Servidores Web de Altas Prestaciones

Práctica 3

Autor: Juan Ocaña Valenzuela

En esta práctica se deben cumplir los siguientes objetivos:

- Configurar una máquina e instalar nginx como balanceador de carga.
- Configurar una máquina e instalar haproxy como balanceador de carga.
- Someter a la granja web a una alta carga, generada con la herramienta Apache Benchmark, teniendo primero nginx y después haproxy.

Configurar una máquina e instalar nginx como balanceador de carga

Se ha creado una nueva máquina Ubuntu Server 10.04, M3, destinada a actuar como balanceador de carga entre nuestra granja web, compuesta de momento por M1 y M2.

En ella instalamos nginx con el comando `sudo apt install nginx`. Para configurarlo como balanceador de carga y no como servidor web debemos primero indicar el `upstream` del servidor, editando el archivo `/etc/nginx/conf.d/default.conf`. En nuestro caso no existe, así que lo creamos e indicamos la siguiente información:

```
upstream servidoresSWAP {
        server 192.168.56.103;
        server 192.168.56.101;
        keepalive 3;
}

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" 24L, 509C                    4,13-20       Todo
```

La IP de M1 es `192.168.56.103` y la de M2 es `192.168.56.101`.

Para hacer efectivos los cambios, reiniciamos el servicio con `sudo systemctl nginx restart`. No obstante, al intentar acceder al servidor nginx a través de su IP, `192.168.56.104`, nos sigue sirviendo su contenido. Para solucionar esto debemos eliminar la línea que configura nginx como servidor web en `/etc/nginx/nginx.conf`.

```
        # 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 applicatio
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;
#       }
#
#       server {
#               listen     localhost:143;
#               protocol   imap;
#               proxy      on;
#       }
#}
-- VISUAL --                                                38        62,39-46    Final
```

Ahora mismo, si accedemos desde un navegador o descargamos el contenido del servidor M3 mediante `curl`, debería servirnos el contenido de M1 y M2 por turnos, ya que ambas máquinas tienen el mismo peso.

Podemos observarlo aquí:

```
patchispatch@m1:~$ curl http://192.168.56.104

<!DOCTYPE html>
<html>
        <head>
                M1
        </head>

        <body>
                <h1>Este es el index de M1</h1>
        </body>
</html>
patchispatch@m1:~$ curl http://192.168.56.104

<!DOCTYPE html>
<html>
        <head>
                M2
        </head>

        <body>
                <h1>Este es el index de M2</h1>
        </body>
</html>
patchispatch@m1:~$ _
```

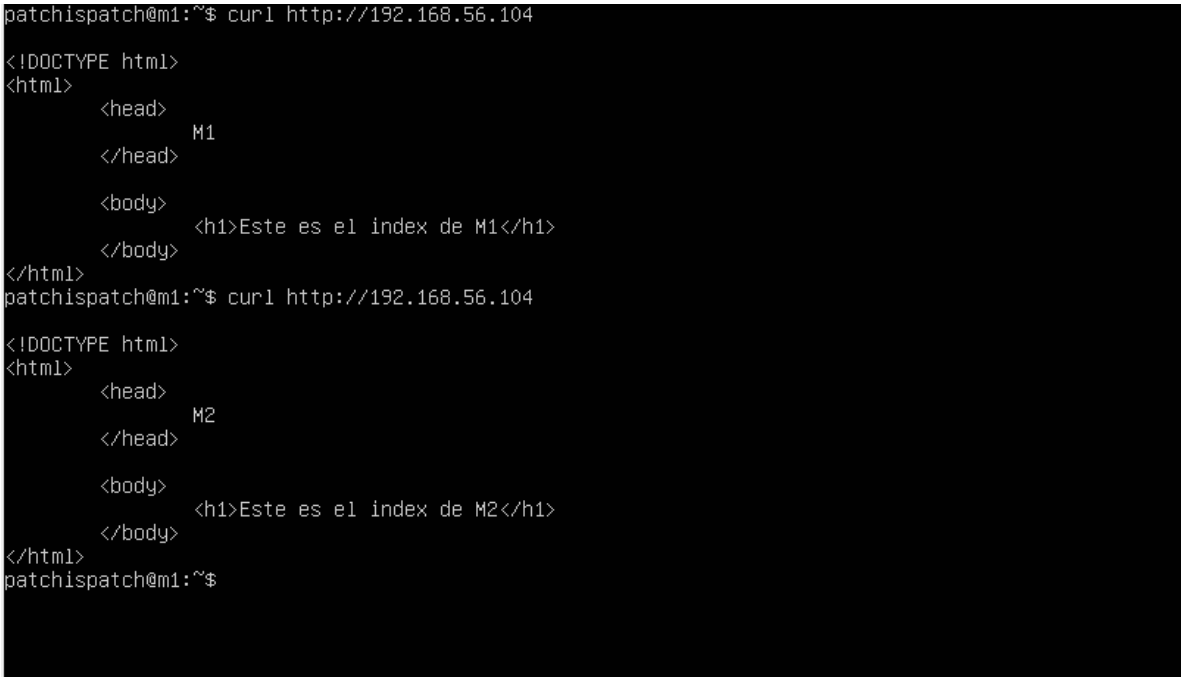
Configurar una máquina e instalar haproxy como balanceador de carga

Ya tenemos la máquina M3, por lo que instalamos haproxy con el comando `sudo apt install haproxy`. Para que todo funcione correctamente, desactivamos nginx con `sudo systemctl stop nginx`.

Para configurar haproxy como un balanceador de carga debemos indicar nuestros servidores M1 y M2, así como el tipo de peticiones a balancear, en el archivo `/etc/haproxy/haproxy.cfg`:

```
        # 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-sslv3

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 192.168.56.103:80 maxconn 32
        server m2 192.168.56.101:80 maxconn 32
"/etc/haproxy/haproxy.cfg" 45L, 1441C escritos                          45,39-46    Final
```

Lanzamos haproxy con `sudo systemctl restart haproxy` y comprobamos el correcto balanceo de la carga:



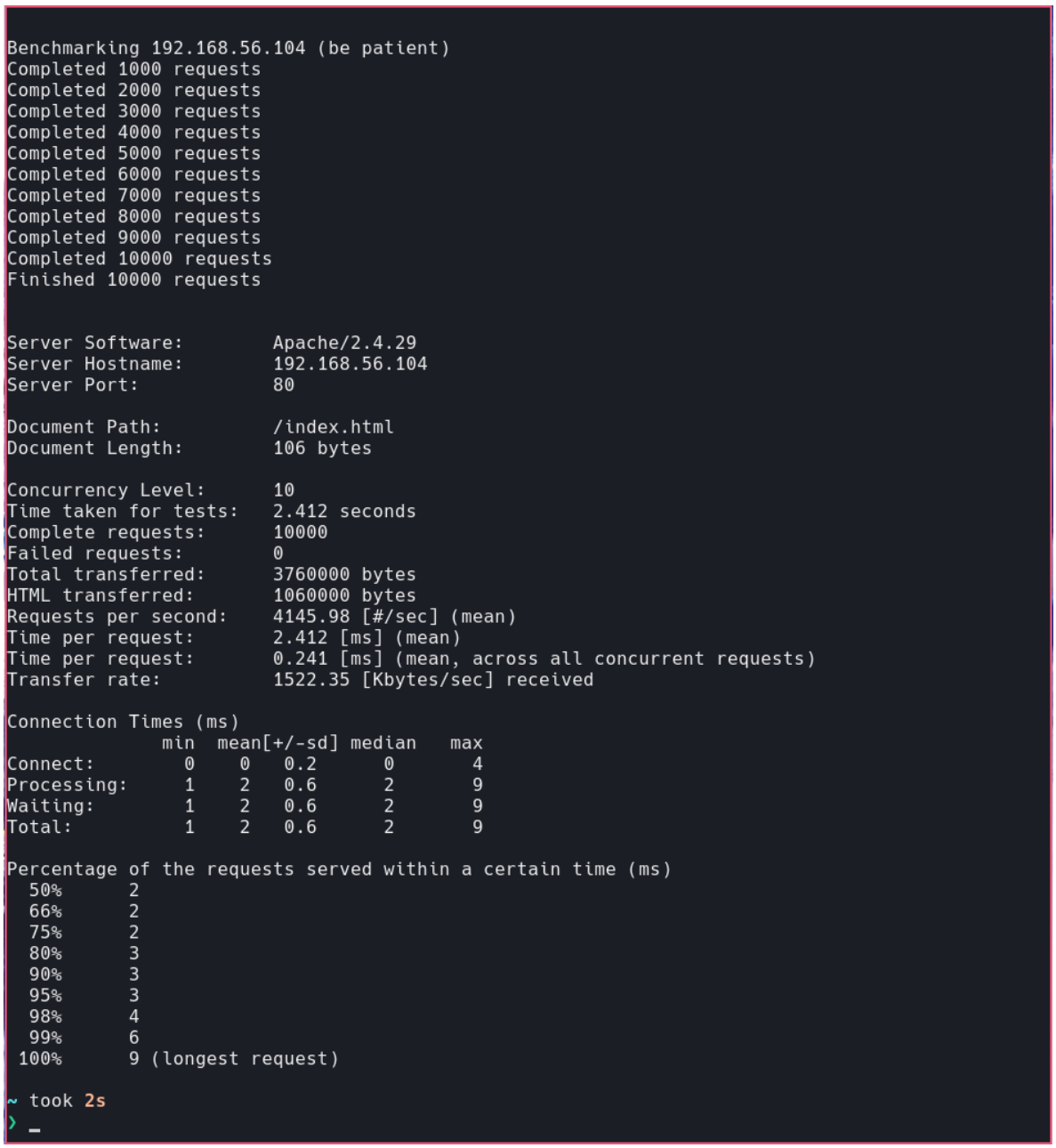
Someter a la granja web a una alta carga, generada con la herramienta Apache Benchmark, teniendo primero nginx y después haproxy

Para probar la resistencia a la carga, ejecutamos el comando `ab -n 10000 -c 10 http://192.168.56.104/index.html`, tanto con nginx como con haproxy. Haremos la petición esta vez desde el host, por variar.

Prueba con nginx:

```
Benchmarking 192.168.56.104 (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:        192.168.56.104
Server Port:            80

Document Path:          /index.html
Document Length:        106 bytes

Concurrency Level:      10
Time taken for tests:   2.170 seconds
Complete requests:      10000
Failed requests:        0
Total transferred:      3750000 bytes
HTML transferred:       1060000 bytes
Requests per second:    4609.30 [#/sec] (mean)
Time per request:       2.170 [ms] (mean)
Time per request:       0.217 [ms] (mean, across all concurrent requests)
Transfer rate:          1687.98 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.1      0       4
Processing:     1    2   0.6      2       9
Waiting:        1    2   0.6      2       9
Total:          1    2   0.6      2       9

Percentage of the requests served within a certain time (ms)
  50%      2
  66%      2
  75%      2
  80%      2
  90%      3
  95%      3
  98%      4
  99%      5
 100%      9 (longest request)

~ took 2s
❯
```

Prueba con haproxy:



Podemos ver que todas las peticiones se han completado con ambos balanceadores, y que los tests se han ejecutado ligeramente más rápido utilizando nginx.