Optimización del rendimiento de una stack LAMP

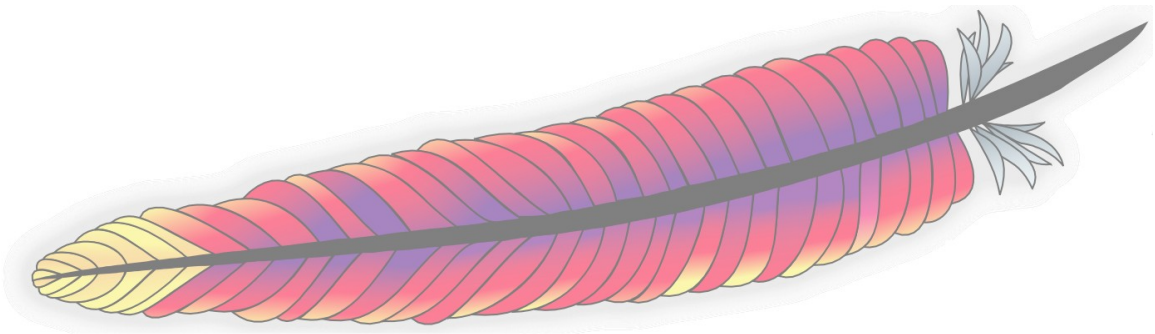


Table of Contents

Objetivos

- Ser capaz de instalar una stack LAMP completa y configurar un sitio web.
- Medir el rendimiento de nuestro servidor, ajustar la configuración para obtener el mayor rendimiento posible e identificar cuellos de botella.

Material necesario:

- Ubuntu 11.04 (salvo la instalación de paquetes, el contenido es independiente de la distribución).
- Una stack LAMP (# apt-get install lamp-server^).
- ab (Apache HTTP server benchmarking tool, incluida con Apache).

Conceptos básicos

¿Que es Apache?

Software para servir páginas web. Está disponible para una gran variedad de plataformas y es de código abierto.

¿Qué es ab?

Es un programa creado por la fundación Apache que sirve para medir el número de peticiones por segundo que puede servir el servidor Apache.

¿Optimización?

Sí. Debido a la gran variedad de hardware que existe y a los diferentes tipos de contenido que pueden servirse (un blog pequeño, un portal tipo Terra, una aplicación web escrita en PHP...), ajustar la configuración de Apache para extraer el mayor rendimiento posible de nuestra máquina es algo necesario.

¿Cómo se mide el rendimiento de un servidor?

Existen muchas formas de hacerlo (test de carga, de bases de datos, de ancho de banda, conexiones simultáneas...). Nosotros vamos a utilizar "ab", que nos permite simular cientos de conexiones simultáneas a un servidor. Veamos un ejemplo:

$ ab -n500 -c10 -k http://184.72.155.144/test.html
-n Es el número de conexiones totales que van a realizarse.
-c Es el número de clientes que va a haber conectados simultáneamente.
-k Realiza múltiples peticiones dentro de una sesión HTTP (KeepAlive).
http://184.72.155.144/test.html URL sobre la cual se realizará la petición.

Y esta es la salida correspodiente a un test de estrés con la configuración por defecto usando mpm-prefork:

```
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 184.72.155.144 (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Finished 500 requests


Server Software:        Apache/2.2.14
Server Hostname:        184.72.155.144
Server Port:            80

Document Path:          /test.html
Document Length:        170 bytes

Concurrency Level:      10
Time taken for tests:   10.825 seconds # Tiempo total que ha durado el test
Complete requests:      500
Failed requests:        0
Write errors:           0
Keep-Alive requests:    500
Total transferred:      241010 bytes
```

```
HTML transferred:       85000 bytes
Requests per second:    46.19 [#/sec] (mean)        # Peticiones por segundo, de media.
Time per request:       216.495 [ms] (mean)         # Tiempo que ha tardado de media en servirse cada
                                                    # peticion, en milisegundos
Time per request:       21.649 [ms] (mean, across all concurrent requests)
Transfer rate:          21.74 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    4   25.7      0     208
Processing:   164  211  145.0    201    2876
Waiting:      164  211  145.0    200    2876
Total:        164  215  160.3    201    3084

Percentage of the requests served within a certain time (ms)
 50%    201
 66%    202
 75%    202
 80%    202
 90%    204
 95%    207
 98%    380
 99%    401
100%   3084 (longest request)
```

216 milisegundos por petición suena muy bien. Veamos el resultado si usamos mpm-worker:

```
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 127.0.0.1 (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Finished 500 requests


Server Software:        Apache/2.2.17
Server Hostname:        127.0.0.1
```

```
Server Port:            80

Document Path:          /index.html
Document Length:        170 bytes

Concurrency Level:      10
Time taken for tests:   0.206 seconds
Complete requests:      500
Failed requests:        0
Write errors:           0
Keep-Alive requests:    500
Total transferred:      199510 bytes
HTML transferred:       43000 bytes
Requests per second:    2427.36 [#/sec] (mean)
Time per request:       4.120 [ms] (mean)
Time per request:       0.412 [ms] (mean, across all concurrent requests)
Transfer rate:          945.86 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.3      0       3
Processing:     0    4   3.6      4      24
Waiting:        0    4   3.6      4      23
Total:          0    4   3.6      4      24

Percentage of the requests served within a certain time (ms)
  50%      4
  66%      5
  75%      5
  80%      6
  90%      7
  95%      9
  98%     18
  99%     21
 100%     24 (longest request)
```

Ten en cuenta que esto es un archivo HTML estático, con archivos más complejos o dinámicos (PHP, Ruby...) este tiempo aumenta considerablemente. Veamos como podemos optimizar la configuración para servir este archivo lo más rápido posible.

Optimización de Apache.

¿Debo usar mpm-prefork o mpm-worker?

Dado que nuestro objetivo es tener una stack LAMP completa y la documentación de PHP no recomienda el uso de mpm-worker por posibles problemas de estabilidad y seguridad (http://www.php.net/manual/en/install.unix.apache2.php), es preferible el uso de mpm-prefork. De ahora en adelante, salvo que se indique lo contrario específicamente, se asume que Apache está configurado para usar mpm-prefork.

Ajustar la configuración de Apache

Editando el archivo apache2.conf (o httpd.conf, según nuestra distribución), hemos de añadir al final lo siguiente:

- MaxClients: Número máximo de clientes simultáneos. Para obtener el valor óptimo de esta directiva, dividimos el total de RAM de nuestro sistema entre el tamaño medio de los procesos apache2 (o httpd, según nuestra distribución). En mi caso con 612 MB de RAM, dejando para el sistema operativo 200 me sale 80.
- HostnameLookups Off: Cuando Apache recibe una petición, resuelve la dirección IP y almacena el nombre de dominio en el log. Pero debido al diseño de Apache, el cliente tendrá que esperar hasta que la resolución se complete para que se sirva su petición, así que lo desactivamos. Si queremos resolver los nombres de dominio de nuestros archivos log, podemos utilizar la utilidad "logresolve" más tarde.
- StartServers: Especifica el número de procesos hijo que se crean al iniciar Apache. Dicho número está controlado dinámicamente según la carga del servidor, así que habitualmente no hay motivo para modificarlo. En este caso, el valor por defecto es 5.
- MinSpareServers: Número de procesos hijo en espera, listos para atender una petición. Si hay menos procesos hijo en espera que el valor de esta directiva, se crean a un ritmo máximo de uno por segundo. Para estos tests usaré 5.
- MaxSpareServers: El número máximo de procesos hijo en espera. Para estos tests, 32.
- MaxRequestsPerChild: El número máximo de peticiones que un proceso hijo puede atender durante su vida. Usaremos 10 en este documento.

Módulos de Apache.

Sirven para expandir la funcionalidad de Apache. Se puede añadir una gran variedad de nuevas funciones, como soporte para PHP con mod_php, autenticación (mod_auth, mod_auth_ldap...) o reescritura de URLs según expresiones regulares (mod_rewrite).
Apache viene con muchos de ellos activados, pero sólo unos pocos son necesarios para empezar a

servir páginas. Para activar y desactivar un módulo se puede comentar la línea que lo define en apache2.conf (LoadModule mod_alias, por ejemplo) o se puede utilizar la utilidad “a2dismod”.

En mi caso, dejaré activados sólo los siguientes:

- alias: Permite definir alias para distintos directorios mediante la directiva Directory.
- authz_host: Permite restringir el acceso a directorios según de que host provenga la petición.
- Dir: Añade / al final de la petición si es necesario.
  Por ejemplo: http://servidor.com/wordpress → http://servidor.com/wordpress/

- env: Permite controlar variables internas que son usadas por varios componentes de Apache.
  Ejemplo: PassEnv LD_LIBRARY_PATH=/usr/lib/
- mime: Determina el tipo de archivo según la extensión del mismo, y envía esta información al navegador.
- Php5: Habilita el procesamiento de archivos .php
- reqtimeout: Especifica timeouts y ratios de transferencia mínimos.
- Setenvif: Ajusta variables de entorno según el User-Agent.
- Headers: Permite controlar las cabeceras HTTP, y reemplazarlas por otras.

Tras quedarnos sólo con estos módulos, veamos si se nota la diferencia:

$ ab -n500 -c10 -k http://184.72.155.144/test.html
```
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 184.72.155.144 (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Finished 500 requests


Server Software:        Apache/2.2.14
Server Hostname:        184.72.155.144
Server Port:            80

Document Path:          /test.html
```

```
Document Length:        170 bytes

Concurrency Level:      10
Time taken for tests:   10.328 seconds
Complete requests:      500
Failed requests:        0
Write errors:           0
Keep-Alive requests:    500
Total transferred:      232930 bytes
HTML transferred:       85000 bytes
Requests per second:    48.41 [#/sec] (mean)
Time per request:       206.563 [ms] (mean)
Time per request:       20.656 [ms] (mean, across all concurrent requests)
Transfer rate:          22.02 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    4  25.5      0     210
Processing:   162  201 163.7    186    2441
Waiting:      162  201 163.7    186    2441
Total:        162  205 182.1    186    2650

Percentage of the requests served within a certain time (ms)
  50%    186
  66%    195
  75%    198
  80%    200
  90%    210
  95%    218
  98%    374
  99%    596
 100%   2650 (longest request)
```

Bueno, no está mal. Pero, ¿cómo rendirá nuestro servidor web sirviendo una aplicación PHP?

Midiendo el rendimiento con páginas dinámicas.

Con la configuración anterior, instalamos Wordpress (www.wordpress.org) y realizamos un test de estrés sobre la página principal.

$ ab -n100 -c5 -k http://184.72.155.144/wordpress/

```
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 184.72.155.144 (be patient).....done


Server Software:        Apache/2.2.14
Server Hostname:        184.72.155.144
Server Port:            80

Document Path:          /wordpress/
Document Length:        6437 bytes

Concurrency Level:      5
Time taken for tests:   119.459 seconds
Complete requests:      100
Failed requests:        0
Write errors:           0
Keep-Alive requests:    0
Total transferred:      669800 bytes
HTML transferred:       643700 bytes
Requests per second:    0.84 [#/sec] (mean)
Time per request:       5972.928 [ms] (mean)
Time per request:       1194.586 [ms] (mean, across all concurrent requests)
Transfer rate:          5.48 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:      160  252 471.2    172    4502
Processing:   483 5567 10703.6   1023   47294
Waiting:      318 5335 10534.4    846   46301
Total:        644 5819 10795.6   1194   47465

Percentage of the requests served within a certain time (ms)
 50%   1194
 66%   1273
 75%   1303
 80%   1353
 90%  29297
```

Time per request: **5972.928 [ms]** (mean)

```
 95%  30528
 98%  38553
 99%  47465
100%  47465 (longest request)
```

6 segundos de media en responder cada petición. Nótese que un 10 % de las peticiones han necesitado entre 30 y 47 segundos para completarse. Esto es inaceptable, y seguro que ese 10% no vuelve a visitarnos.

Probemos con mpm-worker y PHP a través de FastCGI:

```
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 127.0.0.1 (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Finished 500 requests


Server Software:        Apache/2.2.17
Server Hostname:        127.0.0.1
Server Port:            80

Document Path:          /wordpress/index.php
Document Length:        0 bytes

Concurrency Level:      10
Time taken for tests:   258.041 seconds
Complete requests:      500
Failed requests:        0
Write errors:           0
Non-2xx responses:      500
Keep-Alive requests:    500
Total transferred:      172010 bytes
HTML transferred:       0 bytes
Requests per second:    1.94 [#/sec] (mean)
Time per request:       5160.816 [ms] (mean)
```

```
Time per request:       516.082 [ms] (mean, across all concurrent requests)
Transfer rate:          0.65 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.2      0       2
Processing:  2420 5144 587.1   5194    6707
Waiting:     2420 5144 587.0   5194    6707
Total:       2420 5145 587.1   5194    6707

Percentage of the requests served within a certain time (ms)
  50%   5194
  66%   5422
  75%   5558
  80%   5627
  90%   5846
  95%   6054
  98%   6227
  99%   6346
 100%   6707 (longest request)
```

Hay que reducir el tiempo de proceso de PHP como sea. La solución más sencilla es arrojar dinero al problema y aumentar la capacidad de proceso de nuestro servidor, pero aún hay mucho que podemos hacer antes de llegar a ese punto.

Optimizando PHP

El problema de PHP

PHP es un lenguaje de programación interpretado. Un lenguaje interpretado es un lenguaje de programación que está diseñado para ser ejecutado por medio de un intérprete, en contraste con los lenguajes compilados. Es decir: en lugar de compilarse el programa una vez, PHP se compila sobre la marcha cada vez que va a ejecutarse. La gran ventaja es que nuestra aplicación PHP no necesita ser recompilada si cambiamos de sistema operativo, mientras que la cruz es que cada vez que vayamos a ejecutar un archivo .php tiene que ser compilado sobre la marcha.

Este problema puede solventarse mediante un framework que cachee los resultados de esta compilación. Existen varias alternativas en este espacio pero nosotros vamos a trabajar con Alternative PHP Cache (APC). No es la más rápida, pero sí la más estable y es candidata a ser integrada en el núcleo de PHP en un futuro. Algunas alternativas son eAccelerator y XCache.

Instalando y configurando APC

```
# apt-get install php-apc
```

Vamos a usar los ajustes por defecto de APC, que teóricamente proporcionan el mejor rendimiento en todas las situaciones. Si quieres tener mayor control sobre que se cachea y como, tienes más información disponible en el manual (http://www.php.net/manual/en/apc.configuration.php).

Editamos el archivo /etc/php5/conf.d/apc.ini y añadimos lo siguiente:

```
#Activar o desactivar APC
apc.enabled="1"
```

Testeamos:
$ ab -n100 -c5 -k http://184.72.155.144/wordpress/

```
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 184.72.155.144 (be patient).....done


Server Software:        Apache/2.2.14
Server Hostname:        184.72.155.144
Server Port:            80

Document Path:          /wordpress/
Document Length:        6437 bytes

Concurrency Level:      5
Time taken for tests:   12.420 seconds
Complete requests:      100
Failed requests:        0
Write errors:           0
Keep-Alive requests:    0
Total transferred:      669800 bytes
HTML transferred:       643700 bytes
Requests per second:    8.05 [#/sec] (mean)
Time per request:       621.007 [ms] (mean)
Time per request:       124.201 [ms] (mean, across all concurrent requests)
```

```
Transfer rate:          52.66 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:      159  173  11.7    171     206
Processing:   364  434  58.6    422     638
Waiting:      203  259  55.9    244     442
Total:        523  607  63.6    599     836

Percentage of the requests served within a certain time (ms)
 50%    599
 66%    625
 75%    645
 80%    653
 90%    681
 95%    736
 98%    791
 99%    836
100%    836 (longest request)
```

Y con mpm-worker:

```
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 127.0.0.1 (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Finished 500 requests


Server Software:        Apache/2.2.17
Server Hostname:        127.0.0.1
Server Port:            80

Document Path:          /wordpress/index.php
Document Length:        0 bytes
```

```
Concurrency Level:      10
Time taken for tests:   279.398 seconds
Complete requests:      500
Failed requests:        0
Write errors:           0
Non-2xx responses:      500
Keep-Alive requests:    500
Total transferred:      172010 bytes
HTML transferred:       0 bytes
Requests per second:    1.79 [#/sec] (mean)
Time per request:       5587.951 [ms] (mean)
Time per request:       558.795 [ms] (mean, across all concurrent requests)
Transfer rate:          0.60 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   1.2      0      10
Processing:  2295 5567 512.0   5586    7007
Waiting:     2295 5567 512.0   5586    7007
Total:       2295 5568 512.1   5586    7007

Percentage of the requests served within a certain time (ms)
  50%   5586
  66%   5783
  75%   5898
  80%   5966
  90%   6133
  95%   6364
  98%   6582
  99%   6664
 100%   7007 (longest request)
```

Ahora que ya tenemos solucionado el problema de los archivos .php, ¿qué más podemos optimizar? Por ejemplo la carga de imágenes, archivos de hoja de estilo...

Squid como reverse-proxy.

Squid es un servidor proxy y un demonio para caché de páginas web. ¿Cómo puede ayudarnos Squid?

INTERNET → *Squid* (puerto 80) → *Apache* (puerto 8080)

Squid cacheará las peticiones a todos los archivos, de manera que si están en memoria se servirán de ahí en lugar de realizar una petición a Apache.

```
# apt-get install squid
```

Nota importante: En el caso de Wordpress, nos interesa cachear la portada (al fin y al cabo, lo que va a ver más gente). Nos interesa que los comentarios se sirvan desde la copia del servidor, para evitar que los usuarios no vean sus comentarios mientras haya una versión antigua de la página en la caché. Si Wordpress tuviese soporte para Squid (como Media-wiki, por ejemplo), cada vez que cambiase el contenido Wordpress avisaría a Squid para que refresque la caché.

/etc/squid/squid.conf

```
cache_mgr root

#Servidor para el cual haremos de proxy
http_port 80 vhost

# IP y puerto del mismo
cache_peer 127.0.0.1 parent 8080 0 no-query originserver login=PASS

acl apache rep_header Server ^Apache
broken_vary_encoding allow apache

# Lugar donde se almacenará la caché y tamaño
cache_dir ufs /var/spool/squid 10000 16 256
cache_mem 64 MB
maximum_object_size_in_memory 128 KB

# Logs
#logformat common %>a %ui %un [%tl] "%rm %ru HTTP/%rv" %Hs %<st %Ss:%Sh
logformat combined %>a %ui %un [%tl] "%rm %ru HTTP/%rv" %Hs %<st "%{Referer}>h" "%
{User-Agent}>h" %Ss:%Sh

access_log /var/log/squid/access.log combined

cache_log /var/log/squid/cache.log
cache_store_log /var/log/squid/store.log
logfile_rotate 10

hosts_file /etc/hosts
```

```
# Control de acceso. Sólo será permitido el acceso desde el puerto 80
acl all src 0.0.0.0/0.0.0.0
acl manager proto cache_object
acl localhost src 127.0.0.1/255.255.255.255
acl to_localhost dst 127.0.0.0/8
acl Safe_ports port 80
acl purge method PURGE
acl CONNECT method CONNECT

http_access allow manager localhost
http_access deny manager
http_access allow purge localhost
http_access deny purge
http_access deny !Safe_ports
http_access allow localhost
http_access allow all
http_access allow all
http_reply_access allow all

icp_access allow all

cache_effective_group proxy

coredump_dir /var/spool/squid

forwarded_for on

emulate_httpd_log on

redirect_rewrites_host_header off

buffered_logs on

# No cachear contenido dinámico como cgi-bin, direcciones ?, POSTS
hierarchy_stoplist cgi-bin ?
acl QUERY urlpath_regex cgi-bin \?
acl POST method POST
no_cache deny QUERY
no_cache deny POST
```

```
# No cachear el panel de administración de Wordpress, ni phpMyAdmin
acl adminurl urlpath_regex ^/wp-config
no_cache deny adminurl
acl phpmyadminurl urlpath_regex ^/phpmyadmin
no_cache deny phpmyadminurl
```

Como ahora es Squid quién recibe las peticiones de los clientes y no Apache, si queremos obtener estadísticas de nuestro servidor web deberemos analizar los logs de Squid en su lugar.

Editamos el archivo ports.conf y cambiamos la IP y el puerto de Apache de manera que sólo sea accesible a través de Squid.

```
Listen 127.0.0.1:8080
```

Hacemos lo mismo con ./sites-enabled/default

```
<VirtualHost 127.0.0.1:8080>
```

El comportamiento por defecto de Wordpress es servir siempre el contenido desde el disco, sin mirar en la caché. Para evitar esto, vamos a modificar las cabeceras de las peticiones que emite Wordpress mediante un archivo .htaccess de manera que esa decisión corresponda a Squid. En el mismo archivo, fijamos AllowOverride a All.

Creamos un archivo .htaccess en el directorio /wordpress/ con el siguiente contenido

```
Header unset Pragma
Header set Cache-Control "must-revalidate, max-age=0, s-maxage=600"
Header set Vary "Accept-Encoding"
```

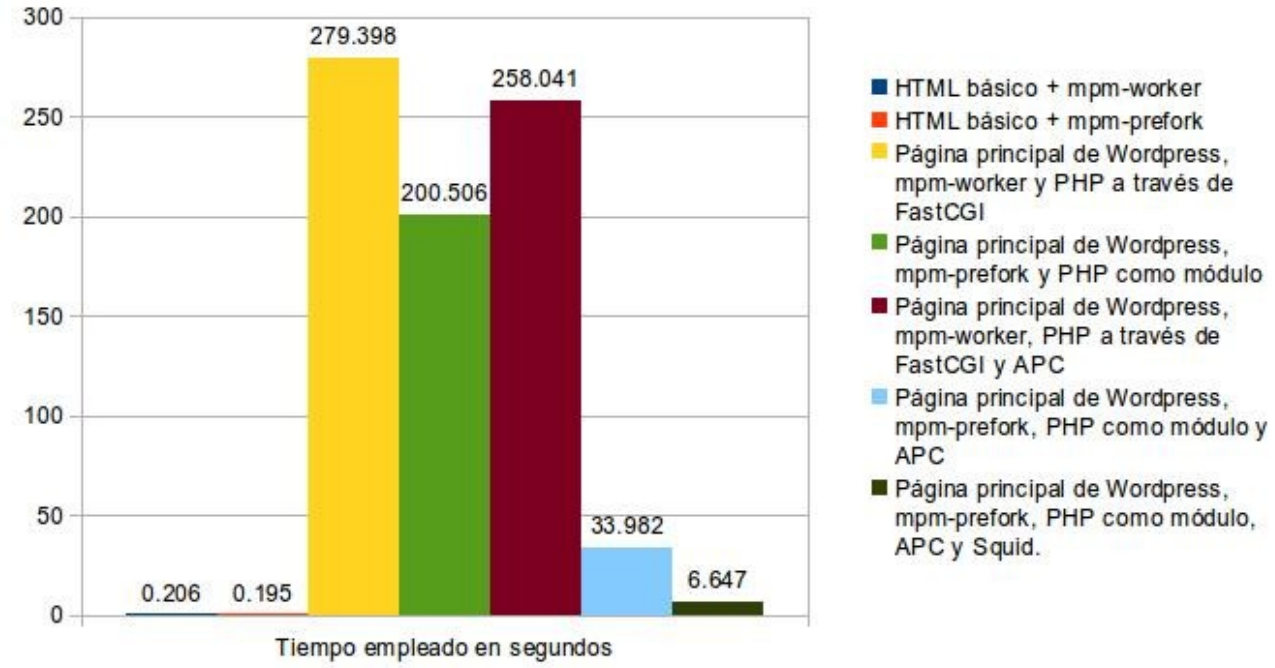
Y ahora, comprobemos que todo funciona:

$ab -n100 -c5 -k http://184.72.155.144/wordpress/

```
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 184.72.155.144 (be patient).....done


Server Software:        Apache/2.2.14
Server Hostname:        184.72.155.144
Server Port:            80

Document Path:          /wordpress/
Document Length:        6437 bytes

Concurrency Level:      5
Time taken for tests:   6.647 seconds
Complete requests:      100
Failed requests:        0
Write errors:           0
Keep-Alive requests:    100
Total transferred:      714800 bytes
HTML transferred:       643700 bytes
Requests per second:    15.04 [#/sec] (mean)
Time per request:       332.372 [ms] (mean)
Time per request:       66.474 [ms] (mean, across all concurrent requests)
Transfer rate:          105.01 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0   10  45.8      0     215
Processing:   173  318 200.0    240    1322
Waiting:      165  269 122.2    210    1042
Total:        173  329 209.9    240    1322

Percentage of the requests served within a certain time (ms)
 50%    240
```

```
66%    354
75%    403
80%    408
90%    609
95%    770
98%   1223
99%   1322
100%   1322 (longest request)
```

Los resultados se ven de forma mucho más clara en el siguiente gráfico:



Optimización de MySQL

Midiendo el rendimiento de la base de datos: SQLBench

Para realizar las pruebas de rendimiento vamos a utilizar un software llamado SQLBench. SQL Bench es una suite de benchmarks diseñada para conocer el rendimiento de nuestra instalación de MySQL. Puede descargarse desde la web de MySQL (http://dev.mysql.com/doc/refman/5.0/en/mysql-benchmarks.html).

Pre-requisitos de SQLBench

Una servidor de base de datos MySQL.
Un intérprete de código Perl, y los módulos relacionados para trabajar con bases de datos MySQL (http://dev.mysql.com/doc/refman/5.0/en/perl-installation.html).

Para instalarlos debemos instalar los paquetes correspondientes. Las instrucciones exactas varían de una distribución a otra. En Debian y derivados ejecutamos lo siguiente desde la terminal:

```
# apt-get install libmysqlclient16-dev
# perl -MCPAN -e shell
cpan> install DBI
cpan> install DBD::mysql
```

Preparemos la base de datos que usará SQLBench:

```
mysql> create database test;
```

A continuación vamos a medir el rendimiento inicial de nuestra base de datos tal y como viene de los repositorios.

Benchmark inicial, sin modificar la configuración.

```
$ ./test-alter-table.sh -user=root -password=123456
Testing of ALTER TABLE
Testing with 1000 columns and 1000 rows in 100 steps
Insert data into the table

Time for insert (1000) 0 wallclock secs ( 0.02 usr  0.01 sys +  0.00 cusr  0.00 csys =  0.03 CPU)
Time for alter_table_add (100): 15 wallclock secs ( 0.02 usr  0.00 sys +  0.00 cusr  0.00 csys =  0.02
CPU)
Time for create_index (8):  2 wallclock secs ( 0.00 usr  0.00 sys +  0.00 cusr  0.00 csys =  0.00 CPU)
Time for drop_index (8):  2 wallclock secs ( 0.00 usr  0.00 sys +  0.00 cusr  0.00 csys =  0.00 CPU)
Time for alter_table_drop (91): 19 wallclock secs ( 0.02 usr  0.01 sys +  0.00 cusr  0.00 csys =  0.03
CPU)

Total time: 38 wallclock secs ( 0.06 usr  0.02 sys +  0.00 cusr  0.00 csys =  0.08 CPU)
```

¿Cómo ver la configuración actual?

Abrimos la consola de mysql con el siquiente comando:

```
$ mysql -uroot -pPASSWORD
```

Una vez dentro, mostramos la configuración en pantalla:

```
mysql> show variables;
```

A continuación vamos a explicar las opciones más importantes junto con la versión de MySQL a partir de la cual están disponibles. La referencia completa se encuentra disponible en la web de MySQL (http://dev.mysql.com/doc/refman/5.5/en/server-system-variables.html).

Para establecer estas opciones hay que añadirlas en el archivo de configuración */etc/my.cnf*, añadirlas a la línea de comando usada para arrancar el servidor MySQL o modificarlas en tiempo de ejecución mediante el comando *set*.

log_slow_queries (<5.6) o **slow_query log**(>=5.6)): Controla si se registran las consultas que tardan más de un determinado tiempo. Por defecto, se almacena toda consulta que se ejecute durante más de 10 segundos y estos valores se leen usando *mysqldumpslow*.

Valor por defecto: Off.
Valor recomendado: On.

long_query_time: Controla el tiempo mínimo durante el cual tiene que ejecutarse una consulta para que pueda ser registrada en el slow_query_log.

Valor por defecto en segundos: 10
Valor recomendado en segundos: 5

log_queries_not_using_indexes (>=4.1): Registra las consultas que no usan los índices, aunque tarden menos del tiempo especificado en long_query_time.

Valor por defecto: Off
Valor recomendado: On

query_cache_size: El tamaño de la cache de consultas. ¡Cuidado! Aunque por norma general las caches se consideran algo positivo, debido a que MySQL tiene que mirar la cache en cada consulta y emplear ciclos de proceso en mantenerla hay que asegurarse de que beneficia positivamente a nuestro caso concreto. Se recomienda experimentar realizando incrementos de 10 megabytes hasta que el rendimiento se vea afectado de forma negativa.

Valor por defecto: 0 (desactiva la cache).

max_connections: Similar a MaxClients en Apache, controla el número máximo de conexiones simultáneas que serán admitidas.

Valor por defecto: 151

wait_timeout: El tiempo límite que una conexión puede estar inactiva. Si tienes muchos usuarios interactivos o usas conexiones persistentes a la base de datos, no se recomienda especificar un valor bajo.

Valor por defecto en segundos: 28800

default-storage-engine: El formato por defecto para las nuevas tablas. Las opciones disponibles son MyISAM e InnoDB, deberás elegir uno u otro según tus necesidades. Esto se discute con detalle en la próxima sección.

binlog-format: El formato del log que se usa para la replicación. Los valores disponibles son:

- **ROW**: Se almacenan los cambios realizados tomando como unidad las filas de la tabla.
- **STATEMENT**: Se almacen los cambios realizados tomando como unidad las consultas SQL. Este modo puede ocasionar problemas de consistencia en los casos descritos en la documentación de MySQL (http://dev.mysql.com/doc/refman/5.5/en/binary-log-mixed.html).
- **MIXED**: Se usa el modo STATEMENT por defecto, excepto en los casos en los que es inadecuado, para los cuales se usa el modo ROW.

Si usas InnoDB como formato de tabla deberías usar el valor MIXED para binlog-format, ya que el valor por defecto (STATEMENT) no es seguro para ciertas consultas y puede generar inconsistencias en la base de datos. Cuando se intenta ejecutar una consulta conflictiva esta podría fallar y devolver un mensaje de error. Algunas aplicaciones no detectan esta condición y fallan (Moodle 2.1 en puntos aleatorios de la instalación, por ejemplo).

Una explicación técnica detallada está disponible en la documentación de MySQL (http://dev.mysql.com/doc/refman/5.5/en/replication-rbr-safe-unsafe.html).

¿Qué tipo de tablas debo usar, InnoDB o MyISAM?

Ventajas de MyISAM frente a InnoDB:

- Mejor rendimiento con hardware más modesto.

- La mayoría de aplicaciones (Wordpress, Drupal...) están escritas con MyISAM como objetivo y no aprovechan las ventajas de InnoDB.

Ventajas de InnoDB frente a MyISAM:

- Es *ACID compliant* (**A**tomicity, **C**onsistency, **I**solation and **D**urability: Atomicidad, Consistencia, Aislamiento y Durabilidad). Si nuestro sevidor sufre un cuelgue, la base de datos estará en un estado consistente y no tendremos que preocuparnos por recuperar consultas ejecutadas a medias o tablas corruptas.
- Bloqueo de filas o de tablas según se necesita, frente a MyISAM que solo soporta bloqueos de tabla.
- Como consecuencia de lo anterior, realizar una copia de seguridad de una tabla MyISAM conlleva que el servidor deje de funcionar durante el tiempo que se necesite para realizar la copia. Con InnoDB, no.
- Otra consecuencia es que podemos realizar varias escrituras simultáneas en la misma tabla si son en diferentes filas, permitiéndonos una mayor concurrencia que MyISAM.

En conclusión: InnoDB es la elección adecuada en la mayoría de los casos, salvo que tengamos alguna aplicación que necesite de funcionalidad que sólo esté disponible en MyISAM (por ejemplo, que haga uso de índices de texto completo) o tengamos pocos recursos de hardware disponibles.

¿Cómo cambiar el motor por defecto de MyISAM a InnoDB?

Advertencia: Haz una copia de seguridad de tu base de datos antes de realizar esta operación.

Puedes convertir tus tablas a InnoDB de la siguiente manera:

- Cambiando el motor por defecto en my.cnf para que todas las tablas creadas a partir de ese momento por defecto usen InnoDB.

```
default-storage-engine=innodb
```

- Para alterar el tipo de tablas ya existentes, usa el comando ALTER TABLE:

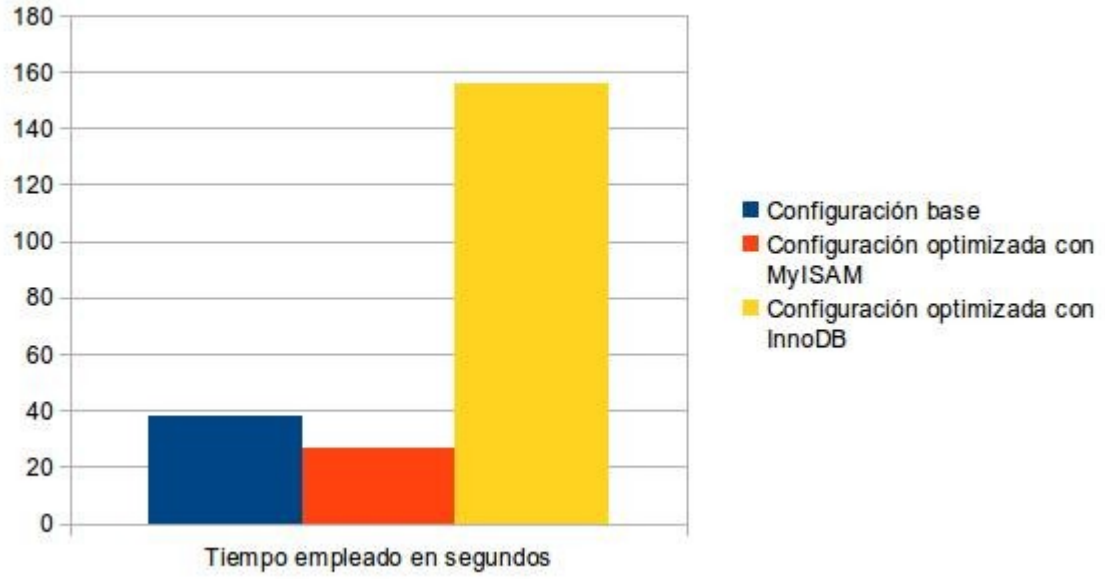
```
ALTER TABLE nombre_tabla ENGINE=InnoDB;
```

Resultados de benchmark con MyISAM, tras ajustar la configuración según la máquina que tenemos:

```
$ ./test-alter-table.sh -user=root -password=123456
Testing of ALTER TABLE
Testing with 1000 columns and 1000 rows in 100 steps
Insert data into the table

Time for insert (1000) 0 wallclock secs ( 0.02 usr  0.01 sys +  0.00 cusr  0.00 csys =  0.03 CPU)
Time for alter_table_add (100): 11 wallclock secs ( 0.02 usr  0.00 sys +  0.00 cusr  0.00 csys =  0.02
CPU)
Time for create_index (8):  0 wallclock secs ( 0.00 usr  0.00 sys +  0.00 cusr  0.00 csys =  0.00 CPU)
Time for drop_index (8):  1 wallclock secs ( 0.00 usr  0.00 sys +  0.00 cusr  0.00 csys =  0.00 CPU)
Time for alter_table_drop (91): 13 wallclock secs ( 0.02 usr  0.01 sys +  0.00 cusr  0.00 csys =  0.03
CPU)

Total time: 27 wallclock secs ( 0.06 usr  0.02 sys +  0.00 cusr  0.00 csys =  0.08 CPU)
```

Benchmark usando InnoDB:

```
~/sql-bench ./test-alter-table.sh -user=root -password=123456
Testing of ALTER TABLE
Testing with 1000 columns and 1000 rows in 100 steps
Insert data into the table

Time for insert (1000)38 wallclock secs ( 0.07 usr  0.02 sys +  0.00 cusr  0.00 csys =  0.09 CPU)
Time for alter_table_add (100): 56 wallclock secs ( 0.02 usr  0.00 sys +  0.00 cusr  0.00 csys =  0.02
CPU)
Time for create_index (8):  7 wallclock secs ( 0.00 usr  0.00 sys +  0.00 cusr  0.00 csys =  0.00 CPU)
Time for drop_index (8):  5 wallclock secs ( 0.00 usr  0.01 sys +  0.00 cusr  0.00 csys =  0.01 CPU)
Time for alter_table_drop (91): 49 wallclock secs ( 0.00 usr  0.00 sys +  0.00 cusr  0.00 csys =  0.00
CPU)

Total time: 156 wallclock secs ( 0.09 usr  0.03 sys +  0.00 cusr  0.00 csys =  0.12 CPU)
```



Es evidente que una instancia micro de Amazon no es suficiente para obtener un rendimiento aceptable usando InnoDB, por lo que tenemos dos opciones: Volver a MyISAM y tener una política de backups proporcional a lo valiosos que sean los datos que almacenemos en la base de datos, o seguir con InnoDB y actualizar nuestro servidor a uno más potente.

Conclusiones

La optimización de un servidor web es un tema muy amplio, con muchas posibilidades de mejora según el tipo de contenido que estemos sirviendo y los recursos de hardware de los que dispongamos. Es importante conocer a fondo el funcionamiento de cada una de las capas de nuestra stack web si queremos poder exprimir al máximo nuestro hardware.