutamiento y routers de respaldo a los que están en espera por si el maestro falla.

Hay que tener en cuenta que VRRP es un protocolo de router, no de routing. Cada instancia de VRRP se limita a una única subred. No anuncia rutas IP, ni afecta las tablas de encaminamiento.

Implementación de VRRP. Un router virtual tiene que utilizar la siguiente dirección MAC: 00-00-5E-00-01-XX. El último byte de la dirección es el identificador de router virtual (Virtual Router IDentifier o VRID), que es diferente para cada router virtual en la red. Esta dirección sólo la utiliza un único router físico a la vez, y es la única forma de que

otros routers físicos puedan identificar el router maestro en un router virtual. Los routers físicos que actúan como router virtuales deben comunicarse entre ellos utilizando paquetes con dirección IP multicast 224.0.0.18 y número de protocolo IP 112.

Los routers maestros tienen una prioridad de 255 y los de respaldo entre 1 y 254. Cuando se realiza un cambio planificado de router maestro se puede cambiar manualmente su prioridad a 0, lo que fuerza a que algún router de respaldo se convierta en maestro rápidamente. De esta forma se reduce el periodo de "agujero negro".

La elección del router maestro. Un fallo en la recepción de un paquete multicast del master durante un tiempo superior a tres veces el tiempo de anuncio, hace que los routers de respaldo asuman que el router maestro está caído. El router virtual cambia su estado a "inestable" y se inicia un proceso de elección para seleccionar el siguiente router maestro de entre los routers de respaldo. Esto se realiza mediante la utilización de paquetes multicast.

Hay que hacer notar que los routers de respaldo únicamente envían paquetes multicast durante el proceso de elección. Una excepción a esta regla es cuando un router físico se configura para que "derroque" al master actual. Esto permite al administrador de red forzar a que un router sea el maestro inmediatamente después de un arranque, por ejemplo cuando un router es más potente que otros, o cuando un router utiliza el ancho de banda más barato. El router de respaldo con la prioridad más alta se convierte en el router maestro aumentando su prioridad a 255 y enviando paquetes ARP con la dirección MAC virtual y su dirección IP física. Esto redirige los paquetes del maestro caído, al router maestro actual. En los casos en los que los routers de respaldo tengan todos el mismo valore de prioridad, el router de respaldo con la dirección IP más alta se convierte en el router maestro.

VRRP se puede usar sobre redes Ethernet, MPLS y Token Ring. Varios fabricantes ofrecen routers y switches de nivel 3 que pueden utilizar el

protocolo VRRP. También están disponibles implementaciones software de la RFC 3768 (en el CDROM que acompaña a este proyecto se incluye una copia del draft) para Linux y BSD [15] [16].

vrrpd es una implementación de VRRP desarrollada bajo licencia GPL para Unix/Linux y portado también a BSD. El sitio del proyecto es http://sourceforge.net/projects/vrrpd/ y de allí puede descargase el código fuente y compilarlo, así como también binarios y documentación.

vrrpd permite varias opciones de configuración. He aquí la sintaxis de la línea de comandos con la descripción correspondiente:

```
# vrrpd [ -a auth ] [ -h ] -i ifname [ -n ]
[ -p prio ] [-s] -v vrid ipaddr
```

Listado 37. Sintaxis del comando vrrpd

```
-a auth      establece el tipo de autenticación
             (aún no implementado)
-d delay     establece el intervalo de anuncio en
             segundos (por defecto 1)
-f piddir    especifica el directorio del archivo
             .pid (por defecto /var/run)
-h           muestra la ayuda en línea
-i ifname    la interfaz de red en la que se
             ejecutará
-n           no tiene en cuenta la dirección MAC
             virtual
-p prio      establece la prioridad del host en el
             cluster (por defecto 100)
-s           modo apropiativo (por defecto)
-v vrid      el id del cluster [1-255]
ipaddr       la dirección virtual del cluster
```

Antes de iniciar el protocolo de redundancia, se puede probar inspeccionando la configuración de las interfaces de red de los routers virtuales, haciendo un especial énfasis en las direcciones MAC (HWaddr). Para el ejemplo desarrollado:

```
root@lbs1:~# ifconfig
eth0 Link encap:Ethernet HWaddr 16:C1:D7:63:91:EB
     inet addr:192.168.0.3 Bcast:192.168.0.255...
```

Listado 38. Configuración de eth0 para LBS1 (1)

```
root@lbs2:~# ifconfig
eth0 Link encap:Ethernet HWaddr E6:CD:F3:39:25:D1
     inet addr:192.168.0.4 Bcast:192.168.0.255...
```

Listado 39. Configuración de eth0 para LBS2 (1)

Una vez instalado vrrpd en ambos LBSs, se deberá iniciar el protocolo de manera similar al ejemplo mostrado a continuación:

```
# vrrpd -i eth0 -v 1 192.168.0.2
```

Listado 40. Línea de comandos para vrrpd

Si se verifican las interfaces de red de los LBSs nuevamente, se podrá notar que una de ellas habrá cambiado por otra del tipo 00-00-5E-00-01-XX (tal como lo especifica el protocolo). Por ejemplo,

```
root@lbs1:~# ifconfig
eth0 Link encap:Ethernet HWaddr 00:00:5E:00:01:01
```

Listado 41. Configuración de eth0 para LBS1 (2)

Si desde el cliente se accede a la dirección que se ha designado como dirección virtual del cluster, uno de los LBS estará activo (para el ejemplo LBS1) y éste responderá en esa dirección. Se puede probar, durante la actividad, desconectarlo de la red para simular una falla. De este modo, el otro LBS asumirá la dirección del cluster, restaurando su funcionalidad.

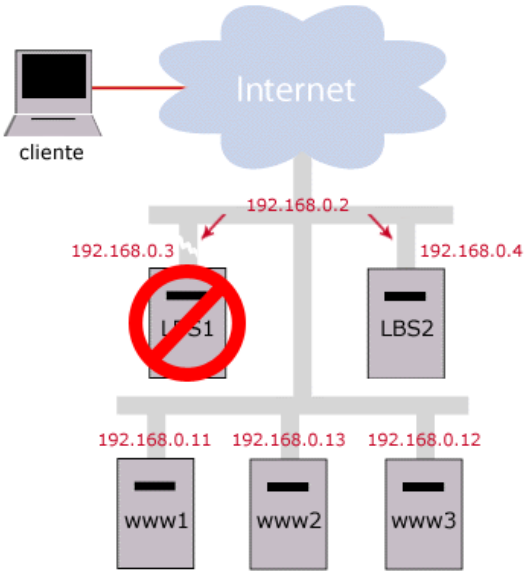


Figura 19. Un LBS maestro es reemplazado por uno de respaldo en caso de que el primero falle

Si se verifica una vez más la configuración de red de la interface de red, ahora de LBS2, se verá lo siguiente:

```
root@lbs2:~# ifconfig
eth0 Link encap:Ethernet HWaddr 00:00:5E:00:01:01
```

Listado 42. Configuración de eth0 para LBS2 (2)

De este modo, se refleja como LBS2 asume la dirección hardware antes asignada a LBS1 y con ella, la responsabilidad de atender solicitudes en la dirección IP virtual del cluster.

Adicionalmente, si las pruebas se llevan a cabo en un ambiente virtual UML, se puede ver en el log del switch de interconexión *uml_switch*, la forma en la que se produce la asignación de la IP virtual inicialmente a un router y cómo es asumida por otro, en caso de que el primero falle. Por ejemplo:

```
New connection
 Addr: 92:de:58:26:7f:6e New port 5  <== LBS1init

New connection
 Addr: 46:78:6b:b8:2f:23 New port 7  <== LBS2init
 Addr: 00:00:5e:00:01:01 New port 5  <== LBS1mast

 (LBS1down)

 Addr: 00:00:5e:00:01:01 New port 7 old port 5
                                     <== LBS2mast
```

Listado 43. Log del uml_switch

APÉNDICE II

CÁLCULO DE NOS Y MBR

Un servidor destinado a atender requerimientos HTTP puede ejecutar varios procesos servidor en simultáneo para tal fin. Aquí se propone un valor recomendado para la cantidad de estos procesos servidor (NOS) para brindar un servicio web con una calidad aceptable, esto es, tiempos de respuesta razonables y sin interrupciones de servicio producto de la saturación, en definitiva, alta disponibilidad. También se plantea un modo de calcular la cantidad mínima de memoria (MBR) con la que debe contar cada servidor para funcionar aceptablemente.

Se necesitan cinco componentes para los cálculos:

- El número máximo (esperado) de requerimientos por minuto (RPM, *requests per minute*)
- El tiempo promedio en segundos que le toma al servidor atender un request (SPR, *seconds per request*)
- El porcentaje de disponibilidad full-time de los servers (SAR, *server availability rate*)
- La memoria en MB que consume cada proceso servidor (SPS, *server process size*)
- El porcentaje de memoria que se dispone para cada proceso servidor (SDM, *server dedicated memory*)

Las fórmulas son [17]:

$$NOS = ceil\left(RPM \times SPR \times \frac{1}{60} \times \frac{100}{SAR} \right)$$

Ecuación 1. Cálculo del número de servidores (NOS)

$$MBR = ceil\left(SPS \times NOS \times \frac{100}{SDM} \right)$$

Ecuación 2. Cálculo del tamaño mínimo de memoria (MBR)

Se propone el siguiente ejemplo práctico: se cuenta con una política de mantenimiento que asegura que el 80% (SAR=80) de los servidores estará disponible en todo momento y se dispone conservativamente del 70% (SDM=70) de la memoria RAM disponible para el servicio. Además se ha estimado la cantidad de requerimientos por minuto (RPM=100) y el tiempo de atención, que oscila entre los 0.5 y 4 segundos (SPR=2). Además, cada proceso servidor (apache-1.3) requiere de 700 a 900 KB de memoria RAM (SPS=0.9). Entonces se tiene:

$$NOS = ceil\left(100 \times 2 \times \frac{1}{60} \times \frac{100}{80} \right) = 5$$

Ecuación 3. Ejemplo de cálculo del NOS

$$MBR = ceil\left(0.9 \times 5 \times \frac{100}{70} \right) = 7$$

Ecuación 4. Ejemplo de cálculo de MBR

REFERENCIAS

[1] Bohn C., Lamont, G. *Load Balancing for Heterogeneous Clusters of PCs*. Future Generation Computer Systems, 2002.

[2] Cortés, Ripoll, Cedó, et al. *An Asynchronous and Iterative Load Balancing Algorithm for Discrete Load Model*. Journal of Parallel and Distributed Computing, 2002.

[3] Naiouf, De Giusti, De Giusti, et al. *Procesamiento paralelo y distribuido. Fundamentos y aplicaciones*. Instituto de Investigación en Informática LIDI (III-LIDI) Facultad de Informática – UNLP, 2006.

[4] Eriksson, U. *Pen Load Balancer*. http://siag.nu/pen

[5] Tanembaum, A. *Distributed Operating Systems*. Prentice-Hall International, 1995.

[6] Coulouris G., Dollimore J., Kindberg T. *Distributed Systems Concepts and Design*. Third Edition. Addison-Wesley, 2001.

[7] *Balance de carga y Algoritmos para balance de carga.* http://es.wikipedia.org/wiki/Balance_de_carga

[8] *Introduction to Server Load Balancing*. http://content.websitegear.com/article/load_balance.htm

[9] *Redbook Communications Server for z/OS V1R2 TCP/IP Implementation Guide*, SG24-6517-00. http://findarticles.com/p/articles/mi_qa3649/is_199812/ai_n8812337

[10] Banawan, S. A. y Zeidat, N. M. *A Comparative Study of Load Sharing in Heterogeneous Multicomputer Systems*. IEEE, 1992.

[11] The Internet Society (1998).

[12] Mosberger, D. y Jin, T. *httperf – una herramienta para medir la performance de un Web server*.
http://www.hpl.hp.com/research/linux/httperf/

[13] Midgley, J. *Autobench – un wrapper en Perl para httperf*.
http://www.xenoclast.org/autobench/

[14] Wong, G. *Hammerhead – Un tester de stress para Web servers*.
http://hammerhead.sourceforge.net

[15] Etienne, J. *VRRPd: overview, implementation and usage*.
http://off.net/~jme/vrrpd/index.html

[16] Cross, V. *Linux on IBM: VRRP*. IBM Redbooks, 2003.
http://www.ibm.com/redbooks

[17] Engelschall, R. S. *Apache Group and FreeBSD, Inc*. Technische Universität München (TUM), Alemania.

BIBLIOGRAFIA

Blum, Richard. *Network Performance Toolkit Using Open Source Testing Tools*, Wiley & Sons, 2003.

Coulouris G., Dollimore J., Kindberg T. *Distributed Systems Concepts and Design*. Second Edition. Addison-Wesley, 1994.

Coulouris G., Dollimore J., Kindberg T. *Distributed Systems Concepts and Design*. Third Edition. Addison-Wesley, 2001.

Jeff, D. User Mode Linux. *Bruce Perens Open Source Series*, Prentice Hall, 2006.

Echaiz J., Davicino S., Ardenghi J. *Una Nueva Clasificación de las Políticas de Distribución de Carga*. VII Workshop Internacional en Sistemas Distribuidos y Paralelismo, pp. 392-401. Chillan, Chile, 2003.

Mosberger, D. y Jin, T. *httperf: A Tool for Measuring Web Server Performance*, 1998.
http://www.hpl.hp.com/research/linux/httperf/wisp98/httperf.pdf

Raynal, M. *Distributed Algorithms and Protocols*. John Wiley & sons, 1988.

Tanembaum, A. *Distributed Operating Systems*. Prentice-Hall International, 1995.

Wikipedia, http://es.wikipedia.org/wiki/

FIGURAS

LISTADOS

ECUACIONES

GLOSARIO

Las definiciones que aquí se presentan pueden consultarse y ampliarse en http://whatis.techtarget.com/definitionsAlpha/

Cliente. Un sistema o proceso que solicita a otro sistema o proceso que le preste un servicio. Una estación de trabajo que solicita el contenido de un archivo a un servidor es un cliente de este servidor. Ver también: "client-server model", "server".

Client-server model (modelo cliente-servidor). Forma común de describir el paradigma de muchos protocolos de red en los que interviene un servidor y uno o más clientes.

Débilmente acoplado. Un sistema de hardware o software se dice débilmente acoplado cuando existe un grado de independencia tal, que permita que sus componentes no dependan unos de otros y puedan ser fácilmente reemplazados, sin que esto produzca o requiera de cambios en los demás componentes del sistema.

DNS (Domain Name Service) (Servicio de Nombres de Dominio). Base de datos distribuida que mapea nombres de sistemas con direcciones IP y viceversa en una estructura jerárquica.

Dominio. Conjunto de computadoras que comparten una característica común, como el estar en el mismo país, en la misma organización o en el mismo departamento. Cada dominio es administrado por un servidor de dominios.

Firewall. Un sistema diseñado para evitar accesos no autorizados desde o hacia una red privada. Los Firewalls pueden estar implementados en hardware o software, o una combinación de ambos.

Los firewalls son frecuentemente utilizados para evitar el acceso no autorizado de usuarios de internet a redes privadas conectadas a la misma, especialmente intranets. Todos los mensajes que dejan o entran a la red, pasan a través del firewall, el cual los examina y bloquea aquellos que no cumplan con determinado criterio de seguridad.

FTP (File Transfer Protocol) (Protocolo de Transferencia de Archivos). Protocolo parte de la arquitectura TCP/IP utilizado para la transferencia de archivos.

FQDN (Fully Qualified Domain Name) (Nombre de Dominio Totalmente Cualificado). El FQDN es el nombre completo de un sistema y no sólo el nombre del sistema. Por ejemplo, "www3" es un nombre de sistema y "www3.misitio.com.ar" es un FQDN.

HLD (Hardware Load-Balance device) (Dispositivo hardware para balance de carga). Es un dispositivo hardware, que podría ser un router o switch, implementando un algoritmo para balance de carga.

Host (sistema central). Computadora que permite a los usuarios comunicarse con otros sistemas centrales de una red. Los usuarios se comunican utilizando programas de aplicación, tales como el correo electrónico, Telnet y FTP.

HTML (Hyper-Text Markup Languaje) (Lenguaje de marcado de hipertexto). Es el lenguaje en el que se escriben los documentos para la World Wide Web.

HTTP (Hyper-Text Transfer Protocol) (Protocolo de Transferencia de Hipertextos). Es el protocolo usado para transmitir páginas HTML.

IMAP (Internet Message Access Protocol) (Protocolo de Acceso a Mensajes de Internet). Protocolo diseñado para permitir la manipulación de mailboxes remotos como si fueran locales. IMAP requiere de un servidor que haga las funciones de oficina de correos.

Intranet. Una red privada dentro de una compañía u organización que utiliza el mismo software que se encuentra en Internet, pero que es solo para uso interno.

IP address (Dirección IP). Dirección definida por el Protocolo Internet en STD 5, RFC 791. Se representa usualmente mediante notación decimal separada por puntos, e identifica unívocamente a un dispositivo dentro de una red.

LAN (Local Area Network) (Red de Area Local). Red de datos para dar servicio a un área geográfica pequeña, un edificio por ejemplo.

LRU (Least Recently Used). Los algoritmos LRU son usados para buscar las entradas menos recientemente usadas, es decir, las más antiguas en términos de acceso.

MAC, MAC Address (Media Access Control Address). Refiere a la dirección de hardware (o física) de una interface de red. Se especifica como un conjunto de seis pares de dígitos hexadecimales de la forma xx:xx:xx:xx:xx:xx.

MAN (Metropolitan Area Network) (Red de Area Metropolitana). Red de datos para dar servicio a un área geográfica mediana, por ejemplo, una ciudad.

Navegador (Browser). Aplicado normalmente a programas usados para conectarse al Internet.

NFS (Network File System). Sistema de archivos distribuido que implementa un conjunto de protocolos que definen la forma en la que un cliente accede a un archivo almacenado en un servidor de archivos (remoto) NFS.

Nodo. Normalmente se refiere a un punto de confluencia en una red, pero en ocasiones, se refiere a una única máquina en Internet.

POP (Post Office Protocol) (Protocolo de Oficina de Correos). Programa cliente que se comunica con el servidor, identifica la presencia de nuevos mensajes, solicita la entre de los mismos y utiliza al servidor como oficina despachadora de correo electrónico cuando el usuario envía una carta.

PPP (Point to Point Protocol) (Protocolo Punto a Punto). Implementación de TCP/IP por líneas seriales (como en el caso del módem). Es una versión más reciente y compleja que SLIP.

Protocolo. Descripción formal de formatos de mensaje y de reglas que dos computadoras deben seguir para intercambiar dichos mensajes.

Proxy. Una substitución de direcciones, usado para limitar la información de direcciones disponibles externamente.

Proxy Server. Un servidor que se sitúa entre la aplicación cliente, como por ejemplo un web browser, y un server real. Intercepta todos los requerimientos al server real para ver si puede resolverlos, y caso contrario, envía el requerimiento al server real, así descongestionando el tráfico en la red.

Router (direccionador). Dispositivo que distribuye tráfico entre redes. La decisión de enrutamiento se realiza en base a información de nivel de red y tablas de direccionamiento. Se emplea un router cuando dos redes que utilizan la misma capa de transporte pero tienen diferentes capas de red convergen en un punto. Por ejemplo, una conexión entre una red local Ethernet y una red pública X.25, necesita un router para convertir las tramas Ethernet a la forma que exige la red X.25.

RRDNS (Round Robin DNS). Es ni más ni menos que un servidor de nombres de dominio (DNS) que implementa un algoritmo de round robin para distribuir la carga de trabajo en un conjunto de servidores. Así, el balance se logra al resolver un nombre dado en una u otra dirección IP de un conjunto disponible.

SHTTP (Secure HTTP) (HTTP seguro). Protocolo HTTP mejorado con funciones de seguridad con clave simétrica. Los servidores SHTTP son fácilmente reconocibles por su URL, generalmente del tipo https://.

SMTP (Simple Mail Transfer Protocol) (Protocolo de Transferencia Simple de correo). Es el protocolo usado para transportar el correo a través de Internet.

SSL (Secure Sockets Layer) (Capa de Socket Segura). Protocolo que ofrece funciones de seguridad a nivel de capa de transporte para TCP.

TCP (Transmission Control Protocol) (Protocolo de control de Transmisión). Uno de los protocolos más usados en Internet. Es un protocolo de capa de transporte.

TCP/IP (Transmission Control Protocol/Internet Protocol) (Protocolo de control de Transmisión/Protocolo de Internet). Arquitectura de red desarrollada por la "Defense Advanced Research Projects Agency" (DARPA) en USA, es el conjunto de protocolos básicos de Internet o de una Intranet.

Telnet. Es el protocolo estándar de Internet para realizar un servicio de conexión desde una terminal remota. Está definido en STD 8, RFC 854 y tiene opciones adicionales descritas en muchos otros RFCs.

TTL (Time-to-live) (Tiempo de vida). En el contexto de este trabajo, es un valor asignado a las respuestas DNS, que indica el tiempo de validez que tienen las mismas. Luego de este tiempo, una respuesta DNS se considera como caduca y debiera renovarse.

UDP (User Datagram Protocol) (Protocolo de Datagramas de usuario). Es la versión no orientada a la conexión del protocolo TCP, es decir, no trata con confirmaciones de la validez de los paquetes enviados por el emisor. Este protocolo es usado para la transmisión de sonido y vídeo a través de Internet. Está diseñado para satisfacer

necesidades concretas de ancho de banda (como no reenvía los datos perdidos) y es ideal para el tráfico de voz digitalizada, pues un paquete perdido no afecta la calidad del sonido.

UML (User Mode Linux). Es una máquina virtual Linux que ejecuta sobre Linux. Técnicamente UML es un *port* de Linux a Linux. Esta tecnología permite simular dispositivos de red tales como switches, hubs, bridges y nodos terminales (computadoras), a través de máquinas virtuales. Esta herramienta es muy útil para hacer experimentos de laboratorio en los que se involucran funcionalidades de interconexión de equipos a través de una red de computadoras.

URL (Uniform Resource Locator) (Localizador Uniforme de recursos). Sistema de direccionamiento estándar para archivos y funciones de Internet, especialmente en la Word Wide Web. El URL está conformado por el servicio (por ejemplo http://) más el nombre de la computadora (por ejemplo, www3.misitio.com.ar) más el directorio y el archivo referido (por ejemplo, /documentos/abril2004.pdf).

VRRP (Virtual Router Redundancy Protocol) (Protocolo de Redundancia por Router Virtual). Protocolo especificado por la RFC 2338/3768, utilizado para lograr alta disponibilidad mediante la creación de una IP virtual (a la que acceden los clientes) y que es mapeada en el primer servidor activo disponible, de un conjunto de servidores latentes.

VRRPd (Virtual Router Redundancy Protocol daemon) (demonio del Protocolo de Redundancia por Router Virtual). Es un servicio o demonio (en el léxico de Unix/Linux), que implementa VRRP.

WAN (Wide Area Network) (Red de Área Extensa). Red de datos para dar servicio a un área geográfica amplia o global, por ejemplo Internet.

WWW, WEB o W3 (World Wide Web). Estrictamente, es la parte de Internet a la que accedemos a través del protocolo HTTP mediante los navegadores o browsers.