

# Práctica 3. Balanceo de carga

*Duración: 2 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.

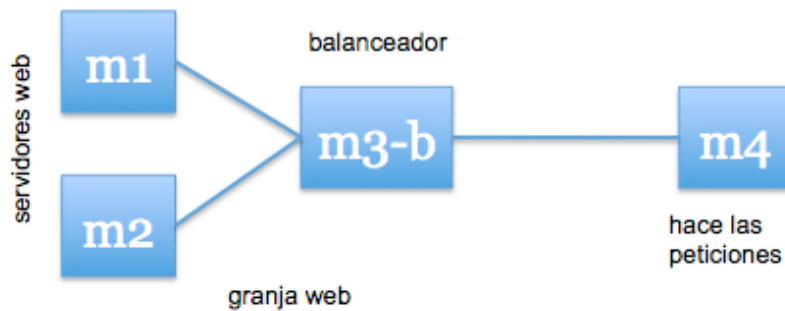
Debido a esos inconvenientes, nosotros utilizaremos balanceo por software. Existen varias alternativas para balancear HTTP por software:

- 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.

Así pues, para esta práctica, debemos crear la siguiente organización de red:



Como se puede ver, debemos tener en ejecución las máquinas servidoras finales (las que hicimos en la Práctica 2 y que ejecutan el servidor Apache).

Además, crearemos una tercera máquina en la que no debe haber ningún software que se apropie del puerto 80, ya que lo necesitará el software de balanceo para recibir las peticiones HTTP desde fuera de la granja web. Así pues, en esta máquina-3 no podemos tener instalado el Apache (o al menos, que no esté en ejecución).

Una vez instalemos y configuremos el software de balanceo (nginx o haproxy), desde una máquina externa a la granja web (puede ser un terminal en el ordenador anfitrión o puede ser una cuarta máquina virtual) ejecutaremos la herramienta curl para hacer peticiones HTTP a la IP de la máquina balanceadora (como vimos en teoría, la VIP de nuestro sistema web).

### 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

El proceso de instalación en Ubuntu se basa en el uso de apt-get. Podemos seguir el tutorial detallado en:

<https://www.liberiangeek.net/2016/07/how-to-install-nginx-webserver-on-ubuntu-16-04/>

Concretamente, para hacer la instalación, se recomienda ejecutar las siguientes órdenes:

```
sudo apt-get update && sudo apt-get dist-upgrade && sudo apt-get autoremove
```

```
sudo apt-get install nginx
sudo systemctl start 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, ya que corresponde a una funcionalidad de servidor web, así es que tenemos que modificar el fichero de configuración `/etc/nginx/conf.d/default.conf`

Debemos eliminar el contenido que tuviese antes, para crear la configuración que necesitamos. Si ese fichero no existiese, se crea nuevo con el contenido que describiremos a continuación.

Primero definimos qué máquinas formarán el cluster web (y en qué puertos está a la escucha el servidor web correspondiente, en nuestro caso, los servidores Apache). Se trata de la sección “`upstream`” de la configuración. En esta primera sección es donde vamos a indicar las IP de todos los servidores finales de nuestra granja web.

A continuación de esa sección debemos indicarle al nginx que use ese grupo definido antes (“`upstream`”) son las máquinas a las que debe repartir el tráfico. Se trata ahora de la sección “`server`”. 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 servidores finales sea HTTP 1.1 así como especificarle que debe eliminar la cabecera `Connection` (hacerla vacía) para evitar que se pase al servidor final la cabecera que indica el usuario. Así pues, el fichero de configuración debe quedar con el contenido siguiente:

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

server{
    listen 80;
    server_name balanceador;

    access_log /var/log/nginx/balanceador.access.log;
    error_log /var/log/nginx/balanceador.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 systemctl start nginx
```

si no obtenemos ningún mensaje de error, todo está funcionando correctamente y ya podemos probar la configuración haciendo peticiones a la IP de esta máquina balanceador. 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

Si al lanzarlo hemos obtenido un error o no está funcionando como balanceador, es posible que haya que eliminar la línea que configura nginx como servidor web en el archivo `/etc/nginx/nginx.conf` (está hacia el final del archivo, y es suficiente con comentarla).

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/nginx\\_http\\_upstream\\_module.html](http://nginx.org/en/docs/http/nginx_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
```

### 4.2. Configuración básica de haproxy como balanceador

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 nano 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
    timeout 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 (como hicimos en el caso del nginx). 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>

## 5. Someter a una alta carga el servidor balanceado

Para medir el rendimiento de un servidor necesitaremos una herramienta que ejecutar en los clientes para crear una carga HTTP específica (habitualmente alta). En algunos casos, las medidas se realizan con benchmarks como SPECweb o WebStone para simular un número determinado de clientes. Puesto que el número de usuarios de un servidor web puede ser del orden de los millones de usuarios, queda claro que simular un número pequeño de clientes no es realista. Por esta razón, debemos ser cuidadosos al elegir las herramientas a usar, y al ejecutarlas con los parámetros adecuados.

Existen diversas herramientas para comprobar el rendimiento de servidores web. Las hay basadas en interfaz de línea de comandos y de interfaz gráfica. Entre las más utilizadas destacan:

- Apache Benchmark
- siege
- httpperf
- OpenSTA
- JMeter
- openwebload
- the grinder

Con estas herramientas podemos analizar el rendimiento de servidores Apache, Internet Information Services (IIS), nginx, etc.

Lo habitual es usar programas de línea de comandos que sobrecarguen lo mínimo posible las máquinas que estamos usando. En esta práctica usaremos la herramienta Apache Benchmark para comprobar el rendimiento de nuestra granja web recién configurada: <http://httpd.apache.org/docs/2.2/programs/ab.html>

Es conveniente ejecutar los benchmark en otra máquina diferente a las que forman parte de la granja web (servidores web o balanceador), de forma que ambos procesos no consuman recursos de la misma máquina, ya que veríamos un menor rendimiento.

Como se ha comentado, Apache Benchmark (ab) es una utilidad que se instala junto con el servidor Apache y permite comprobar el rendimiento de cualquier servidor web. Para utilizarlo debemos entrar en un terminal y ejecutar el comando "ab" como sigue:

```
ab -n 1000 -c 10 http://192.168.2.121/index.html
```

Los parámetros indicados en la orden anterior le indican al benchmark que solicite la página con dirección `http://192.168.2.121/index.html` 1000 veces (-n 1000 indica el número de peticiones) y hacer esas peticiones concurrentemente de 10 en 10 (-c 10 indica el nivel de concurrencia).

ab no simula con total fidelidad el uso del sitio web que pueden hacer los usuarios habitualmente. En realidad pide el mismo recurso (misma página) repetidamente, pero para aplicar una carga alta a nuestra granja web será suficiente.

## 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:**

1. configurar una máquina e instalar el nginx como balanceador de carga



2. configurar una máquina e instalar el haproxy como balanceador de carga
3. someter a la granja web a una alta carga, generada con la herramienta Apache Benchmark, teniendo primero nginx y después haproxy.

En las tareas 1 y 2 debemos hacer peticiones a la dirección IP del balanceador y comprobar que realmente se reparte la carga. Para ello, el `index.html` en las máquinas finales deben ser diferentes para ver cómo las respuestas que recibimos al hacer varias peticiones son diferentes (eso indicará que el balanceador deriva tráfico a las máquinas servidoras finales).

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).

En la tarea 3 debemos usar la herramienta `ab` para someter a una carga muy alta (gran número de peticiones y con alta concurrencia) a la granja web, primero estando nginx como balanceador, y a continuación estando haproxy como balanceador. Como resultado, se debe realizar una comparación de los tiempos medios de servicio entre ambos balanceadores, para poder determinar cuál funciona mejor.

Adicionalmente, y como tarea opcional para conseguir una mayor nota en esta práctica, se propone el uso de algún otro software de balanceo diferente a los dos explicados en este guión (por ejemplo Pound).

Como resultado de la práctica 3 **se mostrará** al profesor el funcionamiento del balanceo de carga con los diferentes balanceadores. En el documento a entregar se describirá cómo se ha realizado la configuración de ambas máquinas y del software, así como los resultados de la comparación de balanceadores al someterlos a la misma alta carga de peticiones HTTP.

## Normas de entrega

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

Se entregará como un archivo de texto en el que se muestre la información requerida. También se puede utilizar la sintaxis de Markdown para conseguir una mejor presentación e incluso integrar imágenes o capturas de pantalla. La entrega se realizará subiendo los archivos necesarios al repositorio SWAP en la cuenta de github del alumno, a una carpeta llamada "practica3".

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.

## Anexo

Para comprobar si hay algún servicio ocupando el puerto 80, podemos hacer uso de la siguiente orden:

```
netstat -tulpn | grep :80
```

y para comprobar qué conexiones hay activas y desde qué IP, podemos utilizar:

```
netstat -an | grep :80 | sort  
netstat | grep http | wc -l
```