

INGENIERÍA DE SERVIDORES: PRÁCTICA 4

---

## Benchmarks

---

Montserrat Rodríguez Zamorano

20 de junio de 2016

## Índice

1. Instale la aplicación. ¿Qué comando permite listar los benchmarks disponibles?	1
2. De los parámetros que le 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?	1
3. Ejecute <code>ab</code> contra las tres máquinas virtuales (desde el SO anfitrión a las máquinas virtuales de la red local, en Ubuntu, CentOS y WS) 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?	3
4. Instale y siga el tutorial en <a href="http://jmeter.apache.org/usermanual/buildweb-test-plan.html">http://jmeter.apache.org/usermanual/buildweb-test-plan.html</a> realizando capturas de pantalla y comentándolas. En vez de usar la web de <code>jmeter</code> , haga el experimento usando alguna de sus máquinas virtuales (Puede hacer una página sencilla, usar las páginas de <code>phpmyadmin</code> , instalar un CMS, etc.).	5
5. Programe un benchmark usando el lenguaje que desee.	9
5.1. Objetivo del benchmark . . . . .	9
5.2. Métricas . . . . .	9
5.3. Instrucciones para su uso . . . . .	10
5.4. Ejemplo de uso analizando los resultados . . . . .	10
6. Cuestiones opcionales	13
6.1. Cuestión opcional 1: Seleccione, instale y ejecute uno, comente los resultados. . . . .	13
6.2. Cuestión opcional 2: ¿Qué es Scala? Instale Gatling y pruebe los escenarios por defecto.	14
6.3. Cuestión opcional 3: Lea el artículo y elabore un breve resumen. . . . .	16

## Índice de figuras

1. Listar los benchmarks disponibles . . . . .	1
2. Parámetros que se le pueden pasar al comando <code>ab</code> . . . . .	2
3. Ejecución del comando <code>ab</code> . . . . .	2
4. <code>ab</code> crea un sólo proceso o hebra . . . . .	3
5. Ejecución del comando <code>ab</code> contra Ubuntu Server . . . . .	3
6. Ejecución del comando <code>ab</code> contra Windows . . . . .	4
7. Ejecución del comando <code>ab</code> contra CentOS . . . . .	4
8. Instalación de <code>jmeter</code> . . . . .	6
9. Interfaz gráfica de <code>jmeter</code> . . . . .	6
10. Crear un grupo de hebras con <code>jmeter</code> . . . . .	6
11. Configurar el grupo de hilos con <code>jmeter</code> . . . . .	7
12. Especificar los valores por defecto para una petición HTTP I . . . . .	7
13. Especificar los valores por defecto para una petición HTTP II . . . . .	7
14. Añadir el gestor de Cookies HTTP . . . . .	8
15. Añadir las peticiones HTTP . . . . .	8
16. Añadir el receptor en <code>jmeter</code> . . . . .	9
17. Consultando resultados en <code>jmeter</code> I . . . . .	9
18. Consultando resultados en <code>jmeter</code> II . . . . .	10
19. Salida por pantalla tras la ejecución del benchmark . . . . .	11
20. Imposibilidad de instalar un benchmark por falta de espacio . . . . .	13
21. Testeando <code>openssl</code> en una máquina virtual . . . . .	13
22. Mejores resultados en la ejecución del benchmark en función del procesador . . . . .	14
23. Ejecución del test en OS X . . . . .	14
24. Iniciando el servicio Gatling . . . . .	15
25. Escenarios por defecto en Gatling . . . . .	15

26.	Probando el escenario 0 por defecto en <b>Gatling</b> . . . . .	16
27.	Resultados de la ejecución con <b>Gatling</b> I . . . . .	16
28.	Resultados de la ejecución con <b>Gatling</b> II . . . . .	17
29.	Resultados de la ejecución con <b>Gatling</b> III . . . . .	17

## 1. Instale la aplicación. ¿Qué comando permite listar los benchmarks disponibles?

Se realiza la instalación de Phoronix Suite en Ubuntu Server. Para instalar la aplicación se utilizará el gestor de paquetes `apt-get`:

```
sudo apt-get install phoronix-test-suite
```

Para listar los benchmarks disponibles se puede utilizar el siguiente comando [1]. El resultado puede verse en la figura (1):

```
phoronix-test-suite list-available-tests
```

```
minim@minim:~$ phoronix-test-suite list-available-tests
Phoronix Test Suite v4.8.3
Available Tests

pts/aio-stress          - AIO-Stress          Disk
pts/apache             - Apache Benchmark    System
pts/apitest            - APITest             Graphics
pts/apitrace           - APITrace            Graphics
pts/askap              - ASKAP tConvolveCuda Graphics
pts/battery-power-usage - Battery Power Usage System
pts/bioshock-infinite  - BioShock Infinite   Graphics
pts/blake2             - BLAKE2              Processor
pts/blogbench          - BlogBench           Disk
pts/bork               - Bork File Encrypter Processor
pts/botan              - Botan               Processor
pts/build-apache       - Timed Apache Compilation Processor
pts/build-boost-interprocess - Timed Boost Interprocess Compilation Processor
pts/build-eigen         - Timed Eigen Compilation Processor
pts/build-firefox       - Timed Firefox Compilation Processor
pts/build-imagemagick   - Timed ImageMagick Compilation Processor
pts/build-linux-kernel  - Timed Linux Kernel Compilation Processor
pts/build-mplayer       - Timed MPlayer Compilation Processor
pts/build-php           - Timed PHP Compilation Processor
pts/build-webkitgtk     - Timed WebKitGTK Compilation Processor
pts/bullet             - Bullet Physics Engine Processor
pts/byte               - BYTE Unix Benchmark Processor
pts/c-ray              - C-Ray               Processor
pts/cachebench         - CacheBench          Processor
pts/caffe              - Caffe AlexNet       System
pts/cairo-demos        - Cairo Performance Demos Graphics
pts/cairo-perf-trace    - cairo-perf-trace    Graphics
pts/clomp              - CLOMP               Processor
pts/coh2               - Company of Heroes 2 Graphics
pts/compilebench       - Compile Bench       Disk
pts/compress-7zip       - 7-Zip Compression   Processor
pts/compress-gzip       - Gzip Compression    Processor
pts/compress-lzma       - LZMA Compression    Processor
pts/compress-pbzip2     - Parallel BZIP2 Compression Processor
pts/corebreach         - CoreBreach          Graphics
```

Figura 1: Listar los benchmarks disponibles

## 2. De los parámetros que le 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?

La herramienta `ab` viene integrada en la instalación de Apache, por lo que no es necesaria ninguna instalación adicional. Se ejecuta `man ab` en la línea de comandos para ver las opciones (ver (2)).

Se consultan en el manual la descripción de las opciones `-c` y `-n`.

- `c`: *conurrencia*. Permite realizar tantas peticiones a la vez como se le indiquen. Por defecto es 1. Por ejemplo, `-c 5` permite atender 5 peticiones concurrentemente.
- `n`: *peticiones*. Permite realizar tantas peticiones como se le indiquen durante la sesión de benchmark. Por defecto es 1. En este caso, `-n 100` permite atender 100 peticiones durante la sesión

```

AB(1)tion/x-www-form-urlencoded" ab AB(1)
Content-Type: text/plain
NAME
  ab - Apache HTTP server benchmarking tool
SYNOPSIS
  ab [-A auth-username:password] [-b window-size] [-B local-address]
  [-c concurrency] [-C cookie-name=value] [-d] [-e csv-file] [-f
  protocol] [-g gnuplot-file] [-h] [-H custom-header] [-i] [-k]
  [-l] [-m HTTP-method] [-n requests] [-p POST-file] [-P
  proxy-auth-username:password] [-q] [-r] [-s timeout] [-S] [-t
  timelimit] [-T content-type] [-u PUT-file] [-v verbosity] [-V]
  [-w <table>] [-x <table>-attributes] [-X proxy[:port]] [-y
  <table>-attributes] [-z <table>-attributes] [-Z ciphersuite]
  [http(s)://hostname[:port]/path]

```

Figura 2: Parámetros que se le pueden pasar al comando `ab`

Se prueba ahora a ejecutar el comando junto con los parámetros anteriores, teniendo en cuenta que la sintaxis del comando es `ab <options>http(s)://hostname[:port]/path`. En este caso, la dirección será la de la máquina virtual de Ubuntu Server, en la que se ha iniciado el servicio Apache. El resultado puede verse en la figura (3).

```

Server Software: Apache/2.4.7
Server Hostname: 192.168.1.114
Server Port: 80

Document Path: /
Document Length: 752 bytes

Concurrency Level: 5
Time taken for tests: 1.669 seconds
Complete requests: 1000
Failed requests: 0
Total transferred: 941000 bytes
HTML transferred: 752000 bytes
Requests per second: 599.19 [#/sec] (mean)
Time per request: 8.345 [ms] (mean)
Time per request: 1.669 [ms] (mean, across all concurrent requests)
Transfer rate: 558.62 [Kbytes/sec] received

Connection Times (ms)
              min    mean[+/-sd] median    max
Connect:     0      4  64.7      0   1024
Processing:   0      4  16.7      3    430
Waiting:      0      4  16.6      3    427
Total:        1      8  66.8      4   1030

Percentage of the requests served within a certain time (ms)
 50%    4
 66%    4
 75%    4
 80%    4
 90%    5
 95%    7
 98%   10
 99%   14
100% 1030 (longest request)
sh-3.2#

```

Figura 3: Ejecución del comando `ab`

Los resultados de la ejecución se comentarán en más detalle en la siguiente cuestión. Para ver cuántos procesos o hebras crea `ab` en el cliente, se ejecuta el siguiente comando [2]:

```
ps -ef | grep ab | wc -l
```

Se ejecuta el comando antes y después de ejecutar `ab` y el resultado puede verse en la figura (4).

Es decir, sólo se crea un proceso. En cuanto a la sintaxis del comando, ya es conocido que `ps` muestra todos los procesos por pantalla, `grep` busca una cadena de caracteres (en este caso, `ab`). La última parte del comando, `wc -l`, sirve para contar el número de líneas que se mostrarían por pantalla [3].

```

minim : bash - Konsole
File Edit View Bookmarks Settings Help
minim@minim:~$ ps -ef | grep ab | wc -l
3
minim@minim:~$ ps -ef | grep ab | wc -l
4

```

Figura 4: ab crea un sólo proceso o hebra

3. Ejecute ab contra las tres máquinas virtuales (desde el SO anfitrión a las máquina virtuales de la red local, en Ubuntu, CentOS y WS) 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?

En esta ocasión, se ha aumentado el número de peticiones a 10000. Los resultados de la ejecución pueden verse de ejecutar el comando contra las máquinas de Ubuntu, WS y CentOS pueden verse en las capturas (5), (6) y (7), respectivamente.

```

Server Software:      Apache/2.4.7
Server Hostname:      192.168.56.101
Server Port:          80

Document Path:        /
Document Length:      11510 bytes

Concurrency Level:     5
Time taken for tests:  18.892 seconds
Complete requests:     10000
Failed requests:        0
Total transferred:     117830000 bytes
HTML transferred:      115100000 bytes
Requests per second:   918.08 [#/sec] (mean)
Time per request:      5.446 [ms] (mean)
Time per request:      1.089 [ms] (mean, across all concurrent requests)
Transfer rate:         10564.24 [Kbytes/sec] received

Connection Times (ms)
  min  mean[+/-sd] median  max
Connect:    0    1 20.4    0   1019
Processing:  0    5  9.3    2    304
Waiting:    0    2  5.3    2    302
Total:       1    5 22.4    3   1025

Percentage of the requests served within a certain time (ms)
 50%    3
 66%    4
 75%    4
 80%    5
 90%   10
 95%   19
 98%  16.32m
 99%   45
100%  1025 (longest request)
MacBook-Air-de-Montse:~ montse$

```

Figura 5: Ejecución del comando ab contra Ubuntu Server

Se comentan de modo general algunos aspectos de las estadísticas. Por ejemplo, en primer lugar, encontramos la versión de Apache utilizada (IIS en el caso de Windows), la dirección IP del servidor y puerto (en este caso, puerto 80 por tratarse de Apache).

Más adelante se pueden ver otros aspectos a comentar, como los siguientes:

- **Concurrency Level:** número de procesos que se ejecutan concurrentemente, en este caso, 5, tal

```
montse — bash — 105x44
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
Completed 10000 requests
Finished 10000 requests

Server Software:      Microsoft-IIS/7.5
Server Hostname:      192.168.1.116
Server Port:          80

Document Path:        /
Document Length:      689 bytes

Concurrency Level:    5
Time taken for tests:  4.827 seconds
Complete requests:    10000
Failed requests:      0
Total transferred:    9320000 bytes
HTML transferred:     6890000 bytes
Requests per second:  2071.00 [#/sec] (mean)
Time per request:     2.413 [ms] (mean)
Time per request:     0.483 [ms] (mean, across all concurrent requests)
Transfer rate:        1885.66 [Kbytes/sec] received

Connection Times (ms)
  min  mean[+/-sd] median  max
Connect:    0   1  22.7   0  1134
Processing:  0   1   2.1   2   206
Waiting:    0   1   2.1   2   206
Total:      0   2  22.8   2  1135

Percentage of the requests served within a certain time (ms)
 50%    2
 66%    2
 75%    2
 80%    2
 90%    2
 95%    3
 98%    3
 99%    3
100%  1135 (longest request)
MacBook-Air-de-Montse:~ montse$
```

Figura 6: Ejecución del comando `ab` contra Windows

```
Finished 10000 requests

Server Software:      Apache/2.4.6
Server Hostname:      10.0.2.8
Server Port:          80

Document Path:        /
Document Length:      4897 bytes

Concurrency Level:    5
Time taken for tests:  8.328 seconds
Complete requests:    10000
Failed requests:      0
Non-2xx responses:    10000
Total transferred:    51790000 bytes
HTML transferred:     48970000 bytes
Requests per second:  1200.80 [#/sec] (mean)
Time per request:     4.164 [ms] (mean)
Time per request:     0.833 [ms] (mean, across all concurrent requests)
Transfer rate:        6073.20 [Kbytes/sec] received

Connection Times (ms)
  min  mean[+/-sd] median  max
Connect:    0   0   0.2   0    6
Processing:  1   4   1.2   4   25
Waiting:    0   4   1.2   3   18
Total:      2   4   1.3   4   26

Percentage of the requests served within a certain time (ms)
 50%    4
 66%    4
 75%    4
 80%    4
 90%    5
 95%    6
 98%    7
 99%    9
100%   26 (longest request)
minim@minim:~$
```

Figura 7: Ejecución del comando `ab` contra CentOS

y como se indicó en la ejecución del comando.

- **Time taken for tests:** tiempo total invertido en la ejecución del test.
- **Complete requests:** número de peticiones completas.
- **Failed request:** número de peticiones fallidas.
- **Requests per second:** número de peticiones atendidas por segundo.
- **Connection times:** tiempos de conexión, procesamiento y espera.
- **Percentage of the request served within a certain time (ms):** tanto por ciento de las peticiones completadas en un determinado tiempo.

Serán algunos de estos parámetros los que se utilizarán para comparar las tres ejecuciones. Intuitivamente, el tiempo de ejecución parecería la opción más razonable para medir el rendimiento. Por ejemplo, es claro que los test tardan más tiempo en ejecutarse en Ubuntu, que completa menos peticiones por segundo (918,08 frente a las 1200,80 que completa en CentOS y las 2071,80 en Windows). La conclusión, si tenemos en cuenta sólo el tiempo de ejecución, es que la máquina que utiliza Windows es 2.25 veces más rápida que la máquina de Ubuntu, la más lenta, pero hay más factores a tener en cuenta.

Por ejemplo, el número de bytes transferidos en cada una es diferente: el total de bytes transferidos en Ubuntu por petición es de 11783 bytes, en Windows de 932 bytes, y en CentOS de 5179 bytes, lo cual podría justificar el mayor tiempo de ejecución de la máquina que utiliza Ubuntu. Otro dato a tener en cuenta es la tasa de transferencia, es decir, el número de bytes transferidos por segundo, de 10564.24 Kbytes/s en Ubuntu, 1885.66 Kbytes/s en Windows y 6073.20 Kbytes/s en CentOS. Es decir, Ubuntu, pese a ser la máquina con un mayor tiempo de ejecución, es la que transfería un mayor número de bytes por segundo, 5.6 veces más que la máquina de Windows.

Puede estudiarse también el campo **Percentage of the request served within a certain time (ms)**, en la que puede verse si ha habido una petición más lenta que otra. A modo de ejemplo, en la máquina de Windows, puede verse cómo el 99 % de las peticiones se completaron en 3 ms, pero tardó 1132 ms en completar el último 1 %.

En conclusión, no puede asegurarse que una de las tres máquinas virtuales proporcione mejores resultados sin estudiar más detalladamente estos datos.

#### 4. **Instale y siga el tutorial en <http://jmeter.apache.org/usermanual/buildweb-test-plan.html> 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.).**

Se instalará la instalación de **Jmeter** en OS X utilizando el gestor de paquetes **brew** (8), mediante la siguiente línea de comandos:

```
brew install jmeter
```

Una vez hecho esto, se consulta en la página indicada el tutorial. Se ejecuta el comando **Jmeter** para iniciar la aplicación, y nos aparecerá una ventana como la que puede verse en (9).

Los pasos a seguir serán los siguientes:

- Crear un grupo de hilos. Esto le indica a **Jmeter** algunos parámetros como el número de usuarios a simular, tiempo de envío de peticiones, etc. Para ello, hacemos clic derecho en el plan de prueba - añadir - hilos - grupo de hilos (10).



```

montse — bash — 80x24
Last login: Fri Jun 10 20:00:51 on ttys000
MacBook-Air-de-Montse:~ montse$ brew install jmeter
=> Downloading https://homebrew.bintray.com/bottles/jmeter-3.0.yosemite.bottle.
##### 100,0%
=> Pouring jmeter-3.0.yosemite.bottle.tar.gz
   /usr/local/Cellar/jmeter/3.0: 2,781 files, 94.5M
MacBook-Air-de-Montse:~ montse$

```

Figura 8: Instalación de jmeter

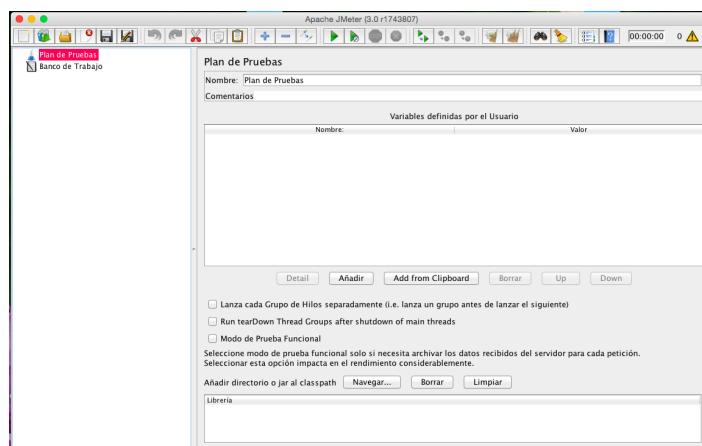


Figura 9: Interfaz gráfica de jmeter

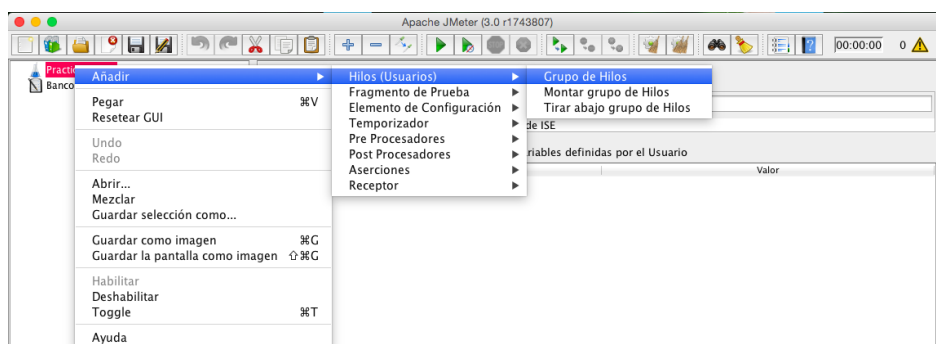


Figura 10: Crear un grupo de hebras con jmeter

- Se modifican las propiedades por defecto. Se selecciona el grupo de hilos creado y se modifican los parámetros a placer. En este caso, se han seguido las indicaciones del tutorial: número de hilos: 5, periodo de subida: 1 segundo, contador del bucle: 2. Este último valor indica el número de veces a repetir el test. De esta forma, la configuración del grupo de hebras ha quedado como se indica en la figura (11).
- Una vez definidos los usuarios, se definen las tareas a realizar. En primer lugar, se especifican los valores por defecto para una petición HTTP. Hacemos clic derecho en el grupo de hilos - añadir - elemento de configuración - valores por defecto para petición HTTP (12)
- Se completan los campos. En este caso, se ha añadido 192.168.1.114/phpmyadmin/ como IP. Es el único campo en el que se especificará algún valor (13).

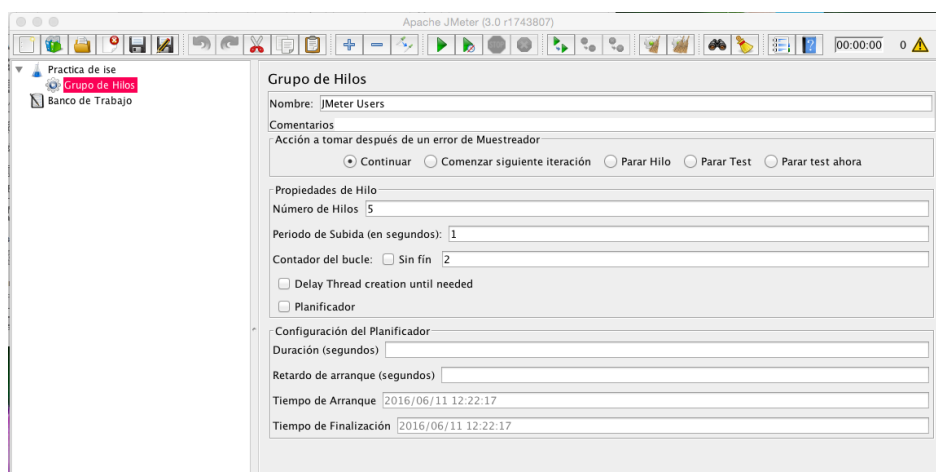


Figura 11: Configurar el grupo de hilos con jmeter

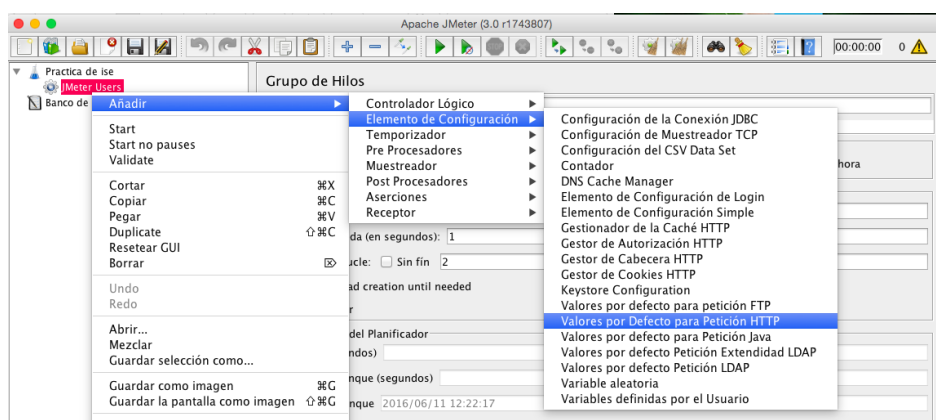


Figura 12: Especificar los valores por defecto para una petición HTTP I

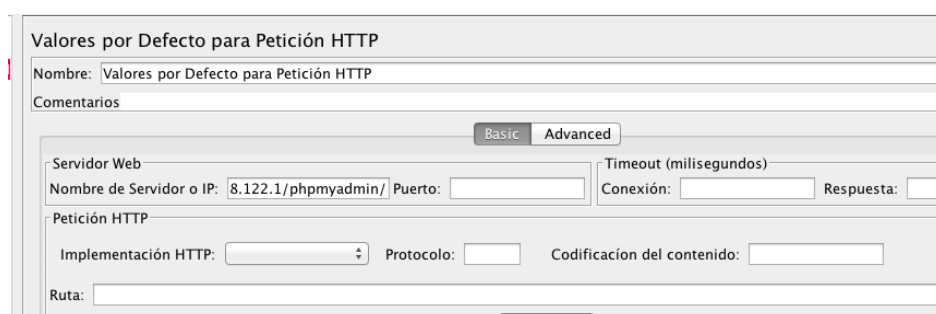


Figura 13: Especificar los valores por defecto para una petición HTTP II

- Puede añadirse o no el gestor de Cookies HTTP. Para añadirlo, se hace clic en añadir - elemento de configuración - gestor de cookies HTTP (ver (14)).
- Se hará una petición HTTP. En la primera de ellas, se cambiará el nombre del campo a *Home page* y se establecerá la ruta a / (15).
- El último paso es añadir un receptor para almacenar los resultados de las peticiones. En este

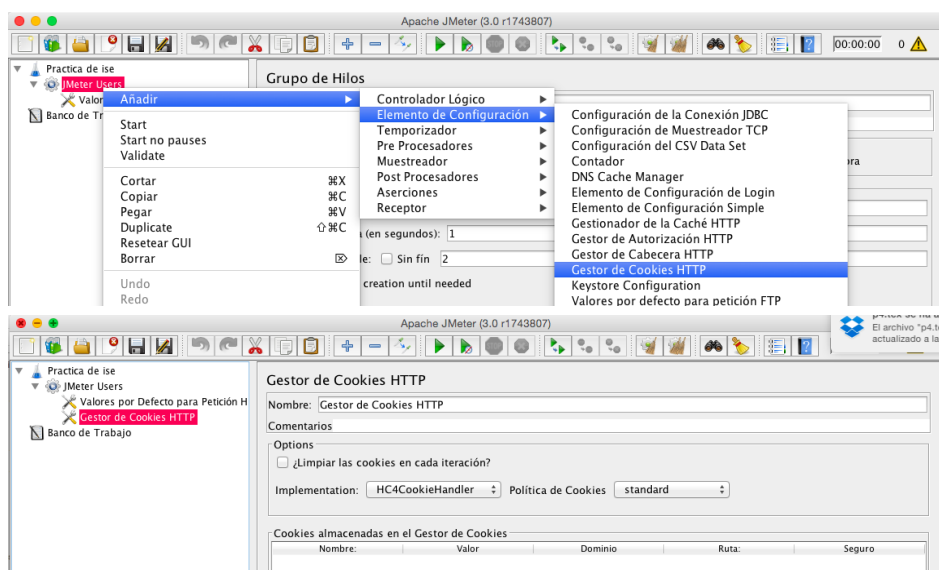


Figura 14: Añadir el gestor de Cookies HTTP

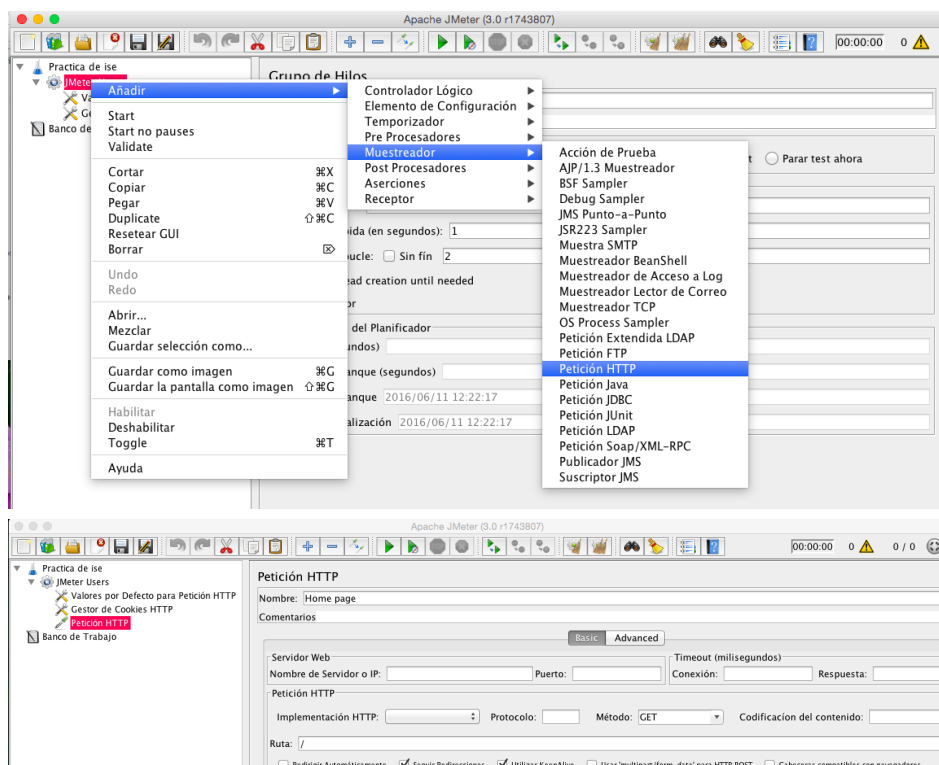


Figura 15: Añadir las peticiones HTTP

caso, se quieren presentar los resultados de forma visual, por lo que se añadirá un receptor gráfico de resultados (16). Para que se viera con más claridad los resultados, se ha cambiado el contador a *sin fin*. Empezarán a aparecer resultados inmediatamente (17). La diferencia con los resultados en el caso de la web de Jmeter es notable (18). Aunque aparece información estadística relevante (media, mediana, desviación), lo más interesante es el rendimiento de la máquina. Como puede

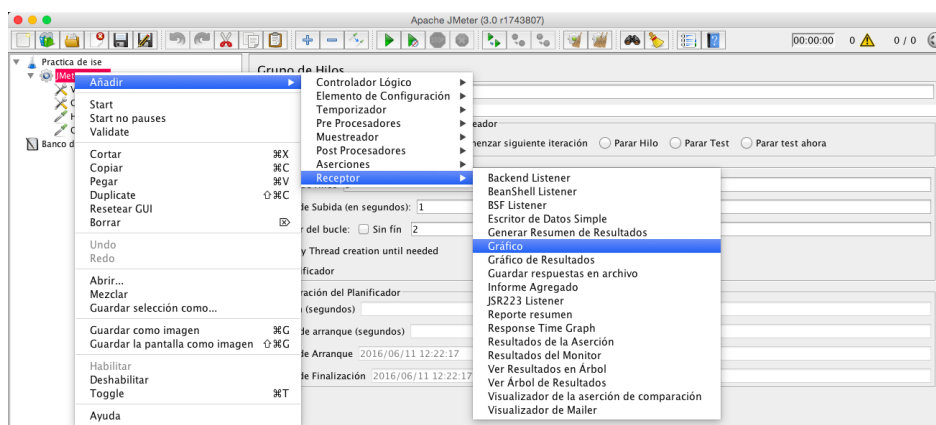


Figura 16: Añadir el receptor en jmeter

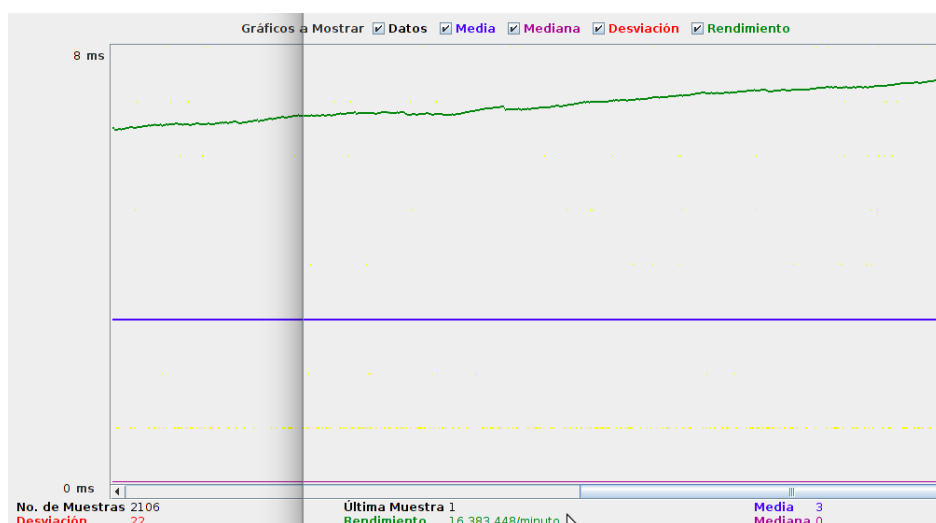


Figura 17: Consultando resultados en jmeter I

verse, la máquina virtual tiene un rendimiento mucho más bajo por minuto que la web de Jmeter.

## 5. Programe un benchmark usando el lenguaje que desee.

**Nota:** para programar la parte del código relativa a la conexión con el USB y la medida de tiempos en OS X, en el que sólo pueden utilizarse algunas librerías, se ha consultado el benchmark disponible en Github que aparece en la siguiente referencia [4].

### 5.1. Objetivo del benchmark

El objetivo del benchmark que se ha programado es medir el tiempo que tarda la máquina en copiar un fichero, que contiene una serie de caracteres generados aleatoriamente, en un USB y en un disco duro, para poder comparar así esos tiempos entre sí y entre distintas máquinas.

### 5.2. Métricas

Las variables medidas durante la ejecución del benchmark han sido:



Figura 18: Consultando resultados en `jmeter II`

- **Elapsed time:** tiempo transcurrido durante la ejecución de la tarea. En este caso, se muestra la media aritmética de los tiempos en 20 ejecuciones, para que el resultado sea lo más objetivo posible. Para entender el código es importante tener en cuenta que `tv_sec` mide el tiempo en segundos y `tv_usec` en microsegundos ( $\mu s$ ), de modo que esta variable se ha expresado en  $\mu s$  para evitar pérdida de precisión.
- **Total bytes written:** número total de bytes copiados en cada prueba. Como cada variable del tipo `char` tiene el tamaño de 1 byte, se ha calculado simplemente sumando los caracteres escritos.
- **Speed:** velocidad de transmisión, medida en KB/s. Para calcularla, se ha dividido el número de KB por ejecución entre el tiempo de ejecución (`elapsed time`) en segundos. Pueden consultarse en el código las transformaciones realizadas.

### 5.3. Instrucciones para su uso

En primer lugar, tiene que compilarse el fichero con `gcc`, de la siguiente forma:

```
gcc benchmark.c -o <nombre-del-ejecutable>
```

Una vez hecho esto, se habrá generado un ejecutable con el nombre que se le ha indicado. Una vez hecho esto, puede ejecutarse el programa con la siguiente sintaxis:

```
./<nombre-del-ejecutable> <ruta-dispositivo>
```

Si la ruta no existe o el USB no se ha conectado a la máquina, aparecerá un error de violación de segmento durante la ejecución. Es importante no realizar otras tareas durante la ejecución del benchmark, para que los resultados no se vean falseados.

**Nota:** es necesario cambiar en el código fuente aquellos fragmentos en los que ponga `/Volumes/MONTSE/` por la ruta en la que ponga la ruta del dispositivo USB. No se ha utilizado directamente `argv[1]` en algunas partes del código para solucionar algunos errores de ejecución.

### 5.4. Ejemplo de uso analizando los resultados

El código fuente se incluye junto al guión de la práctica. El resultado que aparece por pantalla tras la ejecución del benchmark puede verse en la figura (19).

```
Escritorio — bash — 134x65
Creating file...
File created
Writing in /Volumes/MONTSE
0: 158699 microsec
1: 175727 microsec
2: 157998 microsec
3: 165536 microsec
4: 164523 microsec
5: 157659 microsec
6: 166176 microsec
7: 165619 microsec
8: 3632359 microsec
9: 1392505 microsec
10: 219605 microsec
11: 195645 microsec
12: 186497 microsec
13: 1392206 microsec
14: 191841 microsec
15: 1366804 microsec
16: 240924 microsec
17: 216120 microsec
18: 205961 microsec
19: 894763 microsec
Writing in HD
0: 143495 microsec
1: 143124 microsec
2: 143403 microsec
3: 143001 microsec
4: 142604 microsec
5: 142637 microsec
6: 143106 microsec
7: 142701 microsec
8: 142829 microsec
9: 142918 microsec
10: 145098 microsec
11: 143185 microsec
12: 142901 microsec
13: 142980 microsec
14: 143105 microsec
15: 142910 microsec
16: 142964 microsec
17: 147408 microsec
18: 144905 microsec
19: 143145 microsec
Removing files...

Results

USB:
Mean elapsed time (microsec): 221531902
Total bytes written: 1048577
Speed 4.622 KB/s

HD:
Mean elapsed time (microsec): 143421
Total bytes written: 1048577
Speed 7139.826 KB/s
mesa:Desktop montse$
```

Figura 19: Salida por pantalla tras la ejecución del benchmark

Se ha ejecutado el benchmark en dos ordenadores distintos, a los que llamaremos máquina 1 y máquina 2. Los resultados de dichas ejecuciones pueden consultarse en las tablas 1 y 2, respectivamente.

Algunos comentarios acerca de esta ejecución son los siguientes:

- El total de bytes escritos es siempre el mismo puesto que se crea siempre un archivo del mismo tamaño.
- A la hora de escribir en el dispositivo USB, la ganancia de velocidad de la máquina 1 con respecto

Máquina 1	Elapsed time ( $\mu s$ )	Total bytes written	Speed (KB/s)
USB	223813530	1048577	4.575
HD	142175	1048577	7202.398

Tabla 1: Resultados de la ejecución en la máquina 1

Máquina 2	Elapsed time	Total bytes written	Speed (KB/s)
USB	227892540	1048577	4.493
HD	221289	1048577	4627.437

Tabla 2: Resultados de la ejecución en la máquina 2

a la máquina 2 sería:

$$S_{maquina\_2}(maquina\_1) = \frac{v_1}{v_2} = 1,02$$

Esto es, la máquina 1 es 1.02 veces más rápida que la máquina 2 a la hora de copiar el archivo en el dispositivo USB, esto es, la velocidad es prácticamente la misma. Como es lógico, para calcular la ganancia de velocidad puede usarse tanto **elapsed time** como **speed**.

- Se repite el proceso para calcular la ganancia de velocidad a la hora de copiar el archivo en el disco duro. En este caso, la ganancia en velocidad de la máquina 1 con respecto a la máquina 2 es de 1.56.

Aparentemente, con la métrica utilizada, la máquina 1 proporciona mejores prestaciones. Se comparan ahora los tiempos en una misma máquina. Utilizando de nuevo la ganancia, tenemos:

$$S_{USB}(HD) = \frac{v_{HD}}{v_{USB}} = 1574,29$$

Es decir, la máquina 1 copia los archivos en el disco duro 1574.29 veces más rápido que en el dispositivo USB.

En conclusión, la máquina 1 proporciona mejores prestaciones que la máquina 2, y la rapidez con la que se copia un fichero es mucho mayor si se hace en el disco duro que en un dispositivo de almacenamiento externo.

## 6. Cuestiones opcionales

### 6.1. Cuestión opcional 1: Seleccione, instale y ejecute uno, comente los resultados.

En primer lugar, se nota la diferencia entre una suite y un benchmark, para evitar confusión a la hora de hacer la práctica: un benchmark es simplemente un test individual que puede instalarse y ejecutarse de forma individual, mientras las suites son agrupaciones de estos tests [1].

Para instalar un test o benchmark, se utiliza la opción `install`, y para ejecutarlo, `run`. Es importante hacer todas estas operaciones como superusuario [5].

```
sudo phoronix-test-suite install <nombre-test>
```

```
sudo phoronix-test-suite run <nombre-test>
```

donde `<nombre-test>` es el nombre del benchmark a instalar o ejecutar. Se ha probado en primer lugar a instalar algunos test como `pts/apache` o `pts/aio-stress`, pero no había espacio en la memoria para instalar ninguno de ellos (ver figura (20)).

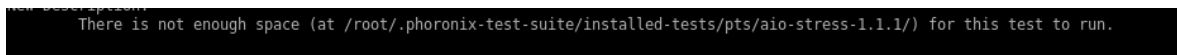


Figura 20: Imposibilidad de instalar un benchmark por falta de espacio

Uno de los test que finalmente se pudo instalar fue `pts/openssl`, que puede utilizarse para medir el rendimiento de cifrado del equipo. El procesador utilizado es un `Intel Core i5`. Como se puede ver en la figura (21), se efectúan 3 pruebas, dando como resultado que la máquina puede realizar un promedio de 82.47 firmas por segundo.

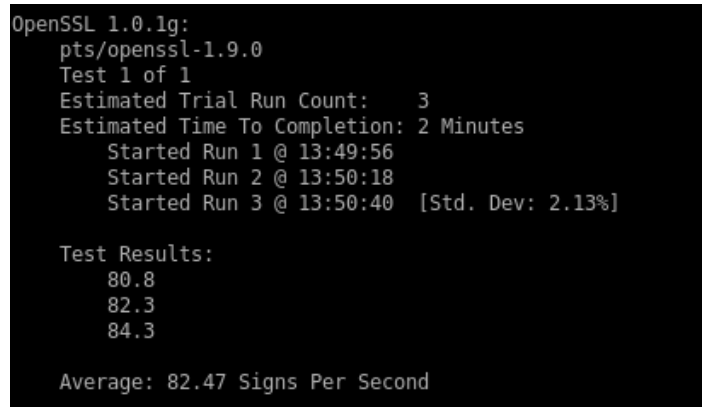


Figura 21: Testeando `openssl` en una máquina virtual

Pero, ¿cómo de bueno es éste resultado? Para comprobarlo, se consultan los resultados en la página <https://openbenchmarking.org>. En esta página se muestran los resultados de este test en media, basados en miles de datos, en función del tipo de procesador. Por ejemplo, en la captura (22) puede verse cómo el resultado está un poco por debajo de los resultados medios para el tipo de procesador, pero también que la versión no es la misma (`OpenSSL 1.0.1g`), por lo que esta comparación no es fiable. Hay que tener en cuenta que estos resultados se han tomado en una máquina virtual.

Se consigue instalar `phoronix-test-suite` en el anfitrión, que tiene el sistema operativo `OS X`, utilizando el gestor de paquetes `brew`, y se ejecuta el test para ver los resultados en esta ocasión. Es



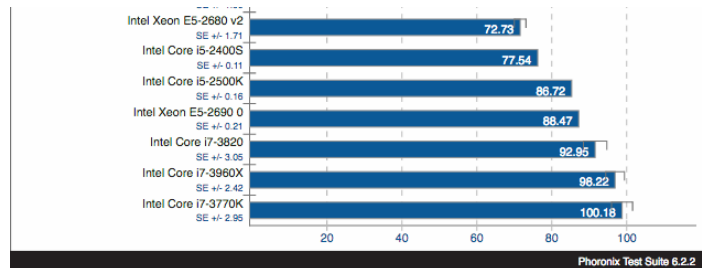


Figura 22: Mejores resultados en la ejecución del benchmark en función del procesador

importante no ejecutar nada más en la máquina en ese momento porque si no los resultados pueden verse falseados (ver la primera figura en (23)). Como puede verse, estos resultados son incluso peores que en la máquina virtual.

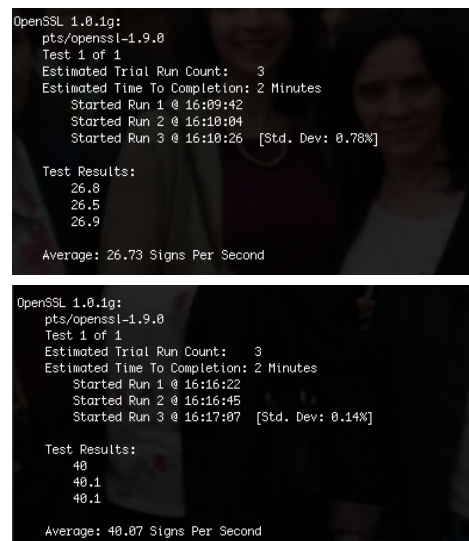


Figura 23: Ejecución del test en OS X

## 6.2. Cuestión opcional 2: ¿Qué es Scala? Instale Gatling y pruebe los escenarios por defecto.

Scala es un acrónimo de *Scalable Language*. Se trata de un lenguaje multiparadigma, escalable y funcional. Es el lenguaje de programación de empresas como Twitter o LinkedIn. Se ejecuta sobre una máquina virtual Java, permitiendo la integración de librerías y herramientas [6]. Una de las herramientas basadas en scala es Gatling. Se puede instalar en Ubuntu a través de la línea de comandos:

```
sudo apt-get install gatling
```

Una vez se haya instalado, iniciamos el servicio. Si es la primera vez que se inicia, es posible que aparezca un error que indique que es necesario modificar el archivo `/etc/default/gatling`. Se modifica dicho archivo como aparece en la captura y se inicia el servicio (24).

Otra opción es descargar Gatling siguiendo el manual desde la página oficial [gatling.io/#/resources/download](http://gatling.io/#/resources/download), y ejecutar el script que se encuentra en la carpeta descargada: `bin/gatling.sh`. Una vez hecho esto, nos aparecerá un menú con algunos de los escenarios por defecto (25) [7].

```
# Defaults for gatling initscript
# sourced by /etc/init.d/gatling
# installed at /etc/default/gatling by the maintainer scripts

#
# This is a POSIX shell fragment
#
#
# uncomment the next line run gatling automatically at startup
#
START_DAEMON="YES"

#
# Gatling options
#
DAEMON_OPTS="-e -v -D -S -F -U -u nobody -c /var/www"

#
# Choose daemon in use. Default is TLS-enabled tlsgatling
# Alternatives: gatling, ptlsgatling (PolarSSL version)
#
#DAEMON="gatling"

minim@minim:~$ sudo service gatling start
* Not starting Gatling (edit /etc/default/gatling to enable)
minim@minim:~$ sudo nano /etc/default/gatling
minim@minim:~$ sudo service gatling start
* Starting Gatling tlsgatling
```

Figura 24: Iniciando el servicio Gatling

```
minim@minim:~/gatling-charts-highcharts-bundle-2.1.7/bin$ ls
gatling.bat  gatling.sh  recorder.bat  recorder.sh
minim@minim:~/gatling-charts-highcharts-bundle-2.1.7/bin$ ./gatling.sh
GATLING HOME is set to /home/minim/gatling-charts-highcharts-bundle-2.1.7
20:47:37.529 [ERROR] i.g.c.ZincCompiler$ - /home/minim/gatling-charts-highcharts-bundle-2.1.7/
20:47:37.691 [ERROR] i.g.c.ZincCompiler$ - one error found
Choose a simulation number:
[0] computerdatabase.BasicSimulation
[1] computerdatabase.advanced.AdvancedSimulationStep01
[2] computerdatabase.advanced.AdvancedSimulationStep02
[3] computerdatabase.advanced.AdvancedSimulationStep03
[4] computerdatabase.advanced.AdvancedSimulationStep04
[5] computerdatabase.advanced.AdvancedSimulationStep05
```

Figura 25: Escenarios por defecto en Gatling

Se ejecuta uno de ellos, por ejemplo, el 0. Se pedirá que se introduzca algunos datos como ID del escenario, que será **basicsimulation** por defecto y descripción de la ejecución. Una vez hecho esto, comenzará la ejecución (26).

Pueden consultarse los resultados en un informe generado en el archivo html que se indicará una vez finalizada la ejecución (27). En este archivo podrá verse en primer lugar, el ID especificado (en este caso, *ise*) junto con la descripción (*cuestion opc 2*), la duración de la prueba y la fecha en la que se realizó.

Los primeros resultados que pueden verse es una gráfica con la información global. Puede verse que todas las peticiones se han resuelto en menos de 800 ms, y que ha respondido a todas exitosamente. Si se avanza un poco en la página, pueden consultarse algunas estadísticas más detalladas. A continuación se analizarán tan sólo algunas de ellas.

Inmediatamente después de esta información global, puede encontrarse una tabla con los detalles de cada petición: ejecución y tiempo de respuesta (28), incluyendo información estadística. Por ejemplo, puede verse cómo la mayor parte de las respuestas se dan en los percentiles 95 y 99. Esta información se recogerá en gráficas más adelante.

```

Select simulation id (default is 'basicsimulation'). Accepted characters are a-z, A-Z, 0-9, - and _
ise
Select run description (optional)
Question opc 2
Simulation computerdatabase.BasicSimulation started...

=====
2016-06-11 11:07:05                                0s elapsed
--- Scenario Name -----
[-----] 0%
    waiting: 1    / active: 0    / done:0
--- Requests -----
> Global (OK=0 KO=0 )
=====

2016-06-11 11:07:10                                5s elapsed
--- Scenario Name -----
[-----] 0%
    waiting: 0    / active: 1    / done:0
--- Requests -----
> Global (OK=2 KO=0 )
> request 1 (OK=1 KO=0 )

```

Figura 26: Probando el escenario 0 por defecto en Gatling

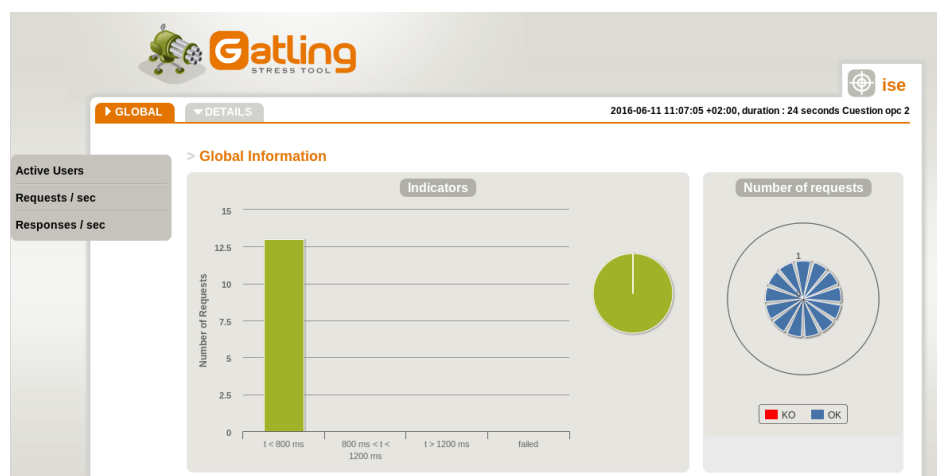


Figura 27: Resultados de la ejecución con Gatling I

Algunas de las gráficas que pueden consultarse se han incluido en la figura (29). La primera de ellas muestra el número de usuarios activos durante la ejecución de la prueba, en este caso es claro que sólo hay uno. En la segunda puede verse el número de peticiones que se han recogido por segundo.

### 6.3. Cuestión opcional 3: Lea el artículo y elabore un breve resumen.

Este artículo busca comparar Gatling y JMeter. Para el experimento se necesita un entorno en el que se puedan manejar los tipos de concurrencia y volúmenes que se lanzarán, así como generar contenido dinámico y estático, entre otras cosas. Para esta tarea se escogerá **nginx**. Se modificará el kernel del sistema operativo y los ajustes de TCP para asegurar que no hay cuellos de botella.

La prueba consistirá en ejecutar una serie de transacciones. Se ejecutará para 10000 usuarios, con un volumen de 30000 peticiones/minuto durante 20 minutos, en Gatling-1.5.3, JMeter-2.9 y JMeter-2.10. Se concluye que no hay demasiada diferencia entre los resultados.

Se pueden hacer algunas observaciones importantes como que Gatling no guarda los tiempos de respuesta en bytes, por lo que los datos del throughput pueden no ser exactos; JMeter es más pesado en la máquina virtual java (JVM) que Gatling; JMeter utiliza más la CPU y la memoria.

STATISTICS <span>Expand all groups   Collapse all groups</span>													
Requests ^	Executions				Req/s ^	Response Time (ms)							
	Total ^	OK ^	KO ^	% KO ^		Min ^	50th pct ^	75th pct ^	95th pct ^	99th pct ^	Max ^	Mean ^	Std Dev ^
Global Information	13	13	0	0%	0.523	145	202	250	272	286	290	206	50
request_...direct1	1	1	0	0%	0.04	158	158	158	158	158	158	158	0
request_1	1	1	0	0%	0.04	202	202	202	202	202	202	202	0
request_2	1	1	0	0%	0.04	290	290	290	290	290	290	290	0
request_3	1	1	0	0%	0.04	244	244	244	244	244	244	244	0
request_...direct1	1	1	0	0%	0.04	154	154	154	154	154	154	154	0
request_4	1	1	0	0%	0.04	250	250	250	250	250	250	250	0
request_5	1	1	0	0%	0.04	174	174	174	174	174	174	174	0
request_6	1	1	0	0%	0.04	154	154	154	154	154	154	154	0
request_7	1	1	0	0%	0.04	242	242	242	242	242	242	242	0
request_8	1	1	0	0%	0.04	258	258	258	258	258	258	258	0
request_9	1	1	0	0%	0.04	155	155	155	155	155	155	155	0
request_10	1	1	0	0%	0.04	145	145	145	145	145	145	145	0
request_...direct1	1	1	0	0%	0.04	261	261	261	261	261	261	261	0

Figura 28: Resultados de la ejecución con Gatling II

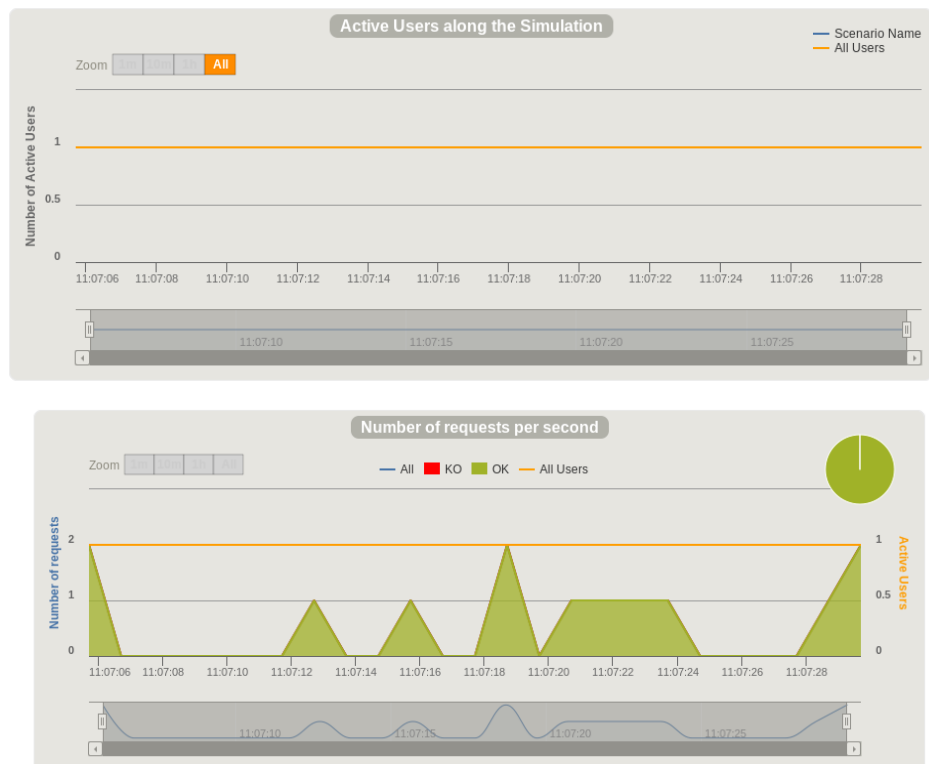


Figura 29: Resultados de la ejecución con Gatling III

El artículo concluye que la elección entre JMeter y Gatling es subjetiva.

## Referencias

- [1] <https://wiki.ubuntu.com/PhoronixTestSuite>
- [2] <https://lpic1.wordpress.com/2012/02/22/comandos-para-resumir-ficheros-cut-y-wc/>
- [3] <https://linuxconfig.org/wc-1-manual-page>
- [4] [https://github.com/kehribar/usbSerial\\_benchmark](https://github.com/kehribar/usbSerial_benchmark)
- [5] <http://www.phoronix-test-suite.com/documentation/phoronix-test-suite.pdf>
- [6] <http://www.scala-lang.org/what-is-scala.html>
- [7] [http://gatling.io/docs/1.5.6/user\\_documentation/tutorial/getting\\_started.html](http://gatling.io/docs/1.5.6/user_documentation/tutorial/getting_started.html)