

Ingeniería de Servidores (2015-2016)
GRADO EN INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE GRANADA

Memoria Práctica 4

Marta Gómez Macías

18 de diciembre de 2015

ÍNDICE

1. Instale la aplicación <i>Phoronix</i> . ¿Qué comando permite listar los benchmarks disponibles?	3
2. De los parámetros que podemos pasar al comando ¿Qué significa <code>-c 5</code> ? ¿y <code>-n 100</code> ? Monitorice la ejecución de <code>ab</code> contra alguna máquina (cualquiera) ¿cuántos procesos o hebras crea <code>ab</code> en el cliente?	4
3. Ejecute <code>ab</code> contra las tres máquinas virtuales (desde el SO anfitrión a las máquinas virtuales de la red local) una a una (arrancadas por separado) y muestre y comente las estadísticas. ¿Cuál es la que proporciona mejores resultados? Fijese en el número de bytes transferidos, ¿es igual para cada máquina?	5
4. Instale <i>jmeter</i> y siga el tutorial para hacer un <i>Web Test Plan</i> realizando capturas de pantalla y comentándolas. En vez de usar la web de <i>jmeter</i> , haga el experimento usando alguna de sus máquinas virtuales (puede hacer una página sencilla, usar las páginas de <i>phpmyadmin</i> , instalar un CMS, etc)	6
4.1. Instalando <i>jmeter</i>	6
4.2. Construyendo nuestro propio <i>Web Test Plan</i>	9
5. Programe un <i>benchmark</i> usando el lenguaje que desee. El <i>benchmark</i> debe incluir: 1) Objetivo del <i>benchmark</i> , 2) Métricas (unidades, variables, puntuaciones, etc), 3) Instrucciones para su uso y 4) Ejemplo analizando los resultados.	12
5.1. Objetivo del <i>benchmark</i>	12
5.2. Métricas	12
5.3. Instrucciones de uso	13
5.4. Ejemplo de ejecución	13
5.5. Detalles sobre el Script	13
6. Cuestiones opcionales	17
6.1. Seleccione, instale y ejecute un benchmark, comente los resultados. Atención: no es lo mismo un benchmark que una suite, instale un benchmark	17
6.3. Lea el artículo de comparación entre <i>Jmeter</i> y <i>Gatling</i> elaborado por la empresa <i>Flood.io</i> y elabore un breve resumen	17

ÍNDICE DE FIGURAS

1.1. Algunas de las suites disponibles para <i>Phoronix</i>	3
1.2. Algunos de los benchmarks disponibles para <i>Phoronix</i>	4
2.1. Output obtenido tras ejecutar <code>ab</code>	5
3.1. Salida obtenida tras ejecutar <code>ab</code> contra el servidor con Ubuntu Server	6
3.2. Salida obtenida tras ejecutar <code>ab</code> contra el servidor con CentOS	7
3.3. Salida obtenida tras ejecutar <code>ab</code> contra el servidor con Windows Server	8
4.1. Ventana inicial de <i>jmeter</i>	8
	1

4.2. Ruta para crear un <i>Grupo de hilos</i>	9
4.3. Propiedades para el grupo de hilos que usaremos como ejemplo	9
4.4. Ruta a seguir para especificar los valores por defecto de una petición HTTP para un grupo de hilos concreto	9
4.5. Propiedades para las peticiones HTTP a simular	10
4.6. Añadiendo soporte para cookies a nuestro <i>test plan</i>	10
4.7. Ruta a seguir para añadir peticiones HTTP a nuestro <i>test plan</i>	10
4.8. Valores que debe tener nuestra petición HTTP	11
4.9. Ruta a seguir para añadir un <i>Receptor</i> a nuestro <i>test plan</i>	11
4.10. Ventana de configuración del Receptor de nuestro <i>test plan</i>	11
4.11. Ejecutando nuestro <i>test plan</i>	12
5.1. Resultados tras ejecutar el benchmark	13
6.1. Salida del benchmark x264 mientras realiza sus pruebas	18

1. INSTALE LA APLICACIÓN *Phoronix*. ¿QUÉ COMANDO PERMITE LISTAR LOS BENCHMARKS DISPONIBLES?

En [8] encontramos una pequeña guía tanto de instalación como de uso de *Phoronix*. Para instalar el benchmark en Ubuntu sólo nos basta con usar el siguiente comando:

```
Instalando phoronix-test-suite en Ubuntu  
sudo apt-get install phoronix-test-suite
```

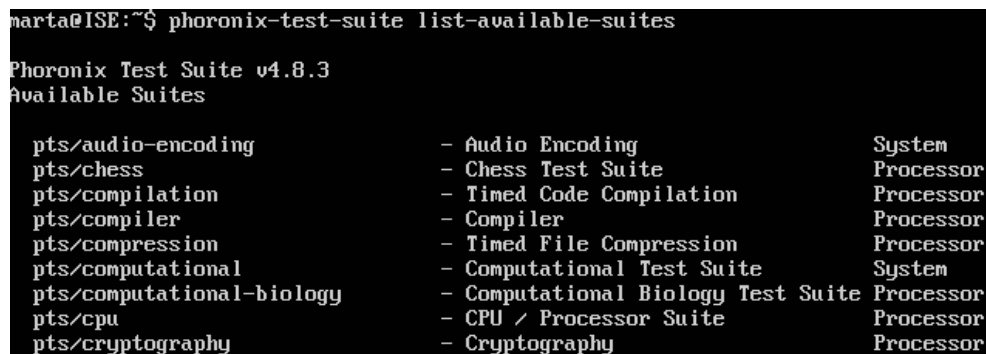
En *Phoronix* tenemos por un lado las *suites* y por otro lado los *test*. Las suites son grupos de *test* y los *test* son pruebas individuales, en el enunciado de la práctica se mencionan como *benchmarks*.

Para ver las *suites* disponibles usamos el comando:

```
Listando las suites disponibles para Phoronix  
phoronix-test-suite list-available-suites
```

Al ejecutar el comando por primera vez nos preguntará varias cosas tales como si aceptamos la licencia o permitimos que se envíen datos a los servidores de *Phoronix*, tras aceptar, empezarán a aparecer en pantalla distintas *suites* poco a poco.

En la [Figura 1.1](#) vemos un ejemplo de output de este comando (sin las preguntas que nos muestra por primera vez) donde vemos algunas *suites* interesantes tales como una *suite* de *gaming*, otra de una base de datos, etc.



```
marta@ISE:~$ phoronix-test-suite list-available-suites  
Phoronix Test Suite v4.8.3  
Available Suites  
  
pts/audio-encoding      - Audio Encoding      System  
pts/chess               - Chess Test Suite    Processor  
pts/compilation         - Timed Code Compilation Processor  
pts/compiler            - Compiler            Processor  
pts/compression         - Timed File Compression Processor  
pts/computational       - Computational Test Suite System  
pts/computational-biology - Computational Biology Test Suite Processor  
pts/cpu                 - CPU / Processor Suite Processor  
pts/cryptography        - Cryptography        Processor
```

Figura 1.1: Algunas de las suites disponibles para *Phoronix*

Para listar los *test* disponibles usamos el comando:

```
Listando los test disponibles para Phoronix  
phoronix-test-suite list-available-tests
```

Donde obtendremos una salida bastante parecida a la obtenida con el comando para listar las *suites* disponibles ([Figura 1.2](#)). Con la diferencia de que la salida será mucho más rápida.

pts/systester	- Systester	Processor
pts/tachyon	- Tachyon	Processor
pts/tesseract	- Tesseract	Graphics
pts/tf2	- Team Fortress 2	Graphics
pts/tiobench	- Threaded I/O Tester	Disk
pts/tremulous	- Tremulous	Graphics
pts/trislam	- Triangle Slammer	Graphics
pts/tscp	- TSCP	Processor
pts/ttsiod-renderer	- TTSIOD 3D Renderer	Processor
pts/unigine-heaven	- Unigine Heaven	Graphics
pts/unigine-sanctuary	- Unigine Sanctuary	Graphics
pts/unigine-tropics	- Unigine Tropics	Graphics
pts/unigine-valley	- Unigine Valley	Graphics

Figura 1.2: Algunos de los benchmarks disponibles para *Phoronix*

2. DE LOS PARÁMETROS QUE QUE PODEMOS PASAR AL COMANDO ¿QUÉ SIGNIFICA -c 5? ¿Y -n 100? MONITORICE LA EJECUCIÓN DE ab CONTRA ALGUNA MÁQUINA (CUALQUIERA) ¿CUÁNTOS PROCESOS O HEBRAS CREA ab EN EL CLIENTE?

Según [4], la opción `-c` especifica el número de peticiones que se hacen al servidor a la vez concurrentemente. Por defecto su valor es 1. Si su valor es mayor a 1, `ab` creará en el cliente tantos procesos concurrentes como se establezca en dicho valor, por tanto, en nuestro ejemplo, `ab` creará 5 procesos concurrentes.

La opción `-n` especifica el número de peticiones que se hacen para la sesión de benchmarking, el valor por defecto es 1 y normalmente no suele dar resultados representativos.

Para ejecutar el benchmark en un cliente sobre un servidor Apache, ejecutamos el siguiente comando:

```
_____ Ejecutando Apache Benchmark sobre un servidor _____
ab -c 5 -n 100 http://10.0.2.10/
```

También podríamos hacerlo sobre nuestra propia máquina cambiando la dirección IP del servidor por `http://localhost/`.

La salida obtenida tras ejecutar `ab` se ve en la [Figura 2.1](#). En primer lugar nos muestra información sobre el servidor al que le ejecutamos el benchmark. Después, nos muestra información sobre el documento HTML con el que va a trabajar el benchmark para hacer las pruebas. A continuación, nos muestra datos sobre el benchmark tales como nivel de concurrencia, tiempo total de ejecución, peticiones completadas y fallidas, etc. Como se esperaba, el nivel de concurrencia ha sido 5 y el número de peticiones, 100. Por último, nos muestra resultados estadísticos a modo de resumen sobre las 100 peticiones que hemos hecho: el máximo tiempo empleado para servir el `index.html` han sido 20 ms y el mínimo, 3. De las 100 peticiones hechas, el 50% ha tardado menos de 6ms en ser respondidas.

```

marta@ISE:~$ ab -c 5 -n 100 http://10.0.2.10/
This is ApacheBench, Version 2.3 <$Revision: 1528965 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 10.0.2.10 (be patient).....done

Server Software:      Apache/2.4.6
Server Hostname:      10.0.2.10
Server Port:          80

Document Path:        /
Document Length:      3026 bytes

Concurrency Level:    5
Time taken for tests:  0.140 seconds
Complete requests:    100
Failed requests:       0
Total transferred:    325100 bytes
HTML transferred:     302600 bytes
Requests per second:  713.68 [#/sec] (mean)
Time per request:      7.006 [ms] (mean)
Time per request:      1.401 [ms] (mean, across all concurrent requests)
Transfer rate:         2265.79 [Kbytes/sec] received

```

(a) Primera parte del output obtenido tras ejecutar ab

```

Connection Times (ms)
      min   mean[+/-sd] median   max
Connect:    0      0   0.3      0      1
Processing:  2      6   3.5      5     19
Waiting:    1      6   3.5      5     19
Total:      3      7   3.4      6     20

Percentage of the requests served within a certain time (ms)
 50%    6
 66%    6
 75%    6
 80%    7
 90%   11
 95%   19
 98%   19
 99%   20
100%   20 (longest request)
marta@ISE:~$ █

```

(b) Segunda parte del output obtenido tras ejecutar ab

Figura 2.1: Output obtenido tras ejecutar ab

3. EJECUTE ab CONTRA LAS TRES MÁQUINAS VIRTUALES (DESDE EL SO ANFITRIÓN A LAS MÁQUINAS VIRTUALES DE LA RED LOCAL) UNA A UNA (ARRANCADAS POR SEPARADO) Y MUESTRE Y COMENTE LAS ESTADÍSTICAS. ¿CUÁL ES LA QUE PROPORCIONA MEJORES RESULTADOS? FÍJESE EN EL NÚMERO DE BYTES TRANSFERIDOS, ¿ES IGUAL PARA CADA MÁQUINA?

Es imposible que el número de bytes transferidos sea igual para cada máquina pues cada una tiene un fichero *index.html* diferente. Para que fuera el mismo tamaño en todas deberíamos eliminar la página de inicio por defecto y poner la misma en todos los servidores.

En el caso de Ubuntu, hemos obtenido los resultados que se ven en la [Figura 3.1](#). La longitud del documento en este caso ha sido de 11510 bytes y la respuesta más lenta se ha hecho en 68 ms.

En el caso de CentOS, hemos obtenido los resultados que se ven en la [Figura 3.2](#). La longitud del documento ha sido de 3026 bytes y la respuesta más lenta se ha hecho en 745 ms. Siendo el archivo a transferir de un tamaño inferior, la respuesta ha sido muchísimo más lenta que en el caso del servidor con Ubuntu Server.

Por último, en el caso de Windows Server, hemos obtenido los resultados que se ven en la [Figura 3.3](#). La longitud del documento ha sido de 689 bytes y la respuesta más lenta se ha realizado en más de un segundo. Teniendo que transferir un archivo que pesa menos de 1KB pienso que esta es la máquina más lenta de las tres.

Si tuviésemos que hacer un “ranking”, cuya puntuación se obtendría con la parte entera de la división del tamaño del documento entre el tiempo de respuesta máximo, Ubuntu quedaría en primer puesto con 169 puntos, le seguiría CentOS con 4 y por último estaría Windows Server con 0.

```

[marta@marta-PC Practica4]$ ab -c 5 -n 100 http://192.168.56.6/
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.56.6 (be patient).....done

Server Software:      Apache/2.4.7
Server Hostname:      192.168.56.6
Server Port:          80

Document Path:        /
Document Length:      11510 bytes

Concurrency Level:    5
Time taken for tests:  0.136 seconds
Complete requests:    100
Failed requests:       0
Total transferred:    1178300 bytes
HTML transferred:     1151000 bytes
Requests per second:  732.94 [#/sec] (mean)
Time per request:     6.822 [ms] (mean)
Time per request:     1.364 [ms] (mean, across all concurrent requests)
Transfer rate:        8433.81 [Kbytes/sec] received

Connection Times (ms)
              min      mean[+/-sd] median   max
Connect:        0        0   0.2      0       1
Processing:      0        7  12.9      2       67
Waiting:         0        2   1.0      2        6
Total:          0        7  13.0      2       68

Percentage of the requests served within a certain time (ms)
 50%    2
 66%    3
 75%    3
 80%    4
 90%   19
 95%   51
 98%   61
 99%   68
100%   68 (longest request)
[marta@marta-PC Practica4]$ █

```

Figura 3.1: Salida obtenida tras ejecutar `ab` contra el servidor con Ubuntu Server

4. **INSTALE *jmeter* Y SIGA EL TUTORIAL PARA HACER UN *Web Test Plan* REALIZANDO CAPTURAS DE PANTALLA Y COMENTÁNDOLAS. EN VEZ DE USAR LA WEB DE *jmeter*, HAGA EL EXPERIMENTO USANDO ALGUNA DE SUS MÁQUINAS VIRTUALES (PUEDE HACER UNA PÁGINA SENCILLA, USAR LAS PÁGINAS DE PHPMYADMIN, INSTALAR UN CMS, ETC)**

4.1. INSTALANDO *jmeter*

En primer lugar, debemos descargar el fichero ejecutable de *jmeter*. Para ello, usamos el siguiente comando:

```

Descargando el ejecutable de jmeter
wget http://apache.rediris.es//jmeter/binaries/apache-jmeter-2.13.tgz

```

Tras ello, nos descargamos el fichero *MD5*, para así verificar que no nos hemos descargado ningún *jmeter* “modificado”:

```

Descargando el fichero MD5 de jmeter
wget https://www.apache.org/dist/jmeter/binaries/apache-jmeter-2.13.tgz.md5

```

```
[marta@marta-PC Practica4]$ ab -c 5 -n 100 http://192.168.56.7/
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.56.7 (be patient).....done

Server Software:      Apache/2.4.6
Server Hostname:      192.168.56.7
Server Port:          80

Document Path:        /
Document Length:      3026 bytes

Concurrency Level:    5
Time taken for tests:  0.860 seconds
Complete requests:    100
Failed requests:       0
Total transferred:    325100 bytes
HTML transferred:     302600 bytes
Requests per second:  116.22 [#/sec] (mean)
Time per request:     43.021 [ms] (mean)
Time per request:     8.604 [ms] (mean, across all concurrent requests)
Transfer rate:        368.98 [Kbytes/sec] received

Connection Times (ms)
  min   mean[+/-sd] median   max
Connect:    0      0  0.2      0      1
Processing:  2     43 158.6      7     743
Waiting:    2     39 146.6      6     677
Total:       2     43 158.8      7     745

Percentage of the requests served within a certain time (ms)
 50%      7
 66%      8
 75%      9
 80%     10
 90%     11
 95%    721
 98%    734
 99%    745
100%    745 (longest request)
[marta@marta-PC Practica4]$
```

Figura 3.2: Salida obtenida tras ejecutar `ab` contra el servidor con CentOS

Para verificarlo, según [1], usamos el siguiente comando:

```
Verificando la descarga de jmeter
md5sum -c apache-jmeter-2.13.tgz.md5
```

Si todo ha ido bien, debemos ver un mensaje diciendonos que la suma coincide.

Descomprimos el archivo descargado, para ello, según [5], usamos el siguiente comando:

```
Descomprimiendo el archivo descargado
tar zxvf apache-jmeter-2.13.tgz
```

Tras descomprimir el archivo, vamos al directorio extraído y abrimos el archivo llamado *README* en el que encontraremos las instrucciones de instalación. Que básicamente consisten en irse al directorio *bin* y ejecutar el archivo *jmeter*:

```
Pasos para ejecutar jmeter una vez extraído el archivo tgz
1 $ cd apache-jmeter-2.13/
2 $ cat README
3 $ cd bin/
4 $ ./jmeter
```

Tras esto, se nos abrirá la ventana que se ve en la Figura 4.1.


```

[marta@marta-PC Practica4]$ ab -c 5 -n 100 http://192.168.56.5/
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.56.5 (be patient).....done

Server Software:      Microsoft-IIS/7.5
Server Hostname:      192.168.56.5
Server Port:          80

Document Path:        /
Document Length:      689 bytes

Concurrency Level:    5
Time taken for tests:  1.057 seconds
Complete requests:    100
Failed requests:      0
Total transferred:    93200 bytes
HTML transferred:     68900 bytes
Requests per second:  94.64 [#/sec] (mean)
Time per request:     52.829 [ms] (mean)
Time per request:     10.566 [ms] (mean, across all concurrent requests)
Transfer rate:        86.14 [Kbytes/sec] received

Connection Times (ms)
      min      mean[+/-sd] median    max
Connect:    0       0  0.3      0       2
Processing:  1    52 223.7      1    1055
Waiting:    1    52 223.7      1    1055
Total:       1    53 223.8      1    1056

Percentage of the requests served within a certain time (ms)
 50%      1
 66%      2
 75%      2
 80%      2
 90%      2
 95%     971
 98%    1056
 99%    1056
100%    1056 (longest request)
[marta@marta-PC Practica4]$

```

Figura 3.3: Salida obtenida tras ejecutar `ab` contra el servidor con Windows Server

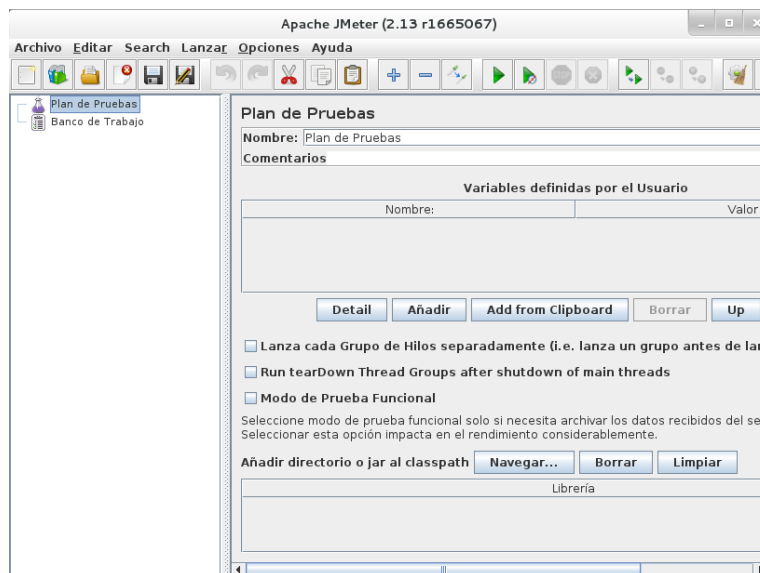


Figura 4.1: Ventana inicial de *jmeter*

4.2. CONSTRUYENDO NUESTRO PROPIO *Web Test Plan*

Los pasos a seguir, según [2], son:

1. En primer lugar, añadimos un *Grupo de hilos* para poder simular el número de usuarios, la frecuencia con la que mandarán solicitudes y cuántas solicitudes mandarán. Para ello, seguimos la ruta **Editar > Hilos (Usuarios) > Grupo de Hilos** (Figura 4.2).

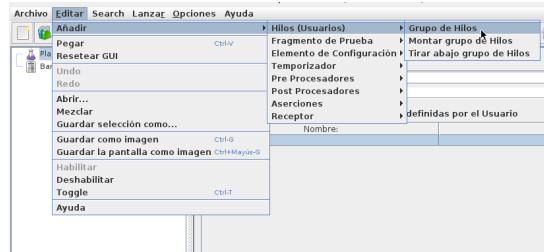


Figura 4.2: Ruta para crear un *Grupo de hilos*

2. Tras esto, se nos abrirá una ventana en la cual podremos modificar las propiedades del *Grupo de Hilos por defecto*. En nuestro caso serán las que se ven en la Figura 4.3: tendremos 5 usuarios mandando peticiones, todos los usuarios se crearán en 1 segundo (es decir, cada usuario tardará en crearse la quinta parte de un segundo) y repetiremos el test dos veces.

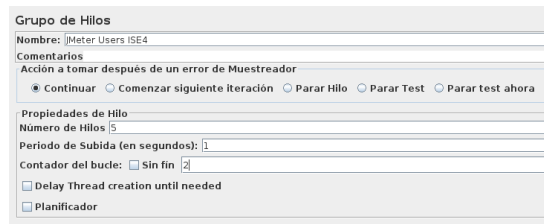


Figura 4.3: Propiedades para el grupo de hilos que usaremos como ejemplo

3. Una vez definidos los usuarios, debemos definir lo que harán. Para ello, especificaremos cómo serán las peticiones HTTP. Para ello hacemos click derecho en el *Grupo de Hilos* creado y seguimos la ruta **Añadir > Elemento de Configuración > Valores por Defecto para Petición HTTP** (Figura 4.4).

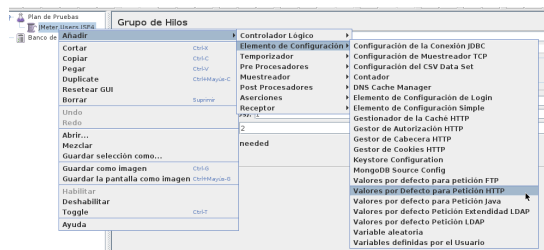


Figura 4.4: Ruta a seguir para especificar los valores por defecto de una petición HTTP para un grupo de hilos concreto

4. En este caso, vamos a dejar todos los valores por defecto y, en vez de introducir el servidor de *jmeter*, introduciremos el que tenemos instalado en Ubuntu Server (10.0.2.9). Así, nos debe quedar tal y como se ve en la Figura 4.5.

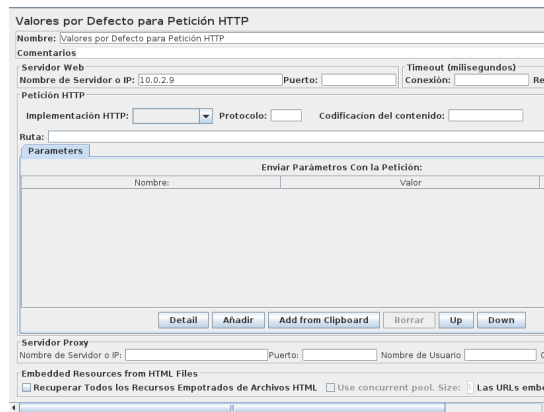
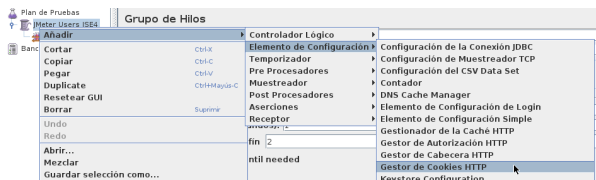
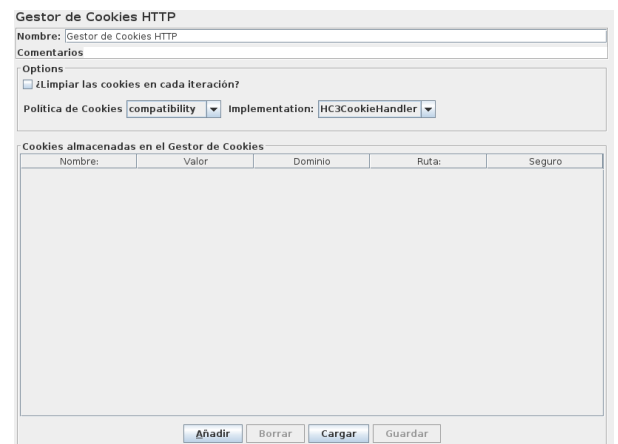


Figura 4.5: Propiedades para las peticiones HTTP a simular

- Opcionalmente, podemos añadir cookies a nuestro *test plan*. Para ello, hacemos click derecho en el *Grupo de hilos* creado y seguimos la ruta **Añadir > Elemento de configuración > Gestor de Cookies HTTP**. (Figura 6(a)). Dejamos todos los valores por defecto en la ventana que obtendremos (Figura 6(b)).



(a) Ruta a seguir para añadir cookies a nuestro *test plan*



(b) Ventana del gestor de cookies

Figura 4.6: Añadiendo soporte para cookies a nuestro *test plan*

- Ahora añadiremos las peticiones HTTP. Como en nuestro caso sólo tenemos una página de inicio (la que trae por defecto el servidor apache en Ubuntu), añadiremos sólo una petición HTTP a nuestra página de inicio. Para ello, hacemos click derecho en nuestro *Grupo de Hilos* y seguimos la ruta **Añadir > Muestreador > Petición HTTP**. (Figura 4.7).

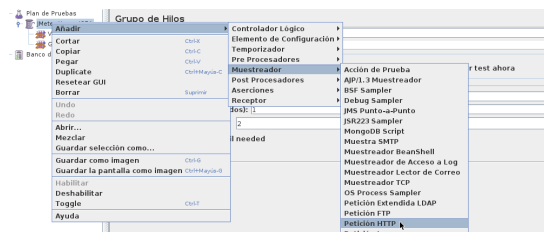


Figura 4.7: Ruta a seguir para añadir peticiones HTTP a nuestro *test plan*

- En la ventana que obtendremos, sólo tenemos que cambiar el nombre de la petición por *Home Page* y establecer la ruta a *“/”*. No debemos indicar el nombre del servidor, ya que lo hemos dejado indicado con anterioridad. Debe quedarnos tal y como se ve en la [Figura 4.8](#).

Figura 4.8: Valores que debe tener nuestra petición HTTP

- Por último, debemos añadir un *Receptor* para poder guardar los resultados en un fichero y representarlos gráficamente. Para ello, hacemos click derecho en nuestro *Grupo de hilos* y seguimos la ruta **Añadir > Receptor > Gráfico de Resultados**. ([Figura 4.9](#)).

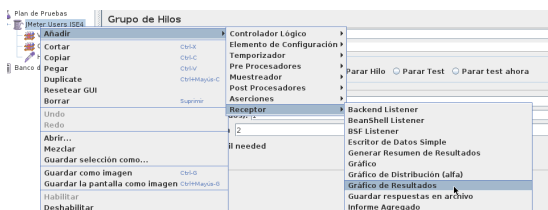


Figura 4.9: Ruta a seguir para añadir un *Receptor* a nuestro *test plan*

- En la ventana que obtendremos, debemos añadir una ruta para guardar los resultados de nuestro *test plan*. Podemos, o bien añadirla escribiendo una ruta, o bien seleccionar la ruta con el botón de Navegar. ([Figura 4.10](#)).

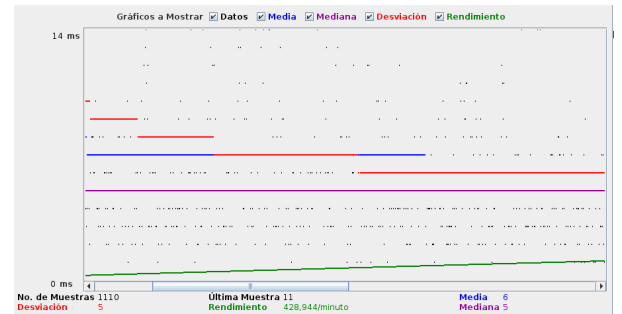
Figura 4.10: Ventana de configuración del Receptor de nuestro *test plan*

- Guardamos el plan de pruebas que hemos hecho y lo lanzamos siguiendo la ruta **Lanzar > Arrancar**. Cuando el test finalice, veremos los resultados reflejados en el gráfico de resultados

(Figura 11(a)). Como sólo lo ejecutamos dos veces, no obtenemos muchos valores para representar en nuestra gráfica, si cambiamos el parámetro *Contador del bucle* de nuestro grupo de hilos a un número mayor, obtendremos más valores para nuestra gráfica (Figura 11(b)). En los resultados obtenidos, podemos ver cómo conforme se va incrementando el número de peticiones, va aumentando el tiempo de respuesta del servidor.



(a) Gráfico obtenido tras ejecutar nuestro *test plan* dos veces



(b) Gráfico obtenido tras ejecutar nuestro *test plan* 200 veces

Figura 4.11: Ejecutando nuestro *test plan*

5. PROGRAME UN *benchmark* USANDO EL LENGUAJE QUE DESEE. EL *benchmark* DEBE INCLUIR: 1) OBJETIVO DEL *benchmark*, 2) MÉTRICAS (UNIDADES, VARIABLES, PUNTUACIONES, ETC), 3) INSTRUCCIONES PARA SU USO Y 4) EJEMPLO ANALIZANDO LOS RESULTADOS.

5.1. OBJETIVO DEL *benchmark*

El *benchmark* a realizar se trata de un script en *Python* usando **peewee** ([6]) para comparar *SQLite* y *MySQL*. En concreto, se medirán los siguientes parámetros:

- Velocidad para hacer una consulta concreta en una tabla grande ($V_{consultag}$).
- Velocidad para hacer una consulta concreta en una tabla pequeña ($V_{consultap}$).
- Velocidad para introducir un dato en una tabla pequeña ($V_{escriturap}$).
- Velocidad para introducir un dato en una tabla grande ($V_{escriturag}$).
- Velocidad para eliminar un dato en una tabla pequeña ($V_{borradop}$).
- Velocidad para eliminar un dato en una tabla grande ($V_{borradog}$).

Donde el tamaño de la tabla grande es de 3503 filas y el de la tabla pequeña, 25.

Con estos parámetros, el objetivo de este benchmark es averiguar qué gestor de bases de datos es mejor para alguien que realiza consultas y escrituras de forma habitual en una base de datos.

5.2. MÉTRICAS

Todos estos parámetros se medirán en **segundos** y para saber el resultado final se usará un diagrama de barras. Cada parámetro medido aparecerá en el diagrama de barras y se tomará como mejor aquel que haga el trabajo en menor número de segundos.

Para obtener cada parámetro, se hará la misma consulta cuatro veces y se hará la media del tiempo obtenido en cada operación. La primera consulta de todas se despreciará, debido a que la base de datos no se encontrará en cache y será más lenta.

5.3. INSTRUCCIONES DE USO

Para usar el benchmark, sólo debemos ejecutar el script en Python `benchmarkbd.py` y esperar a que termine su ejecución.

Ejecutando el benchmark

```
python benchmarkbd.py
```

5.4. EJEMPLO DE EJECUCIÓN

Tras ejecutar el benchmark, obtenemos la gráfica de la [Figura 5.1](#).

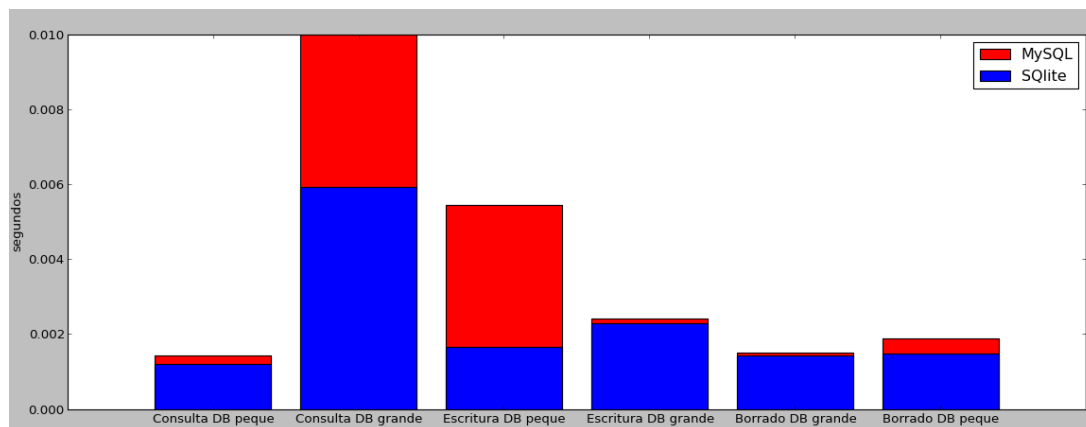


Figura 5.1: Resultados tras ejecutar el benchmark

A pesar de que históricamente *SQLite* era más lento que *MySQL* ([7]), en la última versión se ha visto bastante optimizado, llegando a conseguir mejores resultados que *MySQL*. Por tanto, el gestor de base de datos que mejores resultados ha dado es *SQLite*.

5.5. DETALLES SOBRE EL SCRIPT

Para generar los modelos¹ y así poder empezar a hacer consultas debemos ejecutar el siguiente comando:

Creando los modelos de la base de datos

```
pyhton -m pwiz -e mysql -u root -P Chinook > modelos.py
```

El script en Python, cambia de base de datos en tiempo de ejecución. Mas concretamente, cambia de una a otra cuando se han realizado los calculos de la primera:

benchmarkbd.py

```
1 #!/usr/bin/python
2 # -*- coding: utf-8 -*-
3
4 import peewee
```

¹En *peewee*, los **Modelos** representan cada tabla de la base de datos en una clase de Python.

```

5 from peewee import *
6 import modelos
7 from modelos import *
8 import sys
9 import time
10 import matplotlib.pyplot as plt
11 plt.rcParams()
12 import numpy as np
13 import matplotlib.pyplot as plt
14
15 database_proxy = Proxy() # Create a proxy for our db.
16 mysql_calculado = False
17
18 class BaseModel(Model):
19     class Meta:
20         database = database_proxy # Use proxy for our DB.
21
22 database = MySQLDatabase('Chinook', user='root', passwd='1234')
23
24 database_proxy.initialize(database)
25
26 # Listar todo el contenido de una tabla grande
27 def lista_track():
28     for track in Track.select():
29         print(track.trackid, track.name, track.albumid, track.mediatypeid,
30               track.genreid, track.composer, track.milliseconds, track.bytes,
31               track.unitprice)
32
33 # Listar todo el contenido de una tabla pequeña
34 def lista_genero():
35     for gen in Genre.select():
36         print(gen.genreid, gen.name)
37
38 # Hacer una consulta concreta a una tabla grande
39 def filtrar_track_composer(composer):
40     consulta = Track.select().where(Track.composer == composer)
41     for track in consulta:
42         print(track.trackid, track.name, track.albumid, track.mediatypeid,
43               track.genreid, track.composer, track.milliseconds, track.bytes,
44               track.unitprice)
45
46 # Hacer una consulta concreta a una tabla pequeña
47 def filtrar_generos_nombre(nombre):
48     consulta = Genre.select().where(Genre.name == nombre)
49     for gen in consulta:
50         print(gen.genreid, gen.name)
51
52 # Introducir un dato en la base de datos pequeña
53 def introducir_genero(nombre, num):
54     nuevo_gen = Genre.create(name=nombre, genreid=num)
55
56 # Introducir un dato en la base de datos grande
57 def introducir_track(idcancion, nombre, album, mediatype, genero, compositor, ms, bt, precio):
58     nueva_track = Track.create(trackid=idcancion, name=nombre, albumid=album, mediatypeid=mediat

```

```

59         genreid=genero, composer=compositor, milliseconds=ms, bytes=bt, unitprice=precio)
60
61 def borrar_track (idcancion):
62     q = Track.delete().where(Track.trackid == idcancion)
63     q.execute()
64
65 def borrar_genero (idgenero):
66     q = Genre.delete().where(Genre.genreid == idgenero)
67     q.execute()
68
69 def pruebas():
70     resultados = []
71     tiempos = []
72
73     # Consultar un dato en una base de datos pequeña
74     for x in range(0,4):
75         antes = time.time()
76         filtrar_generos_nombre("World")
77         despues = time.time()
78         tiempo = despues - antes
79         tiempos.append(tiempo)
80
81     tiempos.reverse()
82     tiempos.pop()
83     resultados.append(sum(tiempos)/len(tiempos))
84     del tiempos[:] # limpiamos la tabla de tiempos
85
86     # Consultar un dato en una base de datos grande
87     for x in range(0,4):
88         antes = time.time()
89         filtrar_track_composer("Antonio Vivaldi")
90         despues = time.time()
91         tiempo = despues - antes
92         tiempos.append(tiempo)
93
94     tiempos.reverse()
95     tiempos.pop()
96     resultados.append(sum(tiempos)/len(tiempos))
97     del tiempos[:] # limpiamos la tabla de tiempos
98
99     #Introducir un dato en la base de datos pequeña
100    generos = [(26, "Progressive Death Metal"), (27, "Symphonic Metal"),
101               (28, "Hard Rock"), (29, "Death Metal")]
102
103    for x in range(0,4):
104        antes = time.time()
105        introducir_genero(generos[x][1], generos[x][0])
106        despues = time.time()
107        tiempo = despues - antes
108        tiempos.append(tiempo)
109
110    tiempos.reverse()
111    tiempos.pop()
112    resultados.append(sum(tiempos)/len(tiempos))

```



```

113     del tiempos[:] # limpiamos la tabla de tiempos
114
115     # Introducir un dato en la base de datos grande
116     canciones = [(3504, "The Moor", 347, 4, 26, "Opeth", 685000, 4744929, 0.99),
117                  (3505, "Cry For The Moon", 347, 4, 27, "Epica", 350000, 4744929, 0.99),
118                  (3506, "Shout At The Devil", 347, 4, 28, "Motley Crue", 195000, 4744929, 0.99),
119                  (3507, "War Eternal", 347, 4, 29, "Arch Enemy", 262000, 4744929, 0.99)]
120
121     for x in range(0,4):
122         antes = time.time()
123         introducir_track(canciones[x][0], canciones[x][1], canciones[x][2],
124                         canciones[x][3], canciones[x][4], canciones[x][5], canciones[x][6],
125                         canciones[x][7], canciones[x][8])
126         despues = time.time()
127         tiempo = despues - antes
128         tiempos.append(tiempo)
129
130     tiempos.reverse()
131     tiempos.pop()
132     resultados.append(sum(tiempos)/len(tiempos))
133     del tiempos[:] # limpiamos la tabla de tiempos
134
135     # Borrar un dato de una base de datos grande
136     id_canciones = [3506,3504,3507,3505]
137
138     for x in range(0,4):
139         antes = time.time()
140         borrar_track(id_canciones[x])
141         despues = time.time()
142         tiempo = despues - antes
143         tiempos.append(tiempo)
144
145     tiempos.reverse()
146     tiempos.pop()
147     resultados.append(sum(tiempos)/len(tiempos))
148     del tiempos[:]
149
150     # Borrar un dato de una base de datos pequeña
151     id_generos = [28,26,29,27]
152
153     for x in range(0,4):
154         antes = time.time()
155         borrar_genero(id_generos[x])
156         despues = time.time()
157         tiempo = despues - antes
158         tiempos.append(tiempo)
159
160     tiempos.reverse()
161     tiempos.pop()
162     resultados.append(sum(tiempos)/len(tiempos))
163     del tiempos[:]
164
165     return resultados
166

```

```

167 def calcular_res (pruebas):
168     resultados = []
169     resultados.append(((pruebas[0] + pruebas[1])/2) + pruebas[4])
170     resultados.append(((pruebas[2] + pruebas[3])/2) + pruebas[5])
171     mysql_calculado = True
172     return resultados
173
174 if __name__ == '__main__':
175     res_mysql = pruebas()
176
177     mysql_calculado = True
178
179     if mysql_calculado:
180         database = SQLiteDatabase('Chinook')
181
182     res_sqlite = pruebas()
183
184     parametros = ('Consulta DB peque', 'Consulta DB grande', 'Escritura DB peque',
185                  'Escritura DB grande', 'Borrado DB grande', 'Borrado DB peque')
186     y_pos = np.arange(len(parametros))
187     plt.bar(y_pos, np.array(res_mysql), align='center', color='r', label='MySQL')
188     plt.bar(y_pos, np.array(res_sqlite), align='center', color='b', label='Sqlite')
189     plt.xticks(y_pos, parametros)
190     plt.ylabel('segundos')
191     plt.legend()
192     plt.show()

```

6. CUESTIONES OPCIONALES

6.1. SELECCIONE, INSTALE Y EJECUTE UN BENCHMARK, COMENTE LOS RESULTADOS. ATENCIÓN: NO ES LO MISMO UN BENCHMARK QUE UNA SUITE, INSTALE UN BENCHMARK

En nuestro caso, de toda la lista de benchmarks disponibles, hemos elegido uno para medir la capacidad de la CPU para comprimir video usando la librería x264 ([3]). Para instalarlo, según [8], usamos el siguiente comando:

```

_____ Instalando el benchmark x264 _____
sudo phoronix-test-suite benchmark x264

```

tras esto, empezará a descargar e instalar el benchmark y sus dependencias. Una vez instalado nos mostrará el software y hardware de nuestra máquina y nos preguntará si queremos guardar la información en un fichero y si le decimos que no, empezará a hacer pruebas. En la prueba que hemos hecho en la Figura 6.1, el benchmark nos da como conclusión que podríamos codificar unos 26 frames por segundo.

6.3. LEA EL ARTÍCULO DE COMPARACIÓN ENTRE *Jmeter* Y *Gatling* ELABORADO POR LA EMPRESA *Flood.io* Y ELABORE UN BREVE RESUMEN

En *Flood.io* no se fían de los benchmark “competitivos” ya que están hechos para favorecer a las empresas que lo patrocinan y no dan resultados realmente fiables. Por eso, se pasaron al lado del código abierto y probaron *Jmeter* y *Gatling*.

```
Would you like to save these test results (Y/n): n

x264 2015-11-02:
pts/x264-2.0.0
Test 1 of 1
Estimated Trial Run Count: 5
  Started Run 1 @ 15:22:00
  Started Run 2 @ 15:22:28
  Started Run 3 @ 15:22:53
  Started Run 4 @ 15:23:18
  Started Run 5 @ 15:23:43 [Std. Dev: 4.74%]
  Started Run 6 @ 15:24:08 [Std. Dev: 4.60%]
  Started Run 7 @ 15:24:33 [Std. Dev: 4.27%]
  Started Run 8 @ 15:24:57 [Std. Dev: 3.98%]
  Started Run 9 @ 15:25:22 [Std. Dev: 3.73%]
  Started Run 10 @ 15:25:48 [Std. Dev: 3.52%]

Test Results:
23.37
26.14
26.23
25.98
25.89
26.69
26.31
26.18
25.98
25.77

Average: 25.85 Frames Per Second
```

Figura 6.1: Salida del benchmark x264 mientras realiza sus pruebas

Tras hacer el *test plan* con los dos benchmark, en el cual había 10.000 usuarios y 30.000 peticiones por minuto, el resultado obtenido es que ambos benchmark son bastante parecidos pero tienen una diferencia: *Gatling* no es capaz de guardar el tamaño en bytes de la respuesta, pero sin embargo, a pesar de que *Jmeter* sí lo es consume más recursos de CPU y memoria.

El artículo concluye diciendo que ambos benchmark son muy parecidos en términos de concurrencia y *throughput* y que la elección de uno u otro es puramente subjetiva y hecha sobre alguna otra característica que cada herramienta incluya.

REFERENCIAS

- [1] APACHE, *How to verify the integrity of the downloaded file?* Disponible en http://www.openoffice.org/download/checksums.html#hash_linux. Consultado el 7/12/2015.
- [2] —, *User's manual: Building a web test plan.* Disponible en <http://jmeter.apache.org/usermanual/build-web-test-plan.html>. Consultado el 7/12/2015.
- [3] V. ORGANIZATION, *x264, the best h.264/avc encoder.* Disponible en <https://www.videolan.org/developers/x264.html>. Consultado el 5/12/2015.
- [4] L. M. PAGES, *ab: Apache http server benchmarking tool.* Disponible en <http://linux.die.net/man/1/ab>. Consultado el 6/12/2015.
- [5] —, *tar: manual page for tar 1.23.* Disponible en <http://linux.die.net/man/1/tar>. Consultado el 7/12/2015.
- [6] PEEWEE, *peewee 2.7.4 documentation.* Disponible en <http://peewee.readthedocs.org/en/latest/index.html>. Consultado el 9/12/2015.
- [7] SQLITE, *Sqlite database speed comparison.* Disponible en <https://sqlite.org/speed.html>. Consultado el 18/12/2015.

- [8] U. WIKI, *Phoronix test suite howto*. Disponible en <https://wiki.ubuntu.com/PhoronixTestSuite>. Consultado el 2/12/2015.