

INGENIERÍA DE SERVIDORES (2016-2017)
GRADO EN INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE GRANADA

Memoria Práctica 4

Guillermo Montes Martos

23 de diciembre de 2016

Índice

1. Cuestión 1	4
1.1. Seleccione, instale y ejecute uno de los benchmarks de phoronix, comente los resultados. Atención: no es lo mismo un benchmark que una suite, instale un benchmark.	4
2. Cuestión 2	7
2.1. 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) ¿cuántas “tarefas” crea ab en el cliente?	7
3. Cuestión 3	8
3.1. Ejecute ab contra a las tres máquinas virtuales (desde el SO anfitrión a las máquina virtuales de la red local) una a una (arrancadas por separado).¿Cuál es la que proporciona mejores resultados? Muestre y coméntelos. (Use como máquina de referencia Ubuntu Server para la comparativa). . .	8
4. Cuestión 4	12
4.1. Instale Jmeter y siga el tutorial en http://jmeter.apache.org/usermanual/build-web-test-plan.html realizando capturas de pantalla y comentándolas. En vez de usar la web de jmeter, haga el experimento usando sus máquinas virtuales ¿coincide con los resultados de ab?	12
5. Cuestión 5	18
5.1. 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; 4) Ejemplo de uso analizando los resultados.	18

Índice de figuras

1.1. Instalando Phoronix Test Suite.	4
1.2. Lista de test en Phoronix Test Suite.	5
1.3. Instalando un benchmark con Phoronix Test Suite.	5
1.4. Configurando un benchmark en Phoronix Test Suite.	6
1.5. Ejecutando un benchmark en Phoronix Test Suite.	6
1.6. Gráfica de resultados del benchmark.	7
2.1. Probando Apache Benchmark.	8
3.1. Servidores arrancados y mostrando su IP.	9
3.2. Página por defecto de Apache2 para Ubuntu.	9
3.3. Resultado de la ejecución de ab en Ubuntu Server.	10
3.4. Resultado de la ejecución de ab en CentOS.	10
3.5. Resultado de la ejecución de ab en Windows Server.	11

4.1.	Versión de java instalada para la prueba.	12
4.2.	Añadiendo nuevo grupo de hilos.	13
4.3.	Configurando el nuevo grupo de hilos.	13
4.4.	Añadiendo petición HTTP.	14
4.5.	Configurando petición HTTP.	14
4.6.	Añadiendo nuevo grupo de hilos.	15
4.7.	Añadiendo una petición para una ruta a visitar.	15
4.8.	Configurando ruta principal.	16
4.9.	Configurando ruta info.php.	16
4.10.	Añadiendo un <i>Listener</i>	17
4.11.	Resultados del test con jmeter.	17
5.1.	Resultados del benchmark modificado.	19

1. Cuestión 1

1.1. Seleccione, instale y ejecute uno de los benchmarks de phoronix, comente los resultados. Atención: no es lo mismo un benchmark que una suite, instale un benchmark.

Para la realización del ejercicio, seguiremos los pasos explicados por Ubuntu Wiki en un tutorial sobre esta aplicación [1]. En este, explica que lo primero que tendremos que hacer será instalar el programa principal, el cual podremos encontrar en los repositorios oficiales de Ubuntu (*sudo apt-get install phoronix-test-suite*).

```
[16/12/16 gmm@ubuntu-server:~] $ sudo apt-get install phoronix-test-suite
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes NUEVOS:
  phoronix-test-suite
0 actualizados, 1 nuevos se instalarán, 0 para eliminar y 70 no actualizados.
Se necesita descargar 0 B/390 kB de archivos.
Se utilizarán 2.697 kB de espacio de disco adicional después de esta operación.
Seleccionando el paquete phoronix-test-suite previamente no seleccionado.
(Leyendo la base de datos ... 219363 ficheros o directorios instalados actualmente.)
Preparando para desempaquetar .../phoronix-test-suite_5.2.1-1ubuntu2_all.deb ...
Desempaquetando phoronix-test-suite (5.2.1-1ubuntu2) ...
Procesando disparadores para mime-support (3.59ubuntu1) ...
Procesando disparadores para hicolor-icon-theme (0.15-0ubuntu1) ...
Procesando disparadores para man-db (2.7.5-1) ...
Configurando phoronix-test-suite (5.2.1-1ubuntu2) ...
[16/12/16 gmm@ubuntu-server:~] $ _
```

Figura 1.1: Instalando Phoronix Test Suite.

Una vez instalado, tendremos que modificar uno de los archivos de la aplicación, concretamente un script para instalar dependencias fallidas, ya que este viene de forma errónea por defecto. Así, abrimos el fichero `/usr/share/phoronix-test-suite/pts-core/external-test-dependencies/scripts/install-ubuntu-packages.sh` con vim y modificamos el segundo bloque *if*, el cual debe de quedar de la siguiente forma:

```
if [ -x /usr/bin/aptitude ]; then
    # aptitude is nice since it doesn't fail if a non-existent package is hit
    # See: http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=503215
    $ROOT -- aptitude -y install $*
else
    $ROOT -- apt-get -y --ignore-missing install $*
fi
```

Hecho esto, ya podemos ejecutar phoronix (la primera vez se nos pedirá que respondamos unas preguntas). Lo primero que tendremos que hacer será listar todos los test disponibles, lo cual podremos hacer con el comando *phoronix-test-suite list-available-tests*.

Phoronix Test Suite v5.2.1		
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/blender	- Blender	System
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-webkitfltk	- Timed WebKitFLTK Compilation	Processor

Figura 1.2: Lista de test en Phoronix Test Suite.

Seleccionamos uno de los test de entre la extensa lista, el cual se trata en este caso de *7-Zip Compression (compress-7zip)*. Aunque podríamos instalarlo y ejecutarlo por separado, se elige realizar ambos procesos de una tacada mediante el comando *phoronix-test-suite benchmark compress-7zip*.

```
[17/12/16 gmm@ubuntuserver:~] $ phoronix-test-suite benchmark compress-7zip
Phoronix Test Suite v5.2.1

To Install: pts/compress-7zip-1.6.2

Determining File Requirements .....
Searching Download Caches .....

1 Test To Install
  1 File To Download [3.66MB]
  10MB Of Disk Space Is Needed

pts/compress-7zip-1.6.2:
  Test Installation 1 of 1
  1 File Needed [3.66 MB]
  Downloading: p7zip_9.20.1_src_all.tar.bz2 [3.66MB]
  Downloading .....
  Installation Size: 10 MB
  Installing Test @ 01:26:04
```

Figura 1.3: Instalando un benchmark con Phoronix Test Suite.

Tras la instalación, nos preguntará, entre otras cosas, si deseamos guardar los resultados del benchmark, una ubicación para los ficheros de los resultados, etc.

```

System Information

Hardware:
Processor: Intel Core i7-6700HQ @ 2.59GHz (1 Core), Motherboard: Oracle VirtualBox v1.2, Chipset: In
tel 440FX- 82441FX PMC, Memory: 1024MB, Disk: 2 x 21GB VBox HDD, Graphics: InnoTek VirtualBox, Audio
: Intel 82801AA AC 97 Audio, Network: Intel 82540EM Gigabit

Software:
OS: Ubuntu 16.04, Kernel: 4.4.0-53-generic (x86_64), Compiler: GCC 5.4.0 20160609, File-System: ext4
, Screen Resolution: 800x600, System Layer: Oracle VirtualBox

Would you like to save these test results (Y/n): Y
Enter a name to save these results under: zip
Enter a unique name to describe this test run / configuration: ISE-zip-test

If you wish, enter a new description below to better describe this result set / system configuration
under test.
Press ENTER to proceed without changes.

Current Description: Oracle VirtualBox testing on Ubuntu 16.04 via the Phoronix Test Suite.

New Description:

```

Figura 1.4: Configurando un benchmark en Phoronix Test Suite.

Hecho esto, comienza la ejecución del test. Una vez finalizado, se muestran unos resultados de forma resumida y se nos pregunta si queremos subir los resultados a openbenchmarking.org. Si respondemos de forma positiva, la aplicación nos dará un enlace con el cual podremos consultar los resultados de forma online [2].

```

7-Zip Compression 9.20.1:
pts/compress-7zip-1.6.2
Test 1 of 1
Estimated Trial Run Count: 3
Estimated Time To Completion: 4 Minutes
Started Run 1 @ 01:27:12
Started Run 2 @ 01:27:45
Started Run 3 @ 01:28:17 [Std. Dev: 1.66%]

Test Results:
3403
3505
3410

Average: 3439 MIPS

Would you like to upload the results to OpenBenchmarking.org (Y/n): y
Would you like to attach the system logs (lspci, dmesg, lsusb, etc) to the test result (Y/n): y

Results Uploaded To: http://openbenchmarking.org/result/1612175-KH-ZIP15743918

[17/12/16 gmm@ubuntuserver:~] $

```

Figura 1.5: Ejecutando un benchmark en Phoronix Test Suite.

En los resultados obtenidos, vemos como el test se ha realizado en tres etapas, de cada cual guarda estadísticas de forma separada, aunque luego las reúne. El resultado nos arroja una velocidad de compresión de 3439 MIPS (Millones de Instrucciones Por Segundo). Este dato supone la media de los resultados obtenidos en cada etapa, los cuales varían hasta obtener una desviación típica de 1.66 % y un error estándar de 32.90. Esto quiere decir que la medida puede considerarse válida, pues los resultados de cada etapa

no varían mucho.

7-Zip Compression

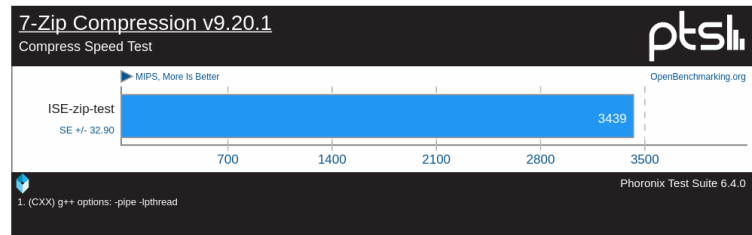


Figura 1.6: Gráfica de resultados del benchmark.

Por último, comparando este resultado con otros obtenidos por otros usuarios del benchmark, los cuales podemos consultar en su página web [3], podemos observar como el resultado obtenido no es muy bueno, lo cual puede explicarse por las bajas características asignadas a la máquina virtual con la cual se ha realizado el test (1 core, 1GB RAM).

2. Cuestión 2

2.1. De los parámetros que le podemos pasar al comando `ab` ¿Qué significa `-c 5` ? ¿y `-n 100`? Monitoree la ejecución de `ab` contra alguna máquina (cualquiera) ¿cuántas “tarefas” crea `ab` en el cliente?

Para consultar esta información, acudimos al manual online de dicho comando [4], donde encontramos ambas opciones perfectamente explicadas. La primera, `-c 5`, hace referencia al número de peticiones a realizar en el mismo momento (5 en este caso), es decir, la concurrencia. Por otro lado, `-n 100` especifica el número total de peticiones a realizar al servidor web por el benchmark (100 peticiones).

Se ejecuta `ab` para monitorizar el tiempo de respuesta de los servidores de la `ugr` [5] con unos parámetros grandes (20000 peticiones con concurrencia de 25) para comprobar cuantos procesos crea en el cliente. Para ello, mostramos todos los procesos del sistema y filtramos únicamente los que se llaman `ab`. El resultado obtenido es que solo se ejecuta un proceso, independientemente de la concurrencia aplicada a la ejecución del benchmark.

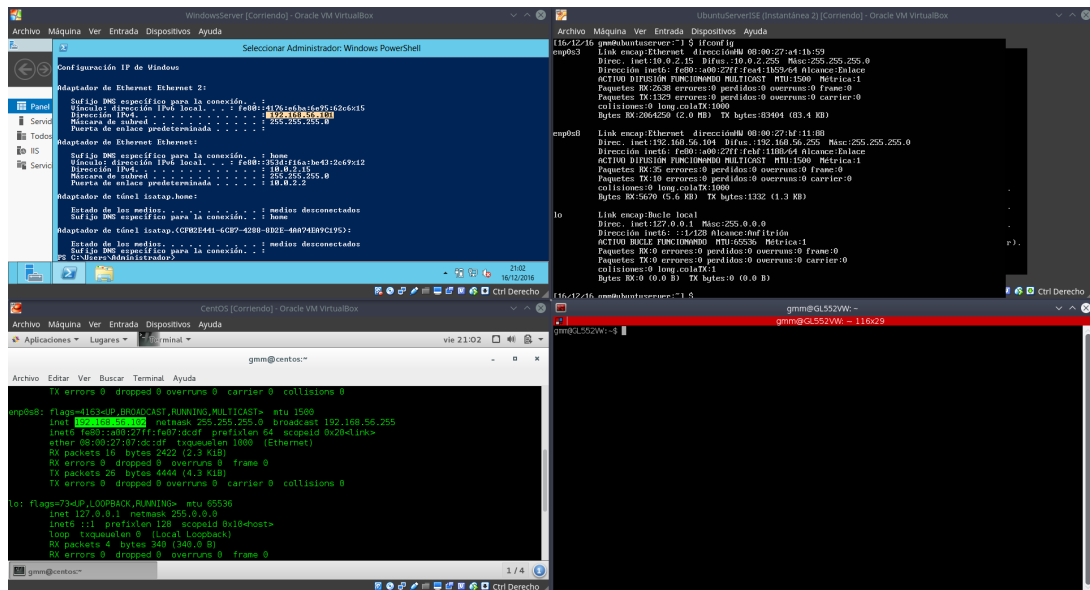


Figura 3.1: Servidores arrancados y mostrando su IP.

Para que el resultado de Apache Benchmark tenga sentido, es necesario establecer la misma página en los tres servidores. En este caso, se ha optado por fijar la página por defecto de Apache2 en Ubuntu para las tres máquinas [6].

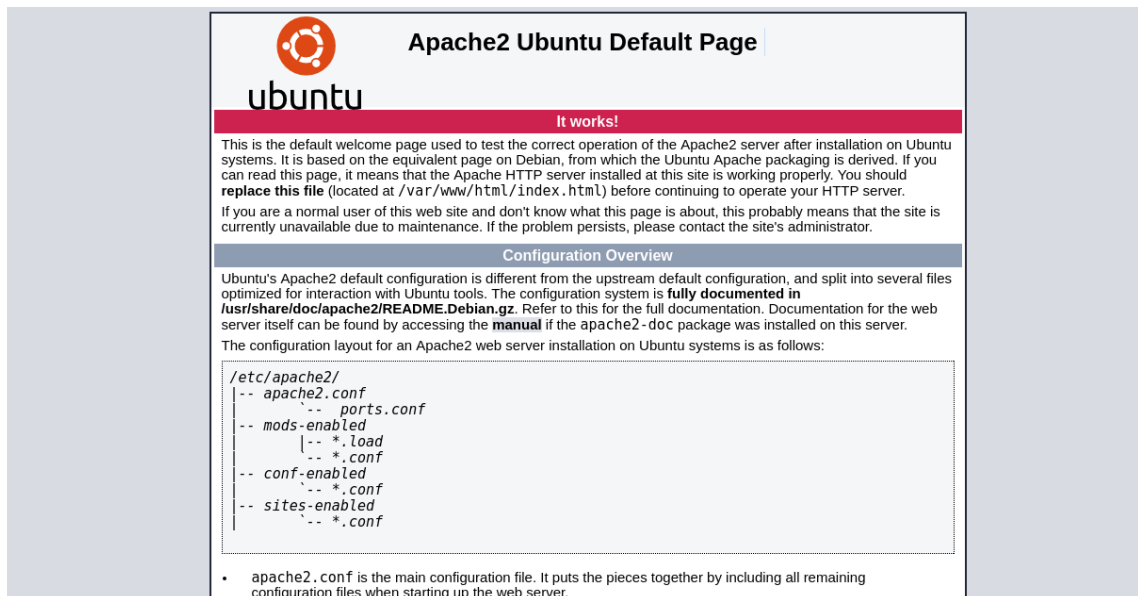


Figura 3.2: Página por defecto de Apache2 para Ubuntu.

Una vez todos los preparativos estén satisfechos, es la hora de ejecutar el benchmark,

realizando un total de 100000 peticiones y estableciendo una concurrencia de 100. Estos parámetros serán los mismos para la ejecución de *ab* en las tres máquinas. Se muestra el resultado obtenido en las siguientes capturas:

```

* [Ubuntu Server 16.04 130x40]
Completed 100000 requests
Finished 100000 requests

Server Software:      Apache/2.4.18
Server Hostname:      192.168.56.104
Server Port:          80

Document Path:        /
Document Length:       11321 bytes

Concurrency Level:     100
Time taken for tests:  31.411 seconds
Complete requests:     100000
Failed requests:        0
Total transferred:     1159500000 bytes
HTML transferred:      1132100000 bytes
Requests per second:   3183.62 [#/sec] (mean)
Time per request:      31.411 [ms] (mean)
Time per request:      0.314 [ms] (mean, across all concurrent requests)
Transfer rate:         36048.86 [Kbytes/sec] received

Connection Times (ms)
  min  mean[+/-sd] median  max
Connect:    0    0  5.5      0   998
Processing:  0   31 20.0     25  613
Waiting:    0   28 12.2     24  577
Total:       0   31 20.7     25 1019

Percentage of the requests served within a certain time (ms)
 50%    25
 66%    33
 75%    37
 80%    39
 90%    49
 95%    60
 98%    79
 99%   101
100%  1019 (longest request)
gmm@GL552VW:~$

```

Figura 3.3: Resultado de la ejecución de *ab* en Ubuntu Server.

```

* [CentOS 7 130x40]
Completed 100000 requests
Finished 100000 requests

Server Software:      Apache/2.4.6
Server Hostname:      192.168.56.102
Server Port:          80

Document Path:        /
Document Length:       11321 bytes

Concurrency Level:     100
Time taken for tests:  23.138 seconds
Complete requests:     100000
Failed requests:        0
Total transferred:     1159700000 bytes
HTML transferred:      1132100000 bytes
Requests per second:   4321.99 [#/sec] (mean)
Time per request:      23.138 [ms] (mean)
Time per request:      0.231 [ms] (mean, across all concurrent requests)
Transfer rate:         48947.36 [Kbytes/sec] received

Connection Times (ms)
  min  mean[+/-sd] median  max
Connect:    0    0  1.3      0    36
Processing:  0   23 36.2     19 1476
Waiting:    0   20 18.8     18 1106
Total:       1   23 36.2     19 1487

Percentage of the requests served within a certain time (ms)
 50%    19
 66%    21
 75%    23
 80%    24
 90%    28
 95%    32
 98%    48
 99%   126
100%  1487 (longest request)
gmm@GL552VW:~$

```

Figura 3.4: Resultado de la ejecución de *ab* en CentOS.

```
Windows Server 12 R2" 130x40
Completed 100000 requests
Finished 100000 requests

Server Software:      Microsoft-IIS/8.0
Server Hostname:      192.168.56.101
Server Port:          80

Document Path:        /
Document Length:       11691 bytes

Concurrency Level:     100
Time taken for tests:   16.538 seconds
Complete requests:      100000
Failed requests:         0
Total transferred:      1193600000 bytes
HTML transferred:       1169100000 bytes
Requests per second:    6046.82 [#/sec] (mean)
Time per request:       16.538 [ms] (mean)
Time per request:       0.165 [ms] (mean, across all concurrent requests)
Transfer rate:          70483.21 [Kbytes/sec] received

Connection Times (ms)
  min   mean[+/-sd] median   max
Connect:  0    1   1.3      0    12
Processing: 0   16   9.9     14   106
Waiting:  0   15   9.9     13   106
Total:     1   17   9.9     15   107

Percentage of the requests served within a certain time (ms)
 50%    15
 66%    17
 75%    19
 80%    21
 90%    28
 95%    39
 98%    47
 99%    52
100%   107 (longest request)

gmm@GL552VW: ~$
```

Figura 3.5: Resultado de la ejecución de *ab* en Windows Server.

La primera diferencia que encontramos servidor web, siendo diferentes versiones de Apache2 en las dos máquinas Linux y Microsoft-IIS en Windows. Otra diferencia bastante llamativa es el tamaño del fichero consultado, ya que, aún teniendo el mismo contenido, el tamaño difiere entre las máquinas Linux y Windows. Esto puede ser debido a la diferente codificación de texto usada en ambos sistemas (*Unicode* [7] en Windows y *UTF-8* [8] en Linux).

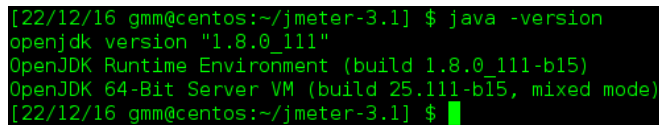
Centrándonos ya en los resultados medidos, vemos como la máquina ganadora del test es Windows, seguida de CentOS, dejando en último lugar a Ubuntu Server. También se observa como difieren el total de bytes transmitidos (*Total transferred*) entre las dos máquinas Linux, aunque ambas usen Apache2. Esto es debido a la diferente versión del servidor web, ya que puede meter distintas cabeceras. Por otro lado, como era de esperar, el total de bytes transmitidos por el servidor Windows es completamente distinto, siendo paradójicamente mayor que el del resto aún habiendo sido más rápida. Es por ello por lo que la velocidad de transferencia (*Transfer rate*) en Windows es bastante mayor que el resto.

Si entramos un poco más en detalle observando el tiempo fraccionado según la operación realizada, vemos como los tiempos de respuesta obtenidos en CentOS para cada petición difieren bastante, ya que el valor de la desviación típica es relativamente alto en comparación con los obtenidos en Ubuntu, mientras que en Windows la diferencia es mínima. Esto nos deja ver unos picos grandes en el tiempo respuesta de CentOS respecto a las otras dos, siendo en Windows bastante chicos. También cabe destacar como el tiempo medio de conexión es mayor en Windows que en Ubuntu.

4. Cuestión 4

- 4.1. Instale Jmeter y siga el tutorial en <http://jmeter.apache.org/usermanual/build-web-test-plan.html> realizando capturas de pantalla y comentándolas. En vez de usar la web de jmeter, haga el experimento usando sus máquinas virtuales ¿coincide con los resultados de ab?

Para la realización de este ejercicio, el primer paso es la instalación de la aplicación, para la cual se sigue el tutorial explicado en su web [9]. Accedemos a la página de descargas y bajamos el fichero zip con los binarios de la última versión [10], la cual en nuestro caso se trata de la 3.1. También tendremos que tener instalada la última versión de java, ya que el benchmark necesita de este intérprete.

A terminal window with a black background and green text. The text shows the command 'java -version' being executed in a directory '~/.jmeter-3.1'. The output displays 'openjdk version "1.8.0_111"', 'OpenJDK Runtime Environment (build 1.8.0_111-b15)', and 'OpenJDK 64-Bit Server VM (build 25.111-b15, mixed mode)'. The prompt returns to the shell.

```
[22/12/16 gmm@centos:~/.jmeter-3.1] $ java -version
openjdk version "1.8.0_111"
OpenJDK Runtime Environment (build 1.8.0_111-b15)
OpenJDK 64-Bit Server VM (build 25.111-b15, mixed mode)
[22/12/16 gmm@centos:~/.jmeter-3.1] $
```

Figura 4.1: Versión de java instalada para la prueba.

Una vez desempaquetamos el fichero zip con los binarios, podemos seguir con el tutorial. Este trata de realizar un test sobre una página web, la cual se será en nuestro caso la albergada en nuestra máquina virtual con Ubuntu Server. El test simulará 5 clientes que realizarán 2 peticiones cada uno. Además, este se repetirá otra vez, de manera que el test se realiza dos veces. Así, en total, tendremos 20 peticiones.

Abrimos la aplicación. El primer paso será añadir un nuevo grupo de hilos, el cual se simulará el grupo de clientes realizando peticiones. Para ello, hacemos click derecho en *Plan de Pruebas* y seleccionamos *Añadir* → *Hilos (usuarios)* → *Grupo de Hilos*.

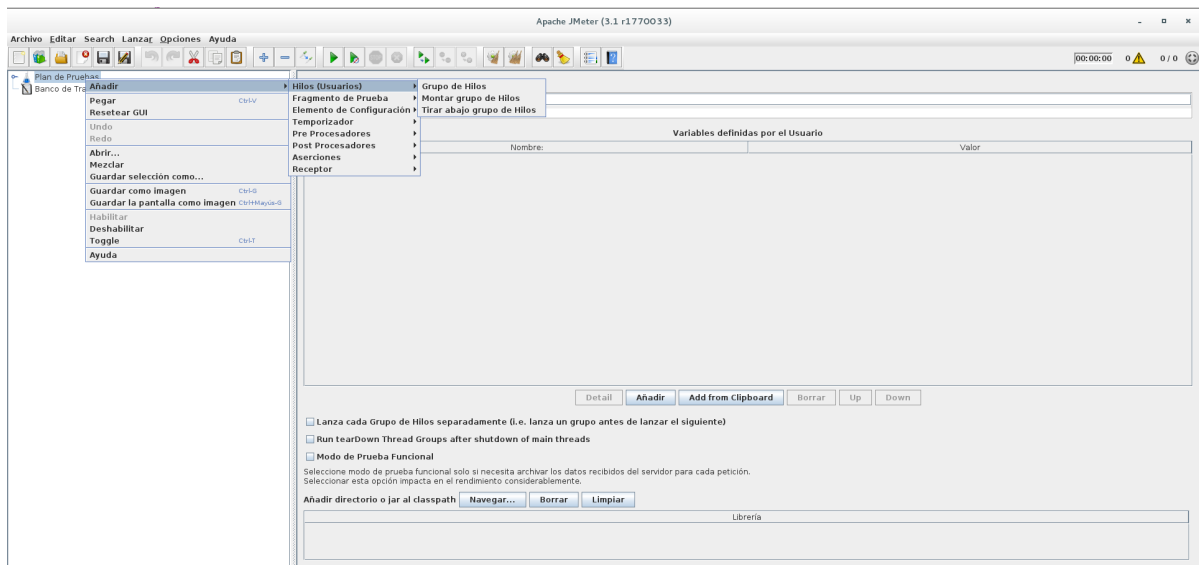


Figura 4.2: Añadiendo nuevo grupo de hilos.

Procedemos a configurarlo, dándole de nombre "Jmeter users", estableciendo el número de hilos en 5 y el número de bucles en 2.

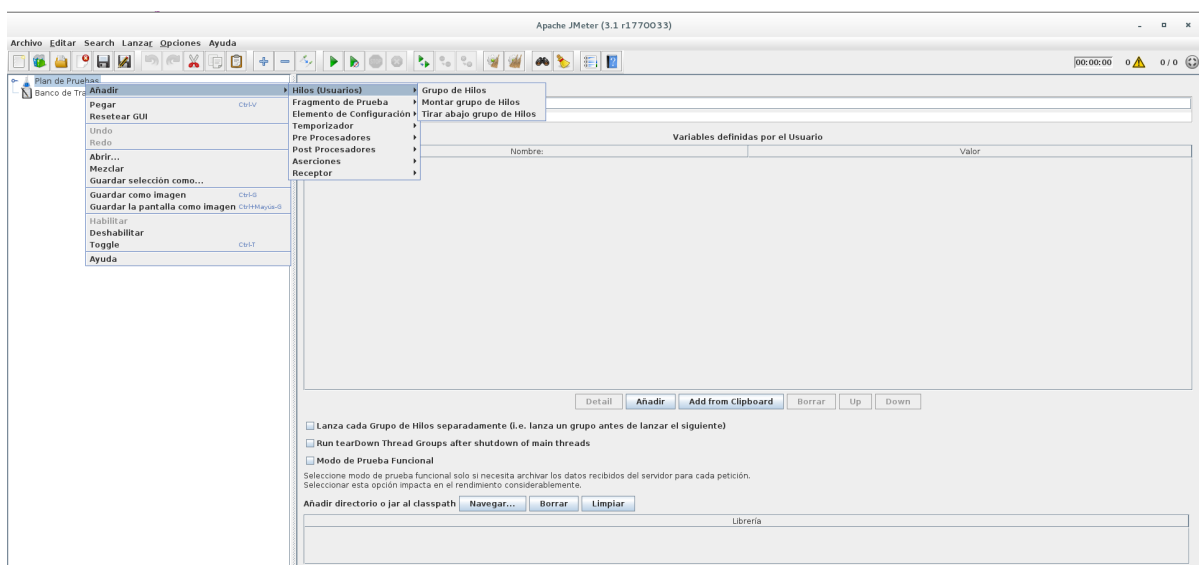


Figura 4.3: Configurando el nuevo grupo de hilos.

Ahora es turno de añadir las peticiones HTTP. Para ello, hacemos click derecho en *Jmeter users* y seleccionamos *Añadir* → *Elemento de configuración* → *Valores por defecto para petición HTTP*.

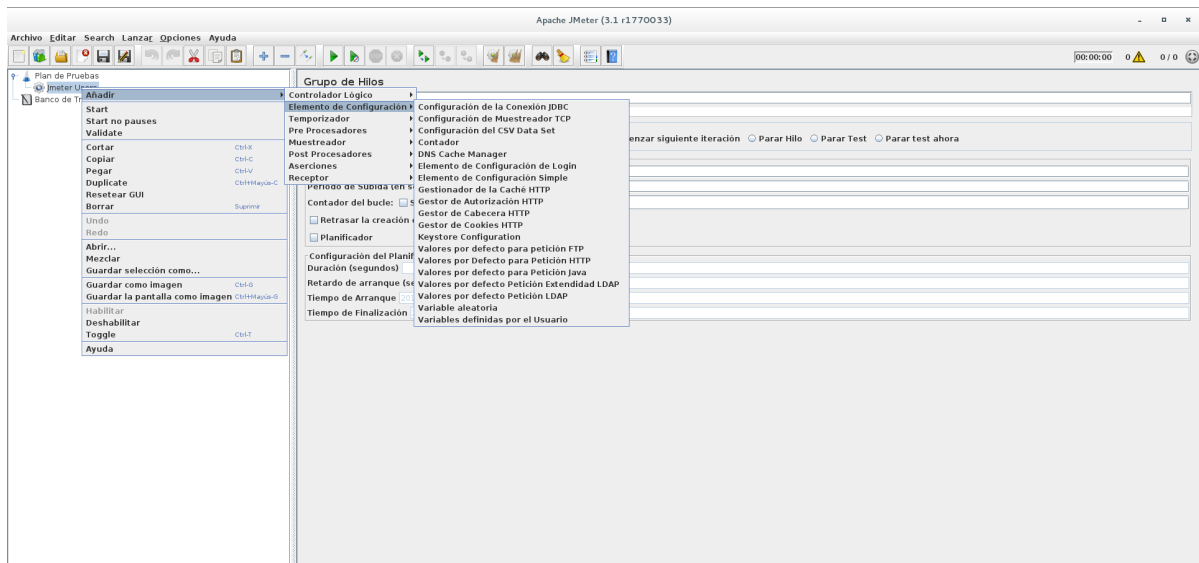


Figura 4.4: Añadiendo petición HTTP.

Dejamos el nombre de esta petición por defecto y añadimos la ip del servidor, que en nuestro caso es Ubuntu Server, la cual encontramos en la ip 192.168.56.104.

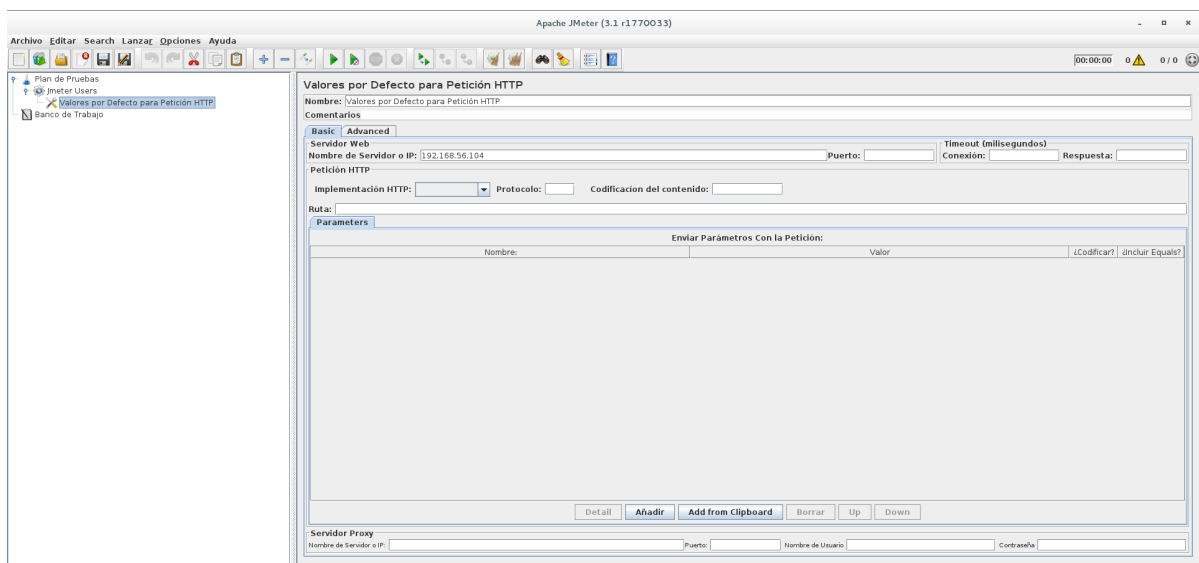


Figura 4.5: Configurando petición HTTP.

En el tutorial, se ofrece la posibilidad de añadir soporte para cookies para el grupo de hilos añadidos. Teniendo en cuenta la página web a la cual se le va a realizar el benchmark, no merece la pena, pero si quisiéramos añadirlo, tendríamos que hacer click derecho en *Jmeter users* y escoger *Añadir* → *Elemento de configuración* → *Gestor de de*

Cookies HTTP.

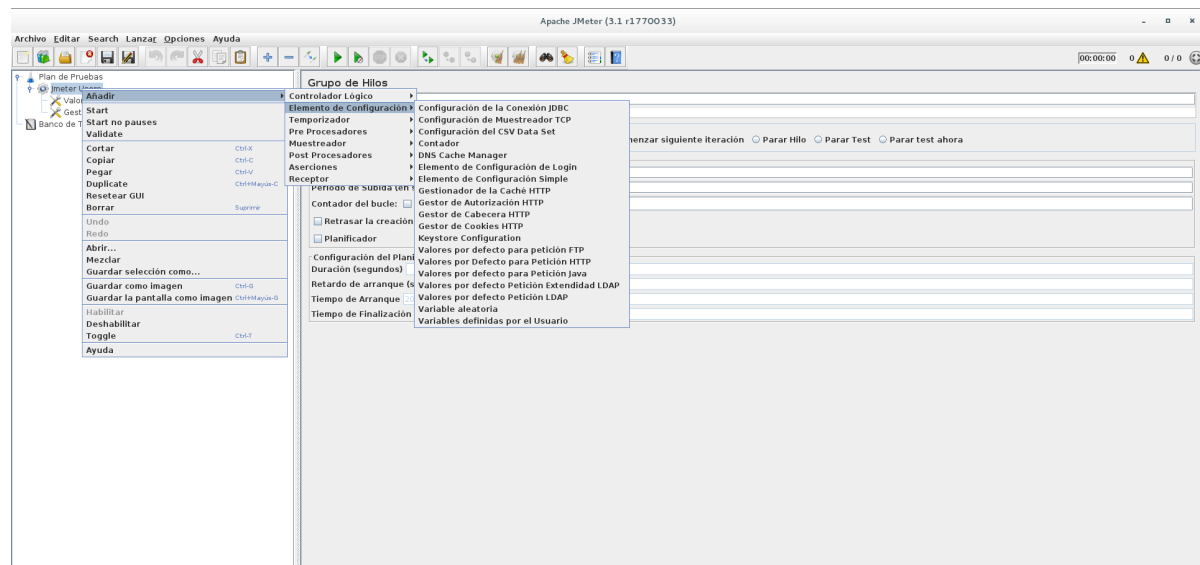


Figura 4.6: Añadiendo nuevo grupo de hilos.

El siguiente paso en el tutorial es añadir las 2 peticiones. Se realizará una a la página principal del sitio web y otra a un script php llamado *info.php*. Para añadirlas, hacemos click derecho en *Jmeter users* y seleccionamos *Añadir* → *Muestreador* → *Petición HTTP*. Debemos de crear dos, una para cada petición, modificando el nombre y la ruta de cada una.

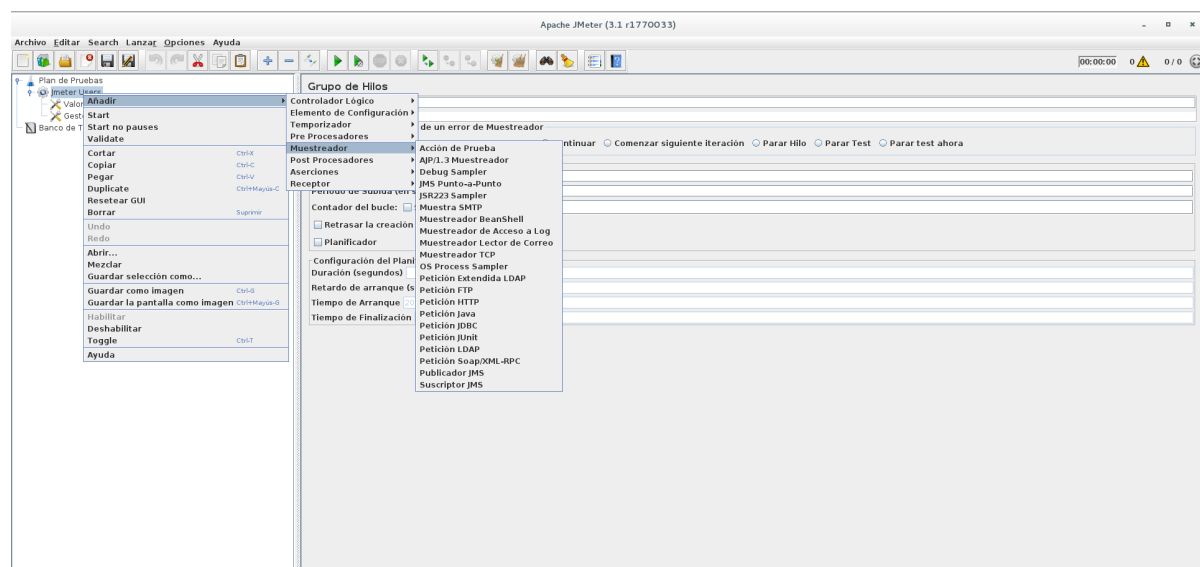


Figura 4.7: Añadiendo una petición para una ruta a visitar.

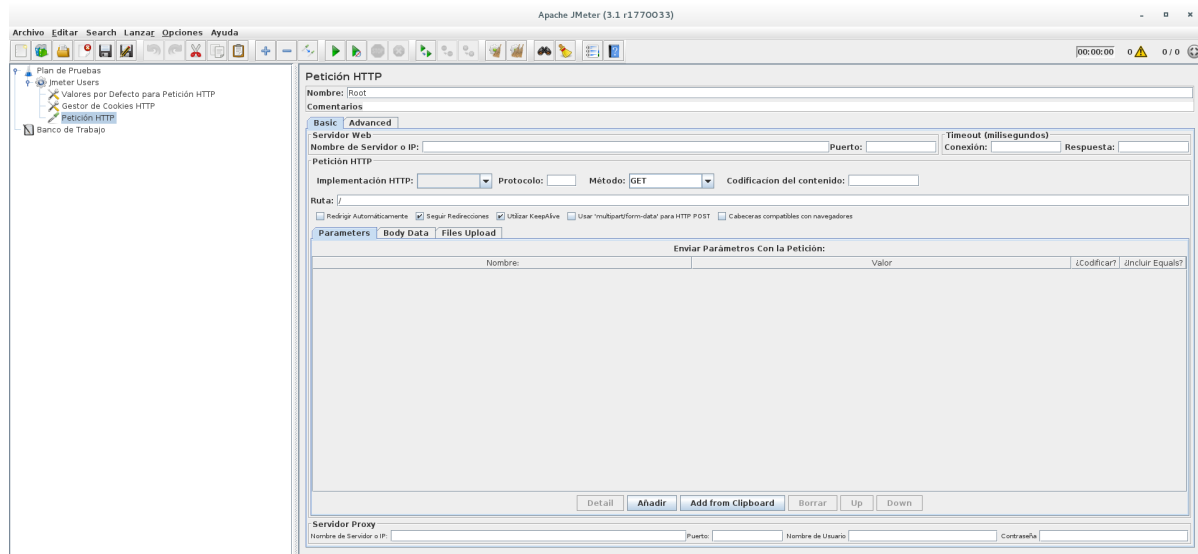


Figura 4.8: Configurando ruta principal.

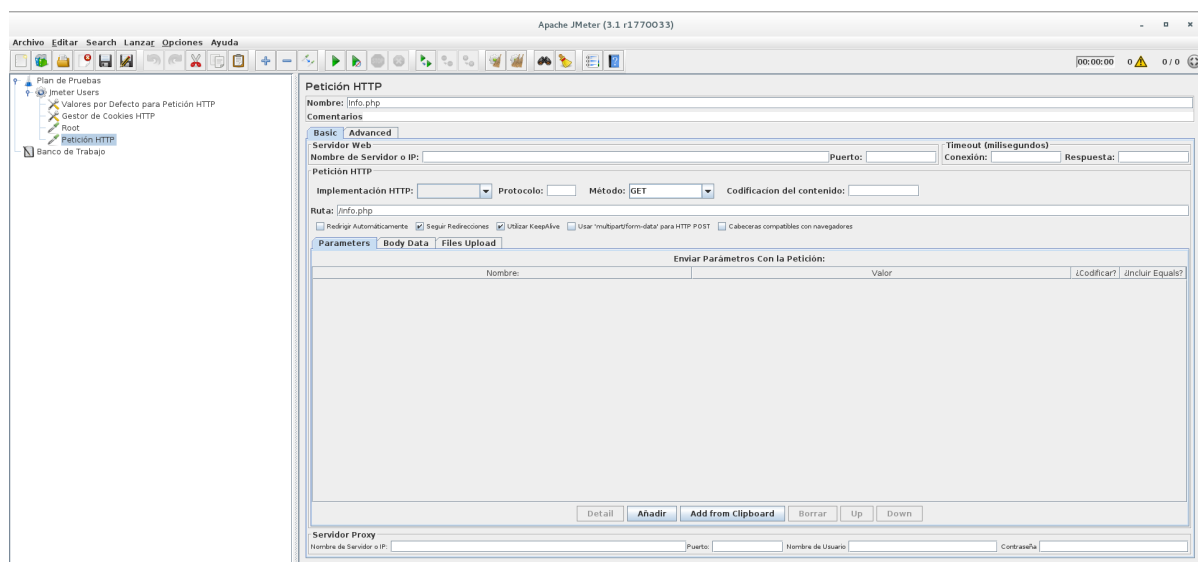


Figura 4.9: Configurando ruta info.php.

El último elemento a añadir será un *Listener* para que podamos consultar los datos obtenidos en el test de manera visual en forma de gráficos. Para ello, hacemos click con el botón secundario en *Jmeter users* y escogemos *Añadir* → *Receptor* → *Gráfico de resultados*, tras lo cual tendremos que añadir un path donde guardar dichos resultados. Ya podremos lanzar nuestro test.

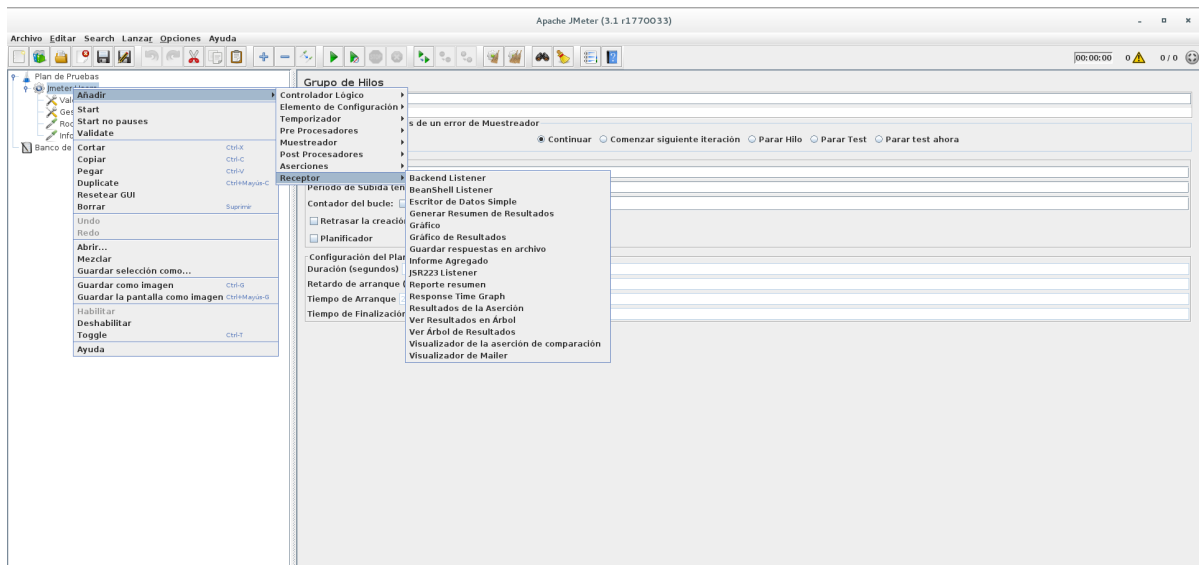


Figura 4.10: Añadiendo un *Listener*.

Como el número de peticiones no es grande, los resultados no son muy significativos. Esto lo podemos comprobar viendo los datos obtenidos, los cuales se diferencian bastante de los obtenidos con Apache Benchmark.

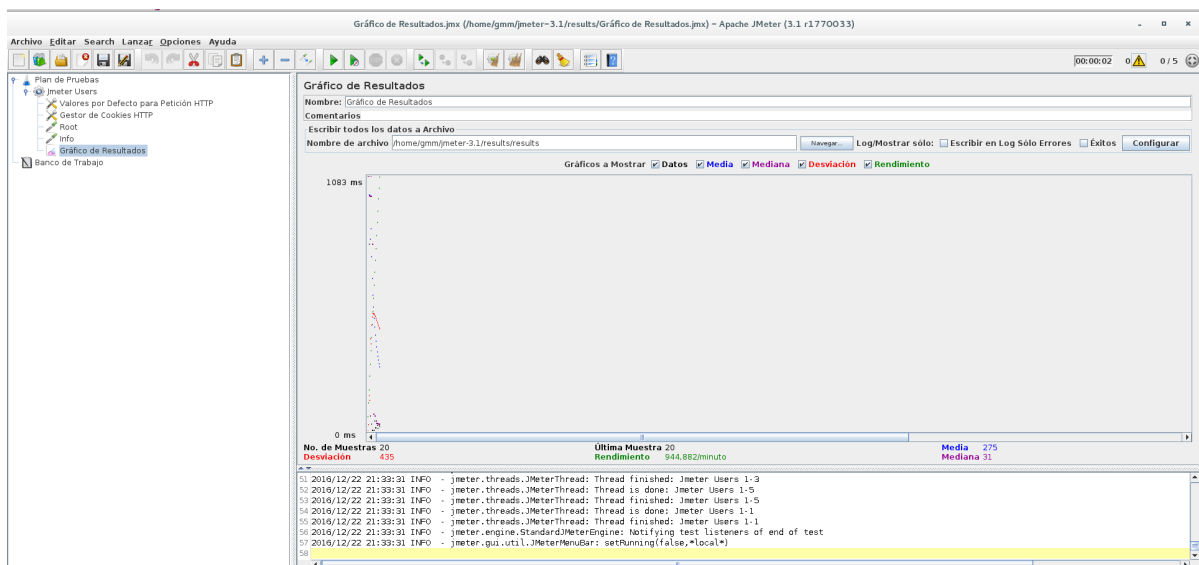


Figura 4.11: Resultados del test con jmeter.

5. Cuestión 5

- 5.1. 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; 4) Ejemplo de uso analizando los resultados.

Dada la complejidad del ejercicio, se ha optado por coger un benchmark publicado en Github [11] y modificarlo a nuestro gusto. Este se trata de un benchmark de disco implementado en bash. Su objetivo es medir el tiempo (real, usuario, sistema) que toma la creación, lectura o copiado de archivos de gran tamaño, así como el porcentaje de CPU usada para dicha tarea. De esta manera, las unidades usadas para el tiempo serán los segundos, mientras que para el uso de CPU será, como es obvio, un tanto por ciento.

Dicho esto, se muestra el código usado para dicho benchmark.

Listing 1: bash version

```
#!/bin/bash

echo "START HD-BENCHMARK"

testfile=$(hexdump -n 16 -v -e '/1 "%02X"' /dev/urandom)

if [ -z "$testfile" ]; then
    echo "Can not create a random filename!"
    exit 1;
fi

echo ""
echo "Write a 4GB file"
time sh -c "dd if=/dev/zero of=$testfile bs=1024 count
            =4000000 conv=fdatasync 2>&1 | tail -n1"

echo ""
echo "Read a 4GB file"
time sh -c "dd if=$testfile of=/dev/null conv=fdatasync 2>&1
            | tail -n1"

echo ""
echo "Copy 4GB file"
time cp $testfile ${testfile}1

rm ${testfile}1 $testfile
```

Tal y como podemos comprobar en el código, se realizan tres operaciones. A cada una de ellas, se le obtienen los datos mencionados anteriormente.

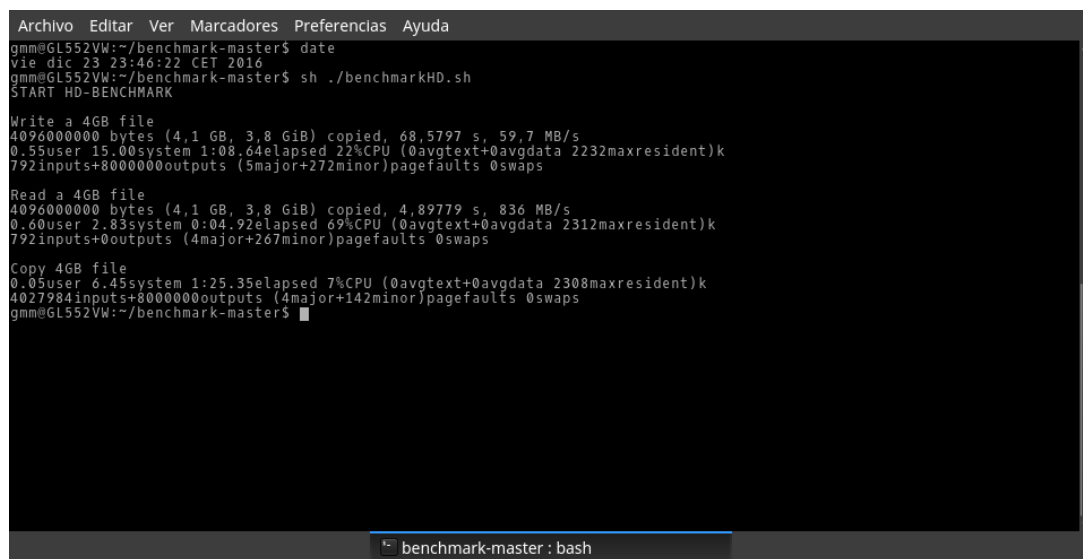
La primera instrucción se trata de crear un archivo de 4GB usando la instrucción *dd*. El contenido de este archivo será únicamente ceros, ya que se le ha puesto como entrada del comando la función */dev/zero* proporcionada por el kernel. Este fichero se guardará con un nombre aleatorio usando anteriormente el comando *hexdump*. Además, redirigimos la salida de errores o *stderr* a la salida estándar o *stdout* por si encontramos algún error durante la ejecución, mostrando únicamente la última línea.

La segunda instrucción lee el archivo creado anteriormente y no lo guarda en ningún lado, puesto que dirige la salida a la función */dev/null* proporcionada por el kernel. Esta simplemente se asegura de eliminar el flujo, dejando la instrucción sin salida. Se toma también las anteriores precauciones mostrando los errores por la salida estándar.

En la última instrucción, simplemente se copia el fichero creado en la primera instrucción en otro fichero temporal. Por último, borramos todos los ficheros temporales creados durante la ejecución del benchmark.

Para la medición de estas instrucciones se ha usado el comando *time* [12], el cual muestra por defecto, entre otras cosas, el tiempo real, de usuario, del sistema y el uso de CPU.

Pasamos al análisis de resultados. Eliminamos la mayoría de procesos que puedan estar realizando operaciones de disco y ejecutamos varias veces el benchmark para reducir la probabilidad de errores. Los resultados de un ejemplo de su ejecución lo encontramos en la siguiente captura.



```
Archivo  Editar  Ver  Marcadores  Preferencias  Ayuda
gmm@GL552VW:~/benchmark-master$ date
vie dic 23 23:46:22 CET 2016
gmm@GL552VW:~/benchmark-master$ sh ./benchmarkHD.sh
START HD-BENCHMARK

Write a 4GB file
4096000000 bytes (4.1 GB, 3.8 GiB) copied, 68.5797 s, 59.7 MB/s
0.55user 15.00system 1:08.64elapsed 22%CPU (0avgtext+0avgdata 2232maxresident)k
792inputs+8000000outputs (5major+272minor)pagefaults 0swaps

Read a 4GB file
4096000000 bytes (4.1 GB, 3.8 GiB) copied, 4.89779 s, 836 MB/s
0.60user 2.83system 0:04.92elapsed 69%CPU (0avgtext+0avgdata 2312maxresident)k
792inputs+0outputs (4major+267minor)pagefaults 0swaps

Copy 4GB file
0.05user 6.45system 1:25.35elapsed 7%CPU (0avgtext+0avgdata 2308maxresident)k
4027984inputs+8000000outputs (4major+142minor)pagefaults 0swaps
gmm@GL552VW:~/benchmark-master$
```

Figura 5.1: Resultados del benchmark modificado.

En ellos podemos comprobar como el tiempo de lectura es mucho menor que el tiempo de escritura, nada fuera de lo normal. Como es obvio, el ratio de MB/s será por el

contrario mayor en la lectura que en la escritura. Algo inusual ocurre con el tiempo de copia, ya que debería de ser menor que el tiempo de escritura, pero por algún motivo que llegamos a descubrir lo supera.

Referencias

- [1] [*Phoronix Test Suite*](#). Consultado el 17 de diciembre de 2016.
- [2] [*ZIP Benchmarks \[1612175-KH-ZIP15743918\]*](#). Consultado el 17 de diciembre de 2016.
- [3] [*7-Zip Compression \[pts/compress-7zip\]*](#). Consultado el 17 de diciembre de 2016.
- [4] [*ab - Linux man page*](#). Consultado el 16 de diciembre de 2016.
- [5] [*Alberto Guillén - UGR*](#). Consultado el 16 de diciembre de 2016.
- [6] [*Apache2 Ubuntu Default Page*](#). Consultado el 16 de diciembre de 2016.
- [7] [*The Unicode Consortium*](#). Consultado el 16 de diciembre de 2016.
- [8] [*UTF-8, a transformation format of ISO 10646*](#). Consultado el 16 de diciembre de 2016.
- [9] [*Getting Started*](#). Consultado el 17 de diciembre de 2016.
- [10] [*Download Apache JMeter*](#). Consultado el 17 de diciembre de 2016.
- [11] jacklib. [*benchmark*](#). Consultado el 23 de diciembre de 2016.
- [12] [*time - Linux man page*](#). Consultado el 23 de diciembre de 2016.