Práctica 3. Balanceo de carga

Duración: 3 sesiones

1. Objetivos de la práctica

En esta práctica configuraremos una red entre varias máquinas de forma que tengamos un balanceador que reparta la carga entre varios servidores finales.

El problema a solucionar es la sobrecarga de los servidores. Se puede balancear cualquier protocolo, pero dado que esta asignatura se centra en las tecnologías web, balancearemos los servidores HTTP que tenemos configurados.

De esta forma conseguiremos una infraestructura redundante y de alta disponibilidad.

2. Alternativas para realizar balanceo de carga

Aún cuando existen soluciones comerciales (dispositivos hardware específicos) para esta función, nosotros queremos usar soluciones software que resulten más baratas.

La forma más elemental de balancear la carga entre varios servidores es utilizando el DNS (como se estudió en la teoría). Este tipo de balanceo tiene la ventaja de su simplicidad y eficiencia. Lo único que se necesitan son varios servidores con distintas IP, por lo que es barato, simple y fácil de mantener.

Sin embargo, también presenta algunos inconvenientes: Por un lado, un balanceador puede tener en cuenta la carga de cada equipo y distribuir las peticiones según esas cargas, mientras que al balancear por DNS no se tiene en cuenta la carga de los equipos. Además, si un servidor se cae, el balanceador de carga lo detecta y redirige las peticiones web a los otros servidores, lo que no ocurre cuando se balancea con DNS.

Debido a esos inconvenientes, nosotros utilizaremos balanceo por software. Existen varias alternativas para balancear HTTP por software:

- HaProxy: http://haproxy.1wt.eu/
- Pound: http://www.apsis.ch/pound/
- Varnish: http://varnish-cache.org
- NginX: http://nginx.org/
- Lighty: http://www.lighttpd.net/
- Apache: http://httpd.apache.org/

Los dos primeros son balanceadores y proxy, pueden balancear cualquier tipo de tráfico. Los últimos tres son servidores web que pueden hacer estas funciones (Apache necesita usar los módulos mod_proxy o mod_proxy_balancer). Lighttpd es un servidor web liviano que soporta balanceo de carga pero no mantiene la sesión del usuario. Por último, nginX es otro servidor web liviano que sí soporta sesiones.

De todas estas opciones utilizaremos nginx configurado como proxy, y haproxy.

3. El servidor web nginx

nginx (pronunciado en inglés “engine X”) es un servidor web ligero de alto rendimiento. Lo usan muchos sitios web conocidos, como: WordPress, Hulu, GitHub, Ohloh, SourceForge, TorrentReactor y partes de Facebook.

La página principal (en español) está en http://wiki.nginx.org/NginxEs

Debido a su buen rendimiento, se usa como servidor web en lugar del Apache o IIS, aunque uno de los usos más extendidos es como balanceador de carga en un cluster web. De esta forma, el servidor con la IP pública (de cara a Internet) ejecuta el nginx, que se ocupa de redirigir el tráfico a los servidores finales. Estos servidores pueden servir su contenido web con cualquier servidor, por ejemplo, Apache.

3.1. Instalar nginx en Ubuntu Server 12.04

El proceso de instalación en Ubuntu se basa en el uso de apt-get.

Lo primero que debemos hacer es importar la clave del repositorio de software:

```
cd /tmp/
wget http://nginx.org/keys/nginx_signing.key
apt-key add /tmp/nginx_signing.key
rm -f /tmp/nginx_signing.key
```

A continuación, debemos añadir el repositorio, editando el fichero /etc/apt/sources.list y añadiendo al final las siguientes líneas:

```
echo "deb http://nginx.org/packages/ubuntu/ lucid nginx" >> /etc/apt/sources.list
echo "deb-src http://nginx.org/packages/ubuntu/ lucid nginx" >> /etc/apt/sources.list
```

Ahora ya podemos instalar el paquete del nginx:

```
apt-get update
apt-get install nginx
```

Una vez instalado, podemos proceder a su configuración como balanceador de carga.

3.2. Balanceo de carga usando nginx

nginx soporta la realización de balanceo de carga mediante la directiva proxy_pass. En nuestro caso nos interesa redirigir el tráfico a un grupo de servidores. Para definir este grupo deberemos dar un nombre al conjunto mediante la directiva upstream.

La configuración de nginx se puede hacer para definir balanceos por round-robin (alternando peticiones entre los diferentes servidores del grupo) o por IP de origen, añadiendo la directiva ip_hash.

La configuración básica de nginx no nos vale tal cual está, así es que tenemos que modificar el fichero de configuración `/etc/nginx/conf.d/default.conf`

Ahí debemos definir primero qué máquinas formarán el cluster web (y en qué puertos está a la escucha el servidor web correspondiente, en nuestro caso, los servidores Apache):

```
upstream apaches {
    server 172.16.168.130;
    server 172.16.168.131;
}
```

Y posteriormente configurar el nginx para indicarle que use ese grupo definido antes (upstream) para pasarles las peticiones. Es importante definir el upstream al principio del fichero de configuración, fuera de sección “server". También es importante para que el proxy_pass funcione correctamente que indiquemos que la conexión entre nginx y los backends sea HTTP 1.1 así como eliminar la cabecera Connection para evitar que se pase al servidor final la cabecera que indica el usuario:

```
server{
  listen 80;
  server_name m3lb;

  access_log /var/log/nginx/m3lb.access.log;
  error_log /var/log/nginx/m3lb.error.log;
  root /var/www/;

  location /
  {
    proxy_pass http://apaches;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_http_version 1.1;
    proxy_set_header Connection "";
  }

}
```

En este ejemplo hemos usado balanceo mediante el algoritmo de round-robin con la misma prioridad para todos los servidores.

Una vez que lo tenemos configurado, podemos lanzar el servicio nginx como sigue:

```
sudo service nginx restart
```

si no obtenemos ningún mensaje de error, todo está funcionando correctamente y ya podemos probar el balanceador haciendo peticiones a la IP de esta máquina. Por ejemplo, podemos usar el comando cURL de la siguiente forma:

```
curl http://172.16.168.132
curl http://172.16.168.132
```

Debería mostrar la página de inicio de cada una de las máquinas, alternativamente, lo que querrá decir que está repartiendo las peticiones entre ambos

Por otro lado, si sabemos que alguna de las máquinas finales es más potente, podemos modificar la definición del upstream para pasarle más tráfico que al resto de las del grupo. Para ello, tenemos un modificador llamado weight, al que le damos un valor numérico que indica la carga que le asignamos. Por defecto tiene el valor 1, por lo que si no modificamos ninguna máquina del grupo, todas recibirán la misma cantidad de carga:

```
upstream apaches {
    server 172.16.168.130 weight=1;
    server 172.16.168.131 weight=2;
}
```

En el ejemplo anterior, la primera máquina la suponemos menos potente o más sobrecargada, así que le hemos asignado menos carga de trabajo que a la segunda (de cada tres peticiones que lleguen, la segunda máquina atiende dos y la primera atenderá una).

Esta configuración es muy útil, pero aún así nos interesará que todas las peticiones que vengan de la misma IP se dirijan a la misma máquina servidora final. Esto es así porque si el usuario está usando una aplicación web que mantiene algún tipo de estado durante la navegación, y el balanceador lo cambia a otra máquina servidora final, puede que reciba algún error.

Para evitarlo, podemos hacer un balanceo por IP, de forma que todo el tráfico que venga de una IP se sirva durante toda la sesión por el mismo servidor final. Para ello, como hemos indicado antes, usaremos la directiva ip_hash al definir el upstream:

```
upstream apaches {
    ip_hash;
    server 172.16.168.130;
    server 172.16.168.131;
}
```

La desventaja del balanceo ip_hash es que todos los usuarios detrás de un proxy o de un NAT serán dirigidos al mismo backend, lo que puede suponer que el balanceo no sea equilibrado. Para evitar esto, los balanceadores modernos permiten balancear usando una cookie, que sí que identifica a los usuarios finales.

A partir de la versión 1.2 de nginx ya es posible utilizar conexiones con keepalive entre el nginx y los servidores finales, de forma que se realice una conexión con una persistencia de múltiples peticiones HTTP en lugar de abrir una conexión nueva cada vez.

Para especificar a nginx que use keepalive, debemos modificar nuestro upstream añadiendo la directiva keepalive y un tiempo de mantenimiento de la conexión en segundos:

```
upstream apaches {
    server 172.16.168.130;
    server 172.16.168.131;
    keepalive 3;
}
```

3.3. Opciones de configuración del nginx para establecer cómo le pasará trabajo a las máquinas servidoras finales

Hemos visto algunas opciones de configuración para las máquinas que hay en el grupo de servidores a los que redirigimos el tráfico. Sin embargo, existen otras para gestionar posibles errores o caídas de los servidores:

`weight = NUMBER`
permite especificar un peso para el servidor (por defecto es 1).

`max_fails = NUMBER`
especifica un número de intentos de comunicación erróneos en "fail_timeout" segundos para considerar al servidor no operativo (por defecto es 1, un valor de 0 lo desactivaría).

`fail_timeout = TIME`
indica el tiempo en el que deben ocurrir "max_fails" intentos fallidos de conexión para considerar al servidor no operativo. Por defecto es 10 segundos.

`down`
marca el servidor como permanentemente offline (para ser usado con ip_hash).

`backup`
reserva este servidor y sólo le pasa tráfico si alguno de los otros servidores no-backup está caído u ocupado. No es compatible con la directiva ip_hash

Ejemplos:

```
upstream backend {
  server maquina1 weight=5;
  server 127.0.0.1:8080 max_fails=3 fail_timeout=30s;
  server maquina3;
}
```

En este ejemplo, las peticiones se distribuyen mediante round-robin, pero lo hemos modificado de forma que de cada siete peticiones, cinco vayan a la maquina1 y otra a cada unas de las restantes máquinas del grupo (la segunda es el mismo balanceador pero con un servidor web adicional configurado en otro puerto). Además, si ocurre un error, la petición se pasará al siguiente servidor, hasta que se consiga servir o todos den error... También hemos establecido que el segundo servidor espere hasta tres intentos fallidos de conexión antes de dar por considerado ese servidor como no operativo, y esperará 30 segundos entre fallos.

Si en la configuración con ip_hash necesitamos apagar uno de los servidores para hacer algún tipo de mantenimiento, debemos marcarlo con la opción down:

```
upstream backend {
  ip_hash;
  server maquina1;
  server maquina2;
  server maquina3 down;
  server maquina4;
}
```

A continuación se ofrecen varias páginas de ayuda para profundizar en las posibilidades de configuración de nginx, ya sea como servidor web o como balanceador de carga:

- http://www.cyberciti.biz/tips/using-nginx-as-reverse-proxy.html
- http://nginx.org/en/docs/http/ngx_http_upstream_module.html

4. Balanceo de carga con haproxy

haproxy es un balanceador de carga y también proxy, de forma que puede balancear cualquier tipo de tráfico.

Es un software muy adecuado para repartir carga y construir una infraestructura de altas prestaciones. Una configuración básica es sencilla de hacer. Posee muchas opciones para realizar cualquier tipo de balanceo y control de los servidores finales.

Veamos cómo instalar y configurar haproxy para realizar funciones de balanceo de carga sencillas.

4.1. Instalar haproxy

Tras instalar el sistema básico, sólo tenemos que usar apt-get para instalar haproxy:

```
sudo apt-get install haproxy joe
```

4.2. Configuración básica de haproxy como balanceador de carga

Una vez instalado, debemos modificar el archivo `/etc/haproxy/haproxy.cfg` ya que la configuración que trae por defecto no nos vale. Así pues, tras consultar cuál es la IP de la máquina balanceadora, y anotar las IP de las máquinas servidoras, editamos el fichero de configuración de haproxy:

```
cd /etc/
cd haproxy/
ifconfig
sudo joe haproxy.cfg
```

Un balanceador sencillo debe escuchar tráfico en el puerto 80 y redirigirlo a alguna de las máquinas servidoras finales (debe conocer sus IP). Usemos como configuración inicial la siguiente:

```
global
      daemon
      maxconn 256

defaults
      mode http
      contimeout  4000
      clitimeout  42000
      srvtimeout  43000

frontend http-in
      bind *:80
      default_backend servers

backend servers
      server      m1 172.16.168.130:80 maxconn 32
      server      m2 172.16.168.131:80 maxconn 32
```

Vemos que nuestro balanceador espera conexiones entrantes por el puerto 80 para redirigirlas a las dos máquinas servidoras (en las que tenemos los Apache instalados y escuchando en el puerto 80).

4.3. Comprobar el funcionamiento del balanceador

Una vez salvada la configuración en el fichero, lanzamos el servicio haproxy mediante el comando:

```
sudo /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg
```

Si no nos sale ningún error o aviso, todo ha ido bien, y ya podemos comenzar a hacer peticiones a la IP del balanceador. Por ejemplo, podemos usar el comando cURL de la siguiente forma:

```
curl http://172.16.168.133
curl http://172.16.168.133
```

Debería mostrar la página de inicio de cada una de las máquinas, alternativamente, lo que querrá decir que está repartiendo las peticiones entre ambos.

4.4. Resumen

En esta sección hemos hecho una configuración muy básica de haproxy, pero completamente funcional. El programa posee muchas más opciones para realizar

cualquier tipo de balanceo y control de los servidores finales. A continuación se ofrecen varias páginas de ayuda para profundizar en las posibilidades de configuración de haproxy:

- http://code.google.com/p/haproxy-docs/
- http://haproxy.1wt.eu/download/1.4/doc/configuration.txt

Cuestiones a resolver

En esta práctica el objetivo es configurar las máquinas virtuales de forma que dos hagan de servidores web finales mientras que la tercera haga de balanceador de carga por software.

En esta práctica se llevarán a cabo, como mínimo, las siguientes tareas:

- configurar una máquina e instalarle el nginx como balanceador de carga
- configurar una máquina e instalarle el haproxy como balanceador de carga

En ambos casos, debemos hacer peticiones a la dirección IP principal y comprobar que realmente se reparte la carga.

Además, se comprobará el funcionamiento de los algoritmos de balanceo round-robin y con ponderación (en este caso supondremos que la máquina 1 tiene el doble de capacidad que la máquina 2).

Como resultado de la práctica 3 se mostrará al profesor el funcionamiento del balanceo de carga, tanto con nginx como con haproxy. En el documento a entregar se describirá cómo se ha realizado la configuración de ambas máquinas y del software.

Normas de entrega

La práctica podrá realizarse de manera individual o por grupos de hasta 2 personas.

Se entregará como un solo archivo de texto en el que se muestre la información requerida. La entrega se realizará subiendo el archivo TXT al repositorio SWAP2015 en la cuenta de github del alumno, nombrando el archivo como:

P3__nombre-apellido1-apellido2.txt

Toda la documentación y material exigidos se entregarán en la fecha indicada por el profesor. No se recogerá ni admitirá la entrega posterior de las prácticas ni de parte de las mismas.

La detección de prácticas copiadas implicará el suspenso inmediato de todos los implicados en la copia (tanto del autor del original como de quien las copió).

Las faltas de ortografía se penalizarán con hasta 1 punto de la nota de la práctica.