

SERVIDORES WEB DE ALTAS PRESTACIONES

Práctica 3: Balanceo de carga en un sitio Web



ugr

Universidad
de **Granada**

Autor: Sergio Aguilera Ramírez
Curso 2019 - 2020

Índice

1. INSTALACIÓN Y CONFIGURACIÓN DE NGINX	2
2. INSTALACIÓN Y CONFIGURACIÓN DE HAPROXY	4
3. BECHMARK APACHE	5
4. INSTALACIÓN Y CONFIGURACIÓN POUND	8
5. BIBLIOGRAFÍA	9

1. INSTALACIÓN Y CONFIGURACIÓN DE NGINX

En primer lugar, vamos a instalar una máquina virtual desde cero, ya que para la instalación tanto de nginx como haproxy (estos balanceadores se verán en los siguientes apartados), el puerto 80 debe estar desocupado. Asimismo, esta máquina cuyo nombre de servidores es balanceador, se le ha asignado la dirección IP 192.168.56.110 (añadido con netplan - Práctica 1 -).

Una vez se creada la máquina virtual, instalamos el servicio nginx mediante las órdenes:

1. `sudo apt - get update &&sudo apt - get dist - upgrade &&sudo apt - get autoremove`
2. `sudo apt - get install nginx`

Tras haber instalado nginx, podemos comprobar que el servicio esta activo mediante el comando: `sudo systemctl start nginx.service`. La siguiente imagen muestra que el estado del servicio esta activo:

```
aguilera4@balanceador:~$ sudo systemctl status nginx.service
● nginx.service - A high performance web server and a reverse proxy server
   Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2020-04-03 15:29:09 UTC; 24s ago
     Docs: man:nginx(8)
   Process: 1557 ExecStart=/usr/sbin/nginx -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
   Process: 1546 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
  Main PID: 1562 (nginx)
    Tasks: 2 (limit: 1108)
   CGroup: /system.slice/nginx.service
           └─1562 nginx: master process /usr/sbin/nginx -g daemon on; master_process on;
             └─1564 nginx: worker process

Apr 03 15:29:09 balanceador systemd[1]: Starting A high performance web server and a reverse proxy server: nginx.service.
Apr 03 15:29:09 balanceador systemd[1]: nginx.service: Failed to parse PID from file /run/nginx.pid: Invalid PID '0'.
Apr 03 15:29:09 balanceador systemd[1]: Started A high performance web server and a reverse proxy server: nginx.service.
lines 1-15/15 (END)
```

Tras instalar el servicio de nginx, procedemos a configurar este balanceador, editando el archivo situado en la ruta `etc/nginx/conf.d/default.conf`, para así poder fijar los servidores a utilizar, en mi caso tuve que crear este archivo ya que no existía. La configuración de este archivo quedaría:

```
upstream servidoresSWAP {
    server 192.168.56.101;
    server 192.168.56.102;
}

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://servidoresSWAP;
        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 "";
    }
}
```

Esta configuración esta basada en el algoritmo round-robin en la que las peticiones son repartidas igualmente entre ambos servidores, es decir, una petición la gestiona la m1 y otra la m2 y vuelve a comenzar el ciclo. Esto podemos comprobarlo haciendo uso de la herramienta curl, si ejecutamos dos veces seguidas el comando `curl http://IP_maquina3` nos muestra una vez el index.html de la m1 y luego el de la m2.

```

aguilera4@balanceador:~$ curl http://192.168.56.110
<HTML>
  <BODY>
    Web de ejemplo de la m1 de Aguilera4 para SWAP
  </BODY>
</HTML>
aguilera4@balanceador:~$ curl http://192.168.56.110
<HTML>
  <BODY>
    Web de ejemplo de la m2 de Aguilera4 para SWAP
  </BODY>
</HTML>
aguilera4@balanceador:~$

```

Además, en mi caso tuve que comentar la orden *include /etc/nginx/sites-enabled/*;*, situada en el fichero que se encuentra en la ruta *'/etc/nginx/nginx.conf'*, ya que no estaba funcionando correctamente como balanceador.

```

##
# Gzip Settings
##

gzip on;

# gzip_vary on;
# gzip_proxied any;
# gzip_comp_level 6;
# gzip_buffers 16 8k;
# gzip_http_version 1.1;
# gzip_types text/plain text/css application/json application/javascript text/xml applica
n/xml application/xml+rss text/javascript;

##
# Virtual Host Configs
##

include /etc/nginx/conf.d/*.conf;
#include /etc/nginx/sites-enabled/*;
}

#mail {
#   # See sample authentication script at:
#   # http://wiki.nginx.org/ImapAuthenticateWithApachePhpScript
#
#   # auth_http localhost/auth.php;
#   # pop3_capabilities "TOP" "USER";
#   # imap_capabilities "IMAP4rev1" "UIDPLUS";
#
#   server {
#       listen     localhost:110;
#       protocol   pop3;
#       proxy      on;
#   }
# }

```

53,1-8 86%

Por otro lado, podemos indicar en el fichero de configuración de nginx que una máquina es mas potente que otra. Esto se lleva a cabo, indicando diferentes pesos a los servidores, esto se hace para enviar un mayor número de peticiones a una máquina, ya que suponemos que esta máquina tendrá la capacidad de soportar un mayor número de peticiones. En nuestro caso vamos a asumir que la m1 es mas potente que la m2 por lo que la configuración quedaría:

```

upstream servidoresSWAP {
    server 192.168.56.101 weight=2;
    server 192.168.56.102 weight=1;
}

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://servidoresSWAP;
        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 "";
    }
}

```

"/etc/nginx/conf.d/default.conf" 29L, 520C 3,31-38 All

Ahora al ejecutar el comando `curl http://IP_maquina3`, de cada 3 peticiones, 2 serán gestionadas por la m1 y solo una por la m2.

```
aguilera4@balanceador:~$ curl http://192.168.56.110
<HTML>
  <BODY>
    Web de ejemplo de la m2 de Aguilera4 para SWAP
  </BODY>
</HTML>
aguilera4@balanceador:~$ curl http://192.168.56.110
<HTML>
  <BODY>
    Web de ejemplo de la m1 de Aguilera4 para SWAP
  </BODY>
</HTML>
aguilera4@balanceador:~$ curl http://192.168.56.110
<HTML>
  <BODY>
    Web de ejemplo de la m1 de Aguilera4 para SWAP
  </BODY>
</HTML>
aguilera4@balanceador:~$
```

2. INSTALACIÓN Y CONFIGURACIÓN DE HAPROXY

En este segundo apartado, vamos a proceder con la instalación de haproxy. Para ello, en la m3 ejecutamos el comando de instalación `sudo apt - get install haproxy`. Una vez instalado, debemos configurar el archivo situado en la ruta `'/etc/haproxy/haproxy.cfg'` para indicar el puerto que este balanceador va a utilizar y cuales son los servidores a los que tiene que enviar las peticiones. El archivo configurado quedaría de la siguiente manera:

```
# Default SSL material locations
ca-base /etc/ssl/certs
crt-base /etc/ssl/private

# Default ciphers to use on SSL-enabled listening sockets.
# For more information, see ciphers(1SSL). This list is from:
# https://hynek.me/articles/hardening-your-web-servers-ssl-ciphers/
# An alternative list with additional directives can be obtained from
# https://mozilla.github.io/server-side-tls/ssl-config-generator/?server=haproxy
ssl-default-bind-ciphers ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:RSA+
AESGCM:RSA+AES:!aNULL:!MD5:!DSS
ssl-default-bind-options no-ssl-v3

defaults
    log          global
    mode         http
    option       httplog
    option       dontlognull
    timeout connect 5000
    timeout client  50000
    timeout server  50000
    errorfile 400 /etc/haproxy/errors/400.http
    errorfile 403 /etc/haproxy/errors/403.http
    errorfile 408 /etc/haproxy/errors/408.http
    errorfile 500 /etc/haproxy/errors/500.http
    errorfile 502 /etc/haproxy/errors/502.http
    errorfile 503 /etc/haproxy/errors/503.http
    errorfile 504 /etc/haproxy/errors/504.http

frontend http-in
    bind *:80
    default_backend servidoresSWAP

backend servidoresSWAP
    server m1 192.168.56.101:80 maxconn 32
    server m2 192.168.56.102:80 maxconn 32
-- INSERT --
```

Una vez instalado y configurado el balanceador levantamos el servicio. Una cosa a tener en cuenta es que para poder iniciar haproxy debemos parar el servicio que teníamos activo para nginx (tendríamos que hacer lo mismo cuando queramos activar nginx). A través del comando `sudo service haproxy restart` podemos comprobar si está activo.

```
aguilera4@balanceador:~$ sudo systemctl status haproxy.service
● haproxy.service - HAProxy Load Balancer
   Loaded: loaded (/lib/systemd/system/haproxy.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2020-04-01 16:14:40 UTC; 24s ago
     Docs: man:haproxy(1)
           file:/usr/share/doc/haproxy/configuration.txt.gz
  Main PID: 1397 (haproxy)
    Tasks: 2 (limit: 1108)
   CGroup: /system.slice/haproxy.service
           └─1397 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid
             1399 /usr/sbin/haproxy -Ws -f /etc/haproxy/haproxy.cfg -p /run/haproxy.pid

Apr 01 16:14:40 balanceador systemd[1]: Starting HAProxy Load Balancer...
Apr 01 16:14:40 balanceador systemd[1]: Started HAProxy Load Balancer.
```

De igual forma que se podía configurar nginx para indicar que una máquina es mas potente que otra fijando pesos a cada servicio se puede hacer también para el balanceador haproxy. Asimismo, asumimos que la m1 es mas potente que la m2, y asignamos los pesos correspondientes.

```
defaults
    log      global
    mode     http
    option   httplog
    option   dontlognull
    timeout  connect 5000
    timeout  client  50000
    timeout  server  50000
    errorfile 400 /etc/haproxy/errors/400.http
    errorfile 403 /etc/haproxy/errors/403.http
    errorfile 408 /etc/haproxy/errors/408.http
    errorfile 500 /etc/haproxy/errors/500.http
    errorfile 502 /etc/haproxy/errors/502.http
    errorfile 503 /etc/haproxy/errors/503.http
    errorfile 504 /etc/haproxy/errors/504.http

frontend http-in
    bind *:80
    default_backend servidoresSWAP

backend servidoresSWAP
    server m1 192.168.56.101:80 maxconn 32 weight 2
    server m2 192.168.56.102:80 maxconn 32 weight 1
"/etc/haproxy/haproxy.cfg" 45L, 1459C                                     44,48-55      Bot
```

Tras configurar la ponderación, comprobamos el correcto funcionamiento, como sabemos cada 3 peticiones solicitadas al balanceador, dos de ellas las gestionará m1 y la restante será gestionada por m2:

```
aguilera4@balanceador:~$ curl http://192.168.56.110
<HTML>
  <BODY>
    Web de ejemplo de la m1 de Aguilera4 para SWAP
  </BODY>
</HTML>
aguilera4@balanceador:~$ curl http://192.168.56.110
<HTML>
  <BODY>
    Web de ejemplo de la m1 de Aguilera4 para SWAP
  </BODY>
</HTML>
aguilera4@balanceador:~$ curl http://192.168.56.110
<HTML>
  <BODY>
    Web de ejemplo de la m2 de Aguilera4 para SWAP
  </BODY>
</HTML>
```

3. BECHMARK APACHE

En esta última sección, vamos a poner a prueba los balanceadores instalados y configurados anteriormente. Para ello, vamos a someter estas máquinas a un bechmark, en concreto Apache Benchmark, esto se realiza mediante el comando `ab -n 10000 -c 10 http://192,168,56,110/`. Este comando indica que vamos a enviar 10000 peticiones al balanceador y establecemos que se pueden realizar peticiones concurrentes de 10 en 10. (Los resultados obtenidos se muestran en los apartados siguientes)

Por otro lado, tras analizar los resultados obtenidos, note que había una lenta respuesta por parte de los servidores, ya que estos responden alrededor de 270 peticiones por segundo, lo cual comparando con lo obtenido por otros compañeros me parecia bastante bajo, le escribí un correo para preguntarle sobre esto y me dijo que era normal, que podía depender de mi CPU. Asimismo, estos resultados son obtenidos en la partición del sistema operativo windows 10 de mi máquina (en windows esta ejecutado desde el bash de linux), por lo que decidí comprobar si pasaba lo mismo en la partición del sistema operativo Linux de mi máquina. Tras realizar los benchmark en esta segunda partición de mi máquina, los resultados obtenidos fueron totalmente diferentes, donde los servidores son capaces de dar respuesta alrededor de 2000 peticiones por segundo. A mi entender no le encuentro mucho sentido a lo que ocurre, lo que supongo es que windows relentiza las peticiones o algo relacionado con esa idea.

Sistema operativo Windows 10

Salida Benchmark Nginx

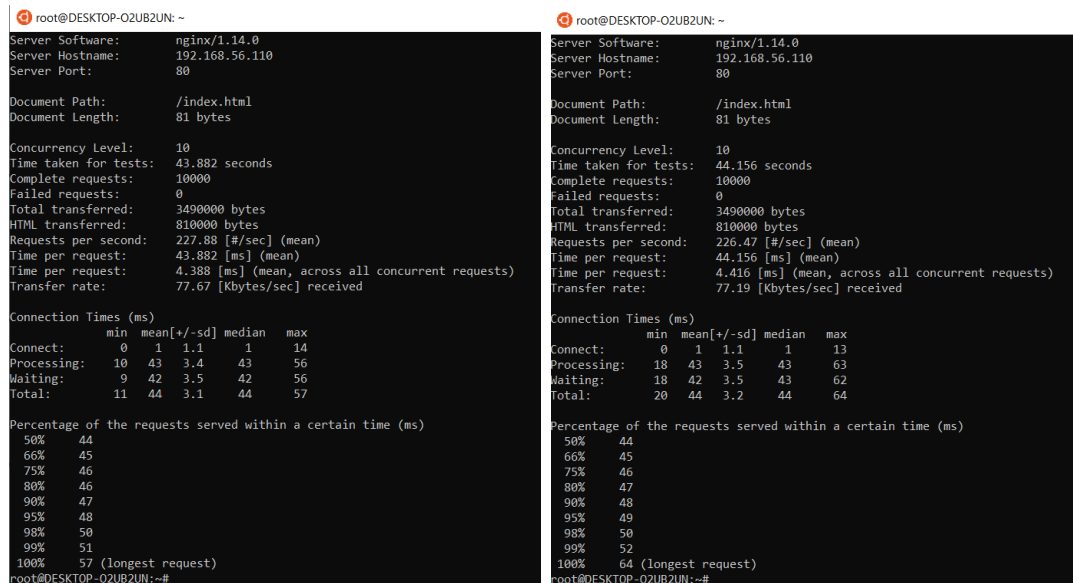


Figura 1: Imagen de la izquierda ->Nginx sin ponderación | Imagen de la derecha ->Nginx con ponderación

Salida Benchmark Haproxy

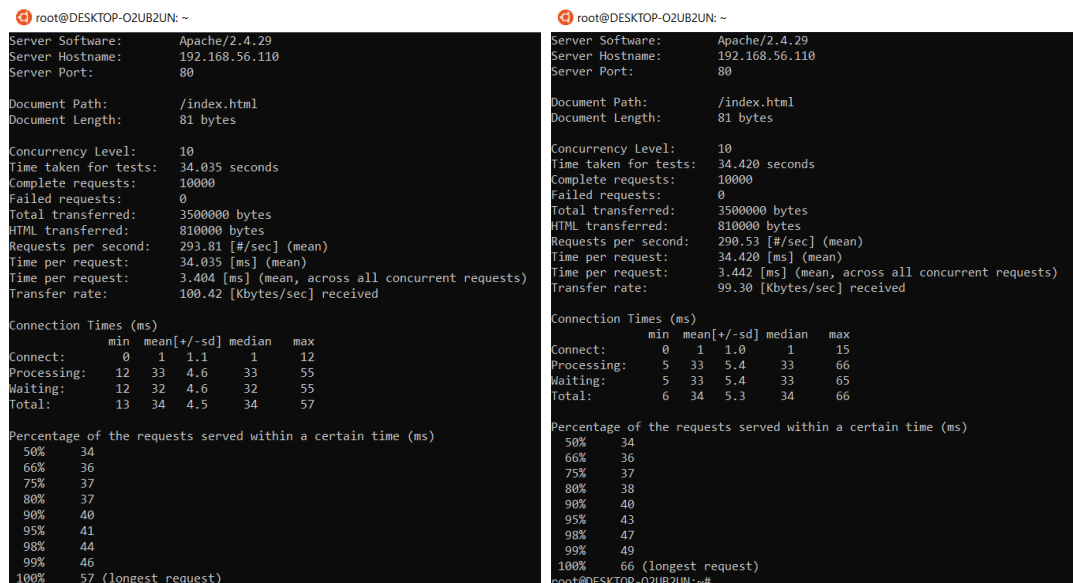
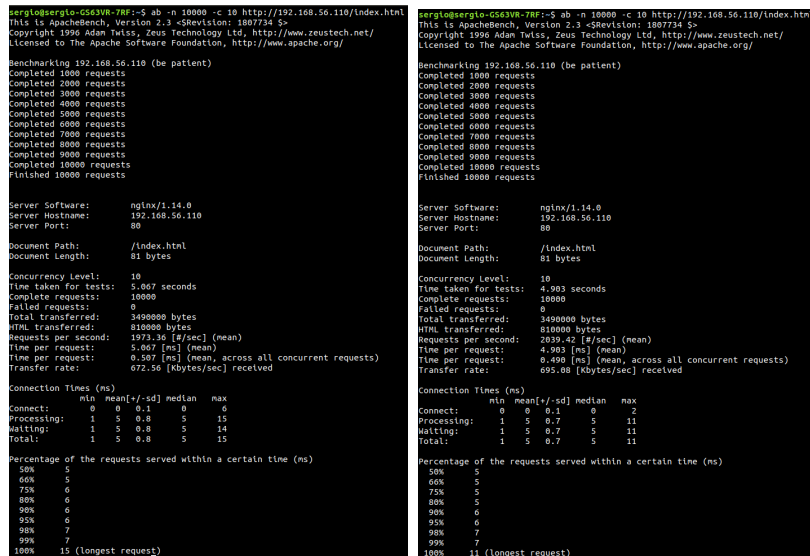


Figura 2: Imagen de la izquierda ->Haproxy sin ponderación | Imagen de la derecha ->Haproxy con ponderación

Comparando los tiempos obtenidos por estos balanceadores, podemos concluir que haproxy es más eficiente, ya que obtiene un tiempo más bajo que Nginx. Esto se debe ya que como podemos ver los tiempos de conexión, procesado y espera es menor, y además presenta una desviación típica mas baja.

Sistema operativo Ubuntu 18.04

Salida Benchmark Nginx



```
sergio@sergio-GS63VR-76P:~$ ab -n 10000 -c 10 http://192.168.56.110/index.html
This is ApacheBench, Version 2.3 <Revision: 1807734>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.56.110 (be patient)
Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
Completed 10000 requests
Finished 10000 requests

Server Software:      nginx/1.14.0
Server Hostname:      192.168.56.110
Server Port:          80
Document Path:        /index.html
Document Length:      81 bytes
Concurrency Level:    10
Time taken for tests:  5.067 seconds
Complete requests:    10000
Failed requests:       0
Total transferred:    3490000 bytes
HTML transferred:     810000 bytes
Requests per second:  1973.36 [#/sec] (mean)
Time per request:     5.067 [ms] (mean)
Time per request:     0.507 [ms] (mean, across all concurrent requests)
Transfer rate:        672.56 [Kbytes/sec] received

Connection Times (ms)
              min      mean[+/-sd] median   max
Connect:     0   0  0.1   0       6
Processing:   1   5  0.8   5      15
Waiting:     1   5  0.8   5      14
Total:       1   5  0.8   5      15

Percentage of the requests served within a certain time (ms)
 50%    5
 60%    5
 75%    6
 80%    6
 90%    6
 95%    6
 98%    7
 99%    7
100%   15 (longest request)

sergio@sergio-GS63VR-76P:~$ ab -n 10000 -c 10 http://192.168.56.110/index.html
This is ApacheBench, Version 2.3 <Revision: 1807734>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.56.110 (be patient)
Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
Completed 10000 requests
Finished 10000 requests

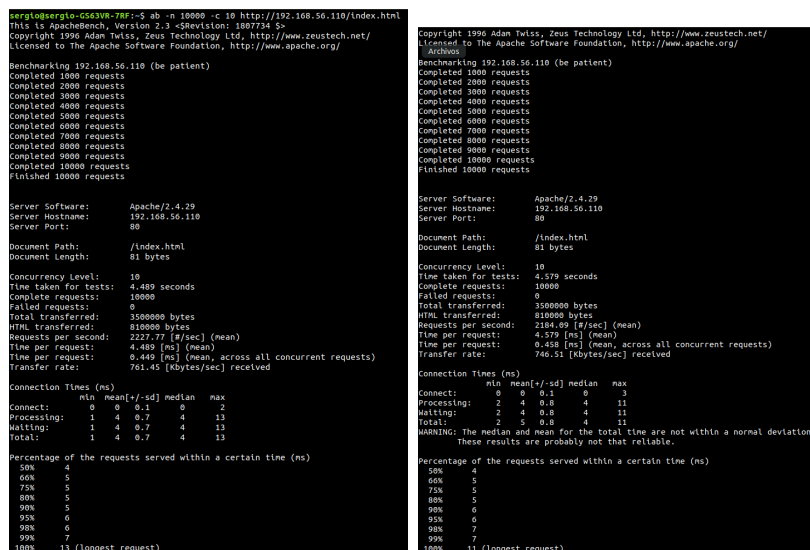
Server Software:      nginx/1.14.0
Server Hostname:      192.168.56.110
Server Port:          80
Document Path:        /index.html
Document Length:      81 bytes
Concurrency Level:    10
Time taken for tests:  4.903 seconds
Complete requests:    10000
Failed requests:       0
Total transferred:    3490000 bytes
HTML transferred:     810000 bytes
Requests per second:  2039.42 [#/sec] (mean)
Time per request:     4.903 [ms] (mean)
Time per request:     0.490 [ms] (mean, across all concurrent requests)
Transfer rate:        695.08 [Kbytes/sec] received

Connection Times (ms)
              min      mean[+/-sd] median   max
Connect:     0   0  0.1   0       2
Processing:   1   5  0.7   5      11
Waiting:     1   5  0.7   5      11
Total:       1   5  0.7   5      11

Percentage of the requests served within a certain time (ms)
 50%    5
 60%    5
 75%    5
 80%    5
 90%    6
 95%    6
 98%    7
 99%    7
100%   11 (longest request)
```

Figura 3: Imagen de la izquierda ->Nginx sin ponderación | Imagen de la derecha ->Nginx con ponderación

Salida Benchmark Haproxy



```
sergio@sergio-GS63VR-76P:~$ ab -n 10000 -c 10 http://192.168.56.110/index.html
This is ApacheBench, Version 2.3 <Revision: 1807734>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.56.110 (be patient)
Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
Completed 10000 requests
Finished 10000 requests

Server Software:      Apache/2.4.29
Server Hostname:      192.168.56.110
Server Port:          80
Document Path:        /index.html
Document Length:      81 bytes
Concurrency Level:    10
Time taken for tests:  4.489 seconds
Complete requests:    10000
Failed requests:       0
Total transferred:    3500000 bytes
HTML transferred:     810000 bytes
Requests per second:  2227.77 [#/sec] (mean)
Time per request:     4.489 [ms] (mean)
Time per request:     0.449 [ms] (mean, across all concurrent requests)
Transfer rate:        761.45 [Kbytes/sec] received

Connection Times (ms)
              min      mean[+/-sd] median   max
Connect:     0   0  0.1   0       2
Processing:   1   4  0.7   4      13
Waiting:     1   4  0.7   4      13
Total:       1   4  0.7   4      13

Percentage of the requests served within a certain time (ms)
 50%    4
 60%    5
 75%    5
 80%    5
 90%    5
 95%    6
 98%    6
 99%    7
100%   13 (longest request)

sergio@sergio-GS63VR-76P:~$ ab -n 10000 -c 10 http://192.168.56.110/index.html
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.56.110 (be patient)
Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
Completed 10000 requests
Finished 10000 requests

Server Software:      Apache/2.4.29
Server Hostname:      192.168.56.110
Server Port:          80
Document Path:        /index.html
Document Length:      81 bytes
Concurrency Level:    10
Time taken for tests:  4.579 seconds
Complete requests:    10000
Failed requests:       0
Total transferred:    3500000 bytes
HTML transferred:     810000 bytes
Requests per second:  2184.09 [#/sec] (mean)
Time per request:     4.579 [ms] (mean)
Time per request:     0.458 [ms] (mean, across all concurrent requests)
Transfer rate:        746.51 [Kbytes/sec] received

Connection Times (ms)
              min      mean[+/-sd] median   max
Connect:     0   0  0.1   0       3
Processing:   2   4  0.8   4      11
Waiting:     2   4  0.8   4      11
Total:       2   4  0.8   4      11
WARNING: The median and mean for the total time are not within a normal deviation
These results are probably not that reliable.

Percentage of the requests served within a certain time (ms)
 50%    5
 60%    5
 75%    5
 80%    5
 90%    6
 95%    6
 98%    7
 99%    7
100%   11 (longest request)
```

Figura 4: Imagen de la izquierda ->Haproxy sin ponderación | Imagen de la derecha ->Haproxy con ponderación

Tras ejecutar los benchmark de Apache en la partición de Ubuntu vemos como los tiempos disminuyen considerablemente, tardando entre 4 y 6 segundos. Asimismo, seguimos obteniendo que el balanceador haproxy obtiene mejores resultados que Nginx.

4. INSTALACIÓN Y CONFIGURACIÓN POUND

Como ejercicio opcional vamos a instalar, configurar y probar el balanceador Pound sobre el mismo benchmark utilizado en los apartados anteriores.

Para su instalación descargaremos el paquete de la ruta `http://ftp.tecnoera.com/ubuntu/pool/universe/p/` mediante el comando:

```
wget http://ftp.tecnoera.com/ubuntu/pool/universe/p/pound/pound2.7-1.3_amd64.deb
```

Una vez descargado el paquete, lo instalamos a través del siguiente comando (donde la opción `-i` indica la instalación del paquete):

```
dpkg -i pound2.7-1.3_amd64.deb
```

Reiniciamos el servicio de pound (`sudo systemctl restart pound.service`) y comprobamos que el servicio esta disponible (`sudo systemctl status pound.service`):

```
aguilera4@balanceador:~$ sudo systemctl status pound.service
[sudo] password for aguilera4:
● pound.service - LSB: reverse proxy and load balancer
   Loaded: loaded (/etc/init.d/pound; generated)
   Active: active (exited) since Mon 2020-04-06 14:42:44 UTC; 1h 5min ago
     Docs: man:systemd-sysv-generator(8)
   Process: 1489 ExecStop=/etc/init.d/pound stop (code=exited, status=0/SUCCESS)
   Process: 1510 ExecStart=/etc/init.d/pound start (code=exited, status=0/SUCCESS)

Apr 06 14:42:44 balanceador systemd[1]: Stopped LSB: reverse proxy and load balancer.
Apr 06 14:42:44 balanceador systemd[1]: Starting LSB: reverse proxy and load balancer...
Apr 06 14:42:44 balanceador pound[1510]: * Starting reverse proxy and load balancer pound
Apr 06 14:42:44 balanceador pound[1510]: starting...
```

Figura 5: Confirmación servicio Pound

Para finalizar la instalación de este balanceador debemos modificar el archivo de configuración situado en `/etc/pound/` y establecer los servidores a donde se enviara las peticiones:

```
## Logging: (goes to syslog by default)
## 0 no logging
## 1 normal
## 2 extended
## 3 Apache-style (common log format)
LogLevel 1

## check backend every X secs:
Alive 30

## use hardware-acceleration card supported by openssl(1):
#SSL Engine "chuo"

# poundctl control socket
Control "/var/run/pound/poundctl.socket"

#####
## listen, redirect and ... to:

## redirect all requests on port 8080 ("ListenHTTP") to the local webserver (see "Service" below):
ListenHTTP
    Address 192.168.56.110
    _Port 80

    ## allow PUT and DELETE also (by default only GET, POST and HEAD)?:
    xHTTP 0

    Service
        BackEnd
            Address 192.168.56.101
            Address 192.168.56.102
            Port 80
        End
    End
End

36,1-8 Bot
```

Figura 6: Configuración pound

Por último, tras ejecutar el benchmark utilizado para los demás balanceadores obtenemos el resultado mostrado en la figura 7, donde el tiempo de respuesta es un poco mayor que haproxy (en el sistema operativo windows).

```

Finished 10000 requests

Server Software:      Apache/2.4.29
Server Hostname:      192.168.56.110
Server Port:          80

Document Path:        /
Document Length:      81 bytes

Concurrency Level:     10
Time taken for tests:  36.121 seconds
Complete requests:     10000
Failed requests:       0
Total transferred:     3500000 bytes
HTML transferred:      810000 bytes
Requests per second:   276.85 [#/sec] (mean)
Time per request:      36.121 [ms] (mean)
Time per request:      3.612 [ms] (mean, across all concurrent requests)
Transfer rate:         94.63 [Kbytes/sec] received

Connection Times (ms)
      min  mean[+/-sd] median  max
Connect:    0    1  20.0     0   1000
Processing:  4   35   8.7    36    80
Waiting:    4   35   8.7    36    80
Total:      5   36  21.8    37  1041

Percentage of the requests served within a certain time (ms)
 50%    37
 66%    40
 75%    41
 80%    42
 90%    45
 95%    46
 98%    49
 99%    50
100%   1041 (longest request)

```

Figura 7: Configuración pound

5. BIBLIOGRAFÍA

<https://www.rosehosting.com/blog/how-to-install-nginx-on-ubuntu-16-04/>

<https://clouding.io/hc/es/articles/360010289000-Balancear-servicio-web-con-HAProxy-en-Ubuntu-18-04>

<http://ftp.tecnoera.com/ubuntu/pool/universe/p/pound/>

Especificaciones del PC: CPU intel core i7 7th generación. (MSI GS63VR 7RF Stealth Pro)