



ugr

Universidad
de Granada

Grado en Ingeniería Informática.

Práctica 4.

Nombre de la asignatura:

Ingeniería de Servidores.

Realizado por:

Néstor Rodríguez Vico



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS
INFORMÁTICA Y DE TELECOMUNICACIÓN.

Granada, 21 de diciembre de 2016.

Índice

1. Cuestión 1: Seleccione, instale y ejecute uno, comente los resultados. Atención: no es lo mismo un benchmark que una suite, instale un benchmark. 4
2. Cuestión 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) ¿cuántas “tarefas” crea ab en el cliente? 7
3. Cuestión 3: Ejecute ab contra a las tres máquinas virtuales (desde el SO anfitrión a las máquinas 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: Instale 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? 11
5. Cuestión 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. 4) Ejemplo de uso analizando los resultados. 14
6. Cuestión opcional 1: ¿Qué es Scala? Instale Gatling y pruebe los escenarios por defecto. 17

Índice de figuras

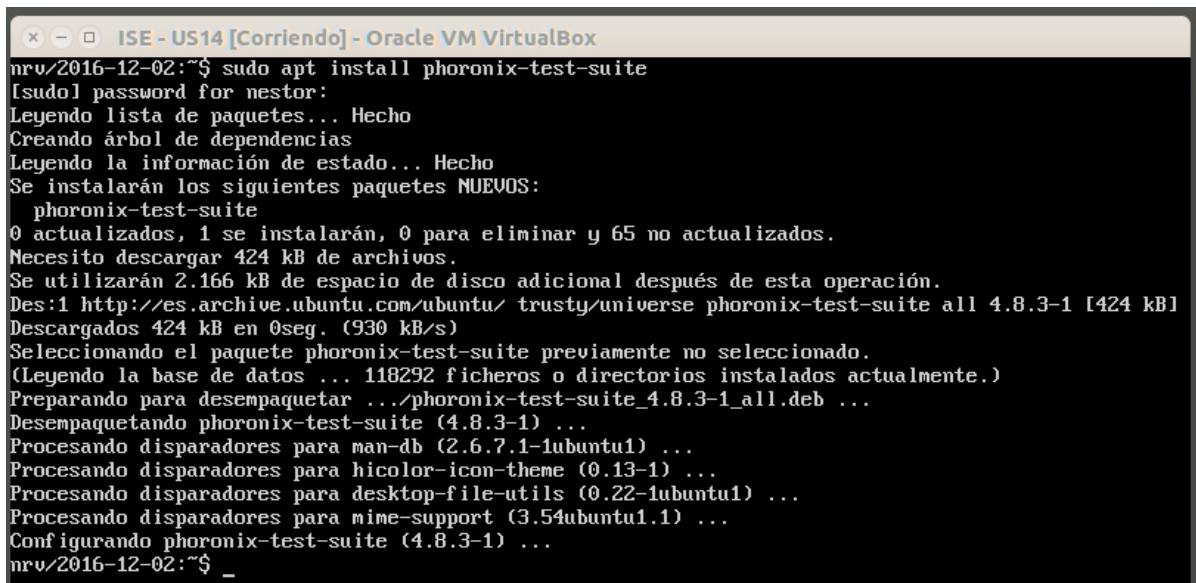
1.1. Instalación de la suite <i>Phoronix</i>	4
1.2. Benchmarks disponibles.	5
1.3. Instalación del benchmark <i>pts/build-apache</i> (1).	5
1.4. Instalación del benchmark <i>pts/build-apache</i> (2).	6
1.5. Resultados del benchmark.	6
2.1. Resultado parcial de la monitorización y número de tareas creadas. . . .	7
2.2. Resultado completo de la monitorización.	8
3.1. Resultado de la monitorización en Ubuntu Server.	9
3.2. Resultado de la monitorización en CentOS.	9
3.3. Resultado de la monitorización en Windows Server.	10
4.1. Creación del grupo de hilos.	12
4.2. Configuración de la tarea a realizar por los usuarios.	13
4.3. Configuración de la petición HTTP.	13
4.4. Resultado de <i>JMeter</i>	14
5.1. Resultado del benchmark.	15
6.1. Instalación de <i>Scala</i>	18
6.2. Ejecución de <i>Gatling</i> y posibles escenarios.	18
6.3. Resultado de la ejecución de <i>Gatling</i>	19
6.4. Copia de los resultados y dirección IP del servidor.	20
6.5. Resultado de <i>Gatling</i> desde el navegador.	20

Índice de tablas

3.1. Comparación de los tres sistemas operativos.	10
---	----

1. Cuestión 1: Seleccione, instale y ejecute uno, comente los resultados. Atención: no es lo mismo un benchmark que una suite, instale un benchmark.

Esta cuestión la voy a realizar en Ubuntu Server. Lo primero que tengo que hacer es instalar la suite *Phoronoxi* [1]. Para ello ejecutamos `sudo apt-get install phoronix-test-suite`, como podemos ver en la figura 1.1.



```
ISE - US14 [Corriendo] - Oracle VM VirtualBox
nrv/2016-12-02:~$ sudo apt install phoronix-test-suite
[sudo] password for nestor:
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 se instalarán, 0 para eliminar y 65 no actualizados.
Necesito descargar 424 kB de archivos.
Se utilizarán 2.166 kB de espacio de disco adicional después de esta operación.
Des:1 http://es.archive.ubuntu.com/ubuntu/ trusty/universe phoronix-test-suite all 4.8.3-1 [424 kB]
Descargados 424 kB en 0seg. (930 kB/s)
Seleccionando el paquete phoronix-test-suite previamente no seleccionado.
(Leyendo la base de datos ... 118292 ficheros o directorios instalados actualmente.)
Preparando para desempaquetar .../phoronix-test-suite_4.8.3-1_all.deb ...
Desempaquetando phoronix-test-suite (4.8.3-1) ...
Procesando disparadores para man-db (2.6.7.1-1ubuntu1) ...
Procesando disparadores para hicolor-icon-theme (0.13-1) ...
Procesando disparadores para desktop-file-utils (0.22-1ubuntu1) ...
Procesando disparadores para mime-support (3.54ubuntu1.1) ...
Configurando phoronix-test-suite (4.8.3-1) ...
nrv/2016-12-02:~$ _
```

Figura 1.1: Instalación de la suite *Phoronix*.

Una vez instalado la suite, listamos los benchmark disponibles ejecutando `phoronix-test-suite list-tests`. Parte de ese listado lo podemos ver en al figura 1.2. Dado que en la figura 1.3 no podemos ver todos los benchmarks disponibles, recomiendo visitar la página de OpenBenchmarking [2] para poder ver todos los benchmarks disponibles. Una vez hemos elegido el benchmark que queremos ejecutar, `pts/build-apache` en mi caso, lo instalamos con `phoronix-test-suite install pts/build-apache`, como podemos ver en la figura 1.3 y en la figura 1.4.

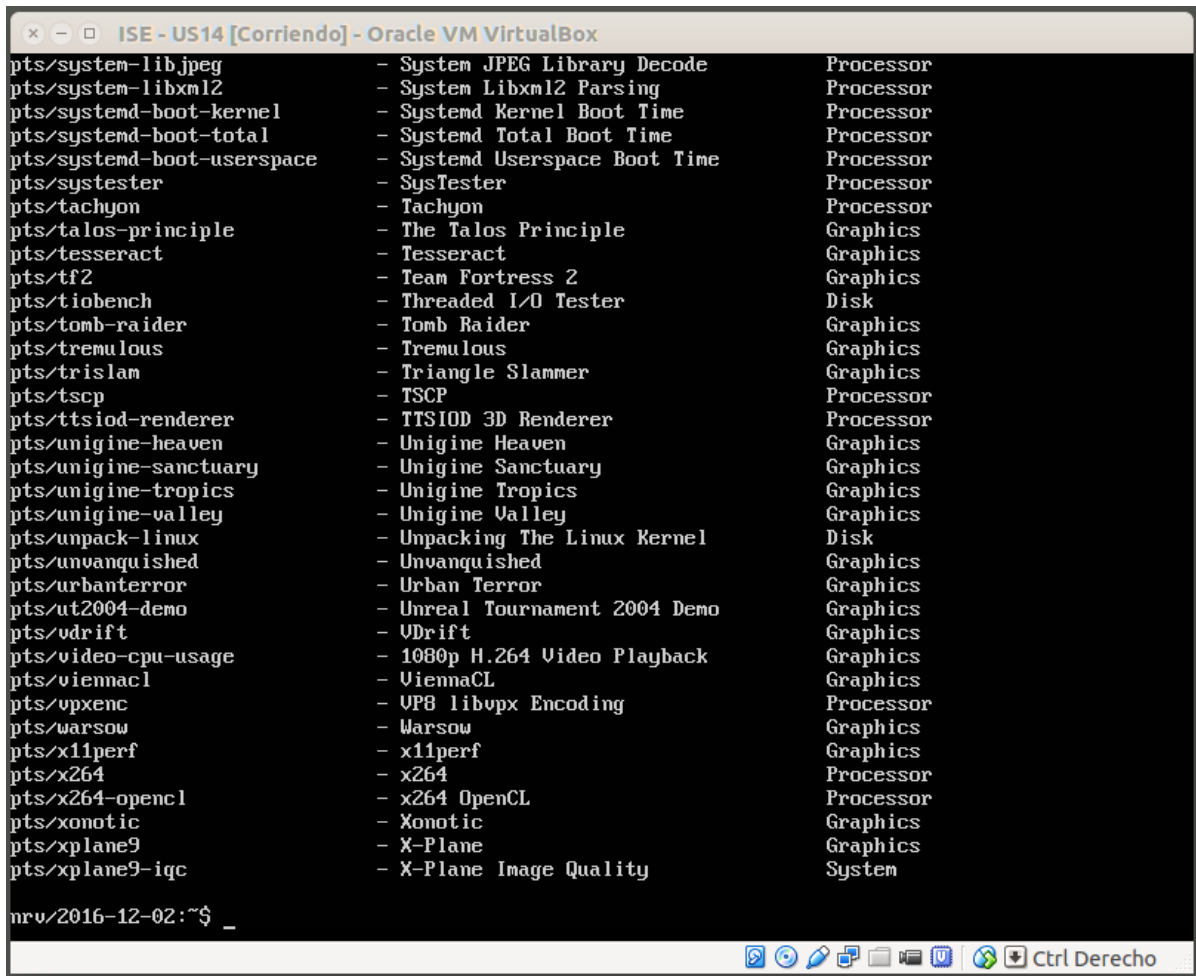


Figura 1.2: Benchmarks disponibles.

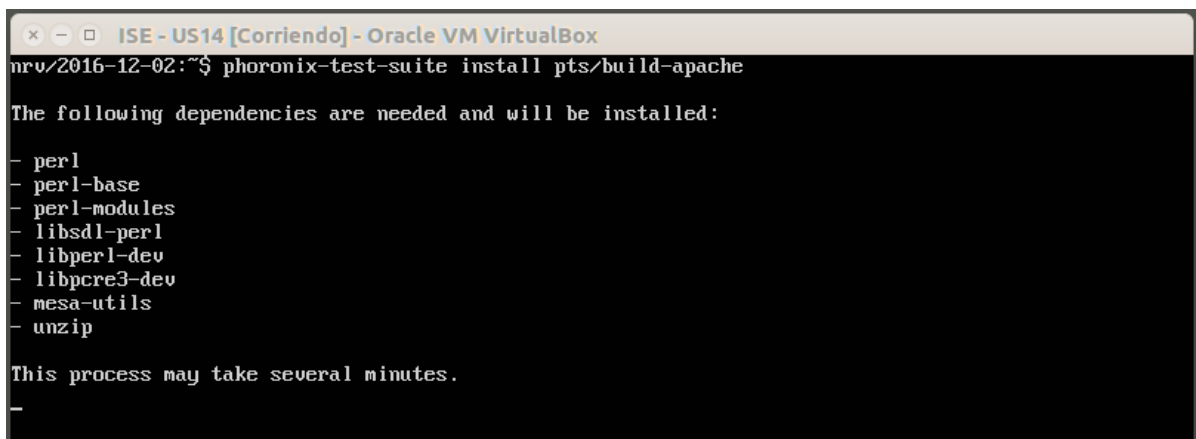


Figura 1.3: Instalación del benchmark *pts/build-apache* (1).

```
Phoronix Test Suite v4.8.3

To Install: pts/build-apache-1.5.1

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

1 Test To Install
  3 Files To Download [6.21MB]

pts/build-apache-1.5.1:
  Test Installation 1 of 1
  3 Files Needed [6.21 MB]
  Downloading: httpd-2.4.7.tar.bz2 [4.77MB]
  Downloading .....
  Downloading: apr-1.5.0.tar.bz2 [0.78MB]
  Estimated Download Time: 1m .....
  Downloading: apr-util-1.5.3.tar.bz2 [0.66MB]
  Estimated Download Time: 1m .....
  Installing Test @ 09:29:42

nrv/2016-12-02:~$ _
```

Figura 1.4: Instalación del benchmark *pts/build-apache* (2).

Una vez instalado, ejecutamos *phoronix-test-suite benchmark pts/build-apache* para iniciar el benchmark. En la figura 1.5 podemos ver los resultados del benchmark. Como podemos ver se han realizado 3 tests, los cuales han tardado 118.5369 segundos, 118.5798 segundos y 118.1869 segundos respectivamente. También podemos ver el instante de tiempo el que se han iniciado los test y el valor medio de todas las ejecuciones, 118.43 segundos en mi caso.

```
Timed Apache Compilation 2.4.7:
pts/build-apache-1.5.1
Test 1 of 1
Estimated Trial Run Count: 3
Estimated Time To Completion: 13 Minutes
Running Pre-Test Script @ 10:35:11
Started Run 1 @ 10:35:44
Running Interim Test Script @ 10:37:47
Started Run 2 @ 10:37:49
Running Interim Test Script @ 10:39:48
Started Run 3 @ 10:39:50 [Std. Dev: 0.18%]
Running Post-Test Script @ 10:41:48

Test Results:
118.53691315651
118.57985901833
118.1869161129

Average: 118.43 Seconds

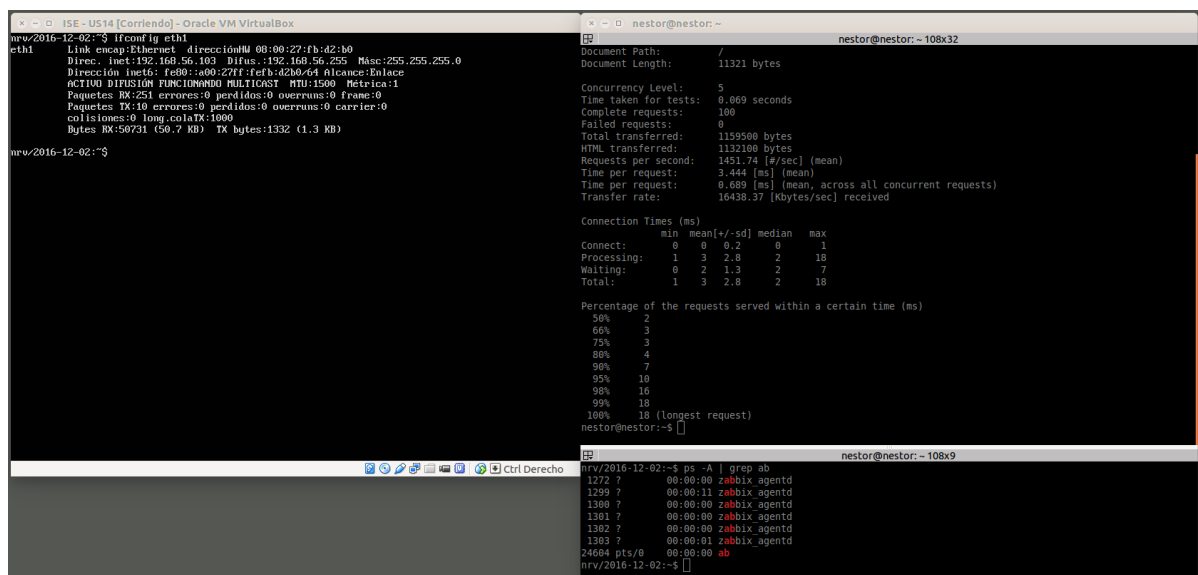
nrv/2016-12-02:~$
```

Figura 1.5: Resultados del benchmark.

2. Cuestión 2: 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 “tareas” crea `ab` en el cliente?

Dicha información la podemos obtener en la documentación de Apache [3]. El argumento `-c 5` indica la concurrencia, es decir, el número de peticiones que se van a realizar de manera simultánea. El argumento `-n 100` indica el número de solicitudes que se van a realizar durante el benchmark.

Voy a ejecutar `ab` contra Ubuntu Server desde mi máquina anfitriona, así que he tenido que instalar `ab` ejecutando `sudo apt install apache2-utils`. Para ello ejecutamos la orden `ab -c 5 -n 100 DireccionIPServidor`. La dirección IP de mi servidor, como podemos ver en la figura 2.1, es `192.168.56.103` (máquina conectadas en modo *host-only*), así que ejecutamos `ab -c 5 -n 100 http://192.168.56.103`, como podemos ver en la figura 2.1. Para ver cuantas tareas se han creado en el cliente, ejecutamos `ps -A | grep ab`. Como podemos ver en la figura 2.1 se ha creado sólo una tarea. En la figura 2.2 podemos ver el resultado completo de la monitorización. Podemos ver desde información básica del servidor hasta el número de bytes transferidos o el tiempo de media empleado en los tests.



```
nv/2016-12-02:~$ ifconfig eth1
eth1    Link encap:Ethernet  direcciónHW 08:00:27:fb:42:b0
        Dirección inet: 192.168.56.103  Difus.:192.168.56.255  Másc:255.255.255.0
        Dirección inet6: fe80::a00:27ff:febf:42b0/64 Alcance:Enlace
        ACTIVO DIFUSION FUNCIONANDO MULTICAST  MTU:1500  Métrica:1
        Paquetes RX:251 errores:0 perdidos:0 overruns:0 frame:0
        Paquetes TX:10 errores:0 perdidos:0 overruns:0 carrier:0
        colisiones:0 long.colaTX:1000
        Bytes RX:50731 (50.7 KB)  TX bytes:1332 (1.3 KB)

nv/2016-12-02:~$

nestor@nestor:~$ ab -c 5 -n 100 http://192.168.56.103
Document Path: /
Document Length: 11321 bytes

Concurrency Level: 5
Time taken for tests: 0.069 seconds
Complete requests: 100
Failed requests: 0
Total transferred: 1159500 bytes
HTML transferred: 1132100 bytes
Requests per second: 1451.74 (#/sec) (mean)
Time per request: 3.444 [ms] (mean)
Time per request: 0.689 [ms] (mean, across all concurrent requests)
Transfer rate: 16438.37 [Kbytes/sec] received

Connection Times (ms)
              min      mean[+/-sd] median   max
Connect:        0        0  0.2      0      1
Processing:      1        3  2.6      2     18
Waiting:        0        2  1.3      2      7
Total:          1        3  2.8      2     18

Percentage of the requests served within a certain time (ms)
 50%    2
 66%    3
 75%    3
 88%    4
 90%    7
 95%   10
 98%   16
 99%   18
100%   18 (longest request)
nestor@nestor:~$

nv/2016-12-02:~$ ps -A | grep ab
1272 ?        00:00:00 zabbix_agentd
1299 ?        00:00:11 zabbix_agentd
1300 ?        00:00:00 zabbix_agentd
1301 ?        00:00:00 zabbix_agentd
1302 ?        00:00:00 zabbix_agentd
1303 ?        00:00:01 zabbix_agentd
24604 pts/0    00:00:00 ab
nv/2016-12-02:~$
```

Figura 2.1: Resultado parcial de la monitorización y número de tareas creadas.

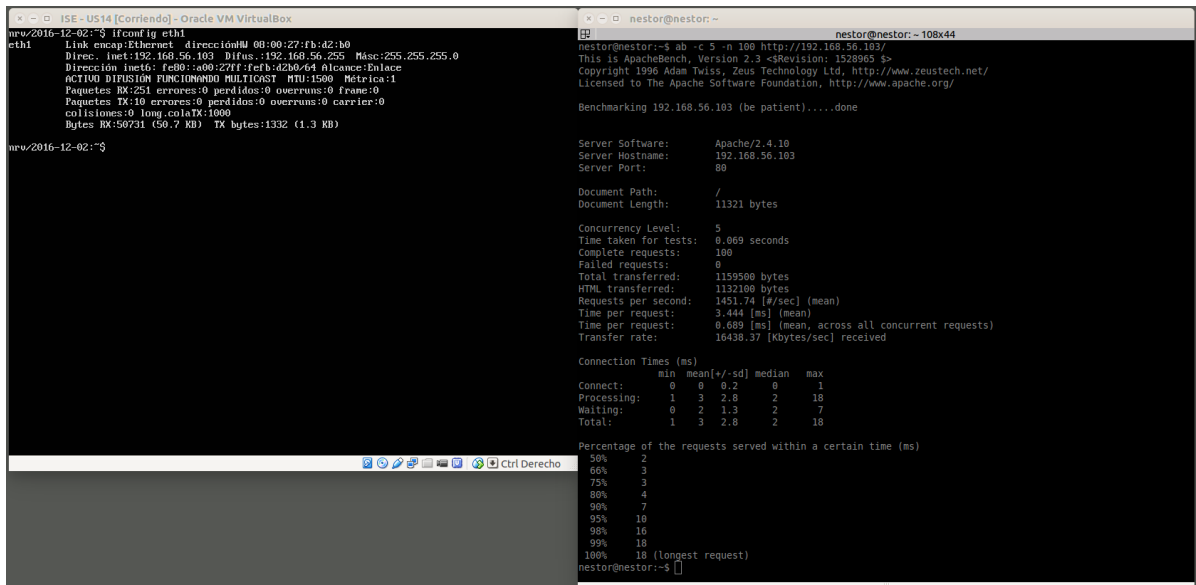


Figura 2.2: Resultado completo de la monitorización.

3. Cuestión 3: 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).

Para monitorizar en las mismas condiciones voy a ejecutar el mismo comando desde mi máquina anfitriona (máquinas conectadas en modo *host-only*) y contra la misma página web ¹. Dicho archivo lo he ubicado en el directorio `/var/www/` en el caso de Ubuntu Server y CentOS y en el directorio `C:\inetpub\wwwroot` en el caso de Windows Server. El comando es el mismo que el usado en la cuestión anterior, `ab -c 5 -n 100 DirecciónIPServidor`. El resultado de la monitorización en Ubuntu Server lo podemos ver en la figura 3.1, el resultado de CentOS lo podemos ver en la figura 3.2 y el resultado de Windows Server lo podemos ver en la figura 3.3.

¹La página web usada se encuentra dentro de la carpeta *Archivos auxiliares* bajo el nombre “*practica4.html*”.

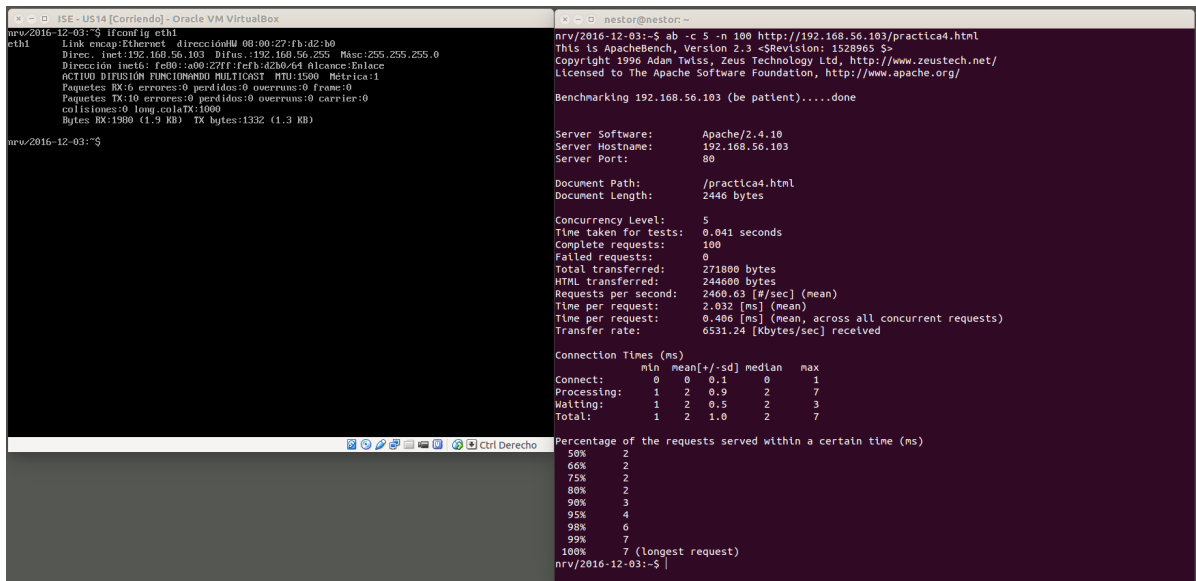


Figura 3.1: Resultado de la monitorización en Ubuntu Server.

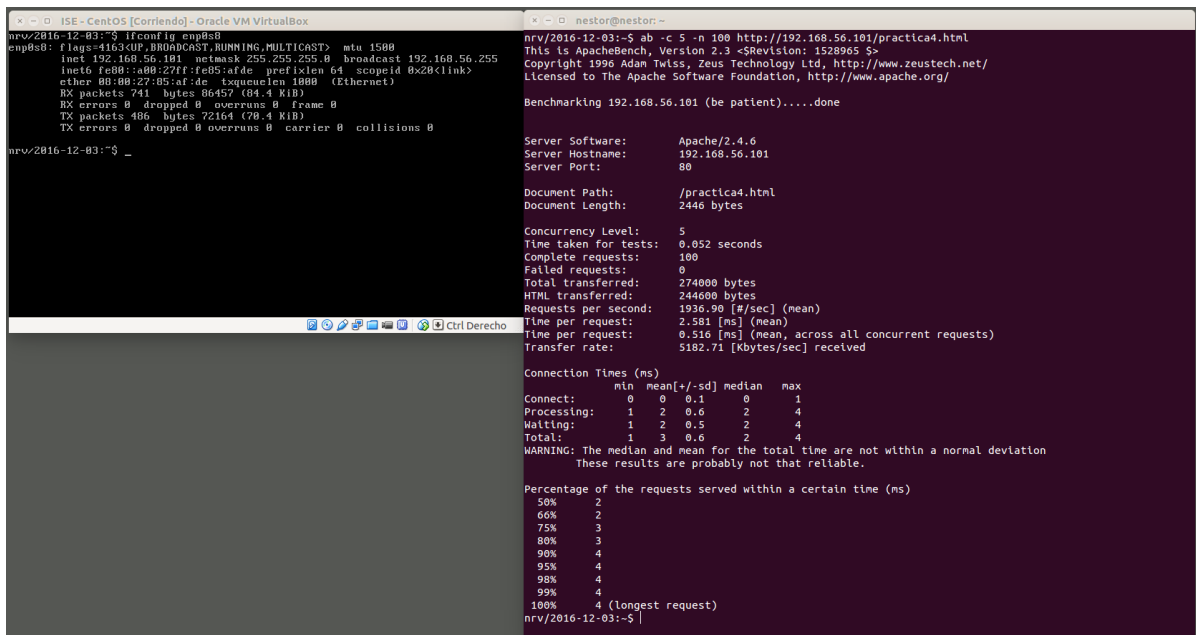


Figura 3.2: Resultado de la monitorización en CentOS.

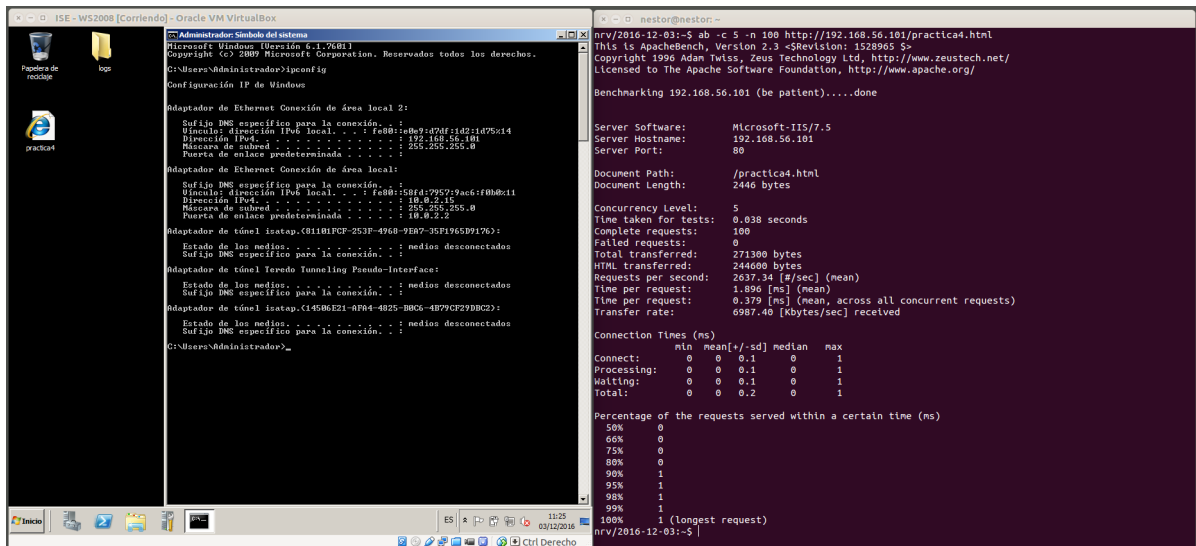


Figura 3.3: Resultado de la monitorización en Windows Server.

Para poder comparar mejor los resultados, vamos a extraer los datos más relevantes en la tabla 3.1:

	Tiempo total (s)	Promedio de solicitudes procesadas por segundo	Tiempo medio por solicitud (ms)	Bytes transmitidos	Velocidad de transferencia (KB/s)
Ubuntu Server	0.041	2460.63	2.032	271800	6531.24
CentOS	0.052	1936.90	2.581	274000	5182.71
Windows Server	0.038	2637.34	1.896	271400	6987.40

Tabla 3.1: Comparación de los tres sistemas operativos.

Tras observar la tabla, podemos decir que el que mejor resultados ofrece ha sido Windows Server, pero esto no quiere decir que siempre sea mejor, ya que el datos obtenidos pueden variar entre distintas monitorizaciones. También cabe destacar que el número de bytes transmitidos es diferente, a pesar de transmitir la misma página en los tres sistemas operativos.

4. Cuestión 4: Instale 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?

Voy a monitorizar Ubuntu Server desde mi máquina anfitriona (máquinas conectadas en modo *host-only*). Para ello, instalamos *JMeter* en mi portátil ejecutando *sudo apt install jmeter*. Una vez instalado, lo lanzamos ejecutando *jmeter*. Como bien nos indica el enunciado, voy a seguir la información de *JMeter* [4] para realizar este ejercicio. Los pasos a seguir son los siguientes:

1. Añadimos un grupo de hilos. Un grupo de hilos indica a *JMeter* cuantos usuarios queremos simular, cuántas peticiones van a mandar y con que frecuencia serán mandadas. Para añadir el grupo de hilos, le damos click derecho a *Plan de Prueba*, luego le damos a *Añadir*, a continuación elegimos *Hilos (Usuarios)* y por último elegimos *Grupo de hilos*.
2. A continuación configuramos dicho grupo. Los parámetros que he modificado son los que podemos ver a continuación y se ve pueden ver en la figura 4.1.
 - **Nombre:** Este parámetro es más que nada por estética. En mi caso lo he cambiado a *Usuarios*.
 - **Número de Hilos:** En mi caso, voy a simular 30 usuarios.
 - **Periodo de subida (en segundos):** En mi caso, lo he dejado a 1, que es el valor por defecto. Este parámetro representa el retraso que se produce entre la creación de dos usuarios.
 - **Contador del bucle Count:** Indica cuantas veces *JMeter* debe repetir el test. En mi caso le he indicado que repita el test 20 veces.
3. Una vez creados los usuarios, vamos a definir las tareas que van a realizar. Para ello seleccionamos *Usuarios*, el grupo que hemos creado en los pasos anteriores, le damos click derecho y elegimos *Añadir*, luego elegimos *Elemento de Configuración* y finalmente *Valores por Defecto para Petición HTTP*. A continuación le indicamos la dirección IP de nuestro servidor, *192.168.56.103* (máquinas conectadas en modo *host-only*). El resto de parámetros los he dejado por defecto, tal y como podemos ver en la figura 4.2.
4. Añadimos un gestor de cookies. Para ello seleccionamos *Usuarios*, el grupo que hemos creado en los pasos anteriores, le damos click derecho y elegimos *Añadir*, luego elegimos *Elemento de Configuración* y finalmente *Gestor de Cookies HTTP*.

5. A continuación añadimos las peticiones HTTP. Para ello seleccionamos *Usuarios*, el grupo que hemos creado en los pasos anteriores, le damos click derecho y elegimos *Añadir*, luego elegimos *Muestreador* y finalmente *Petición HTTP*. Dentro de la petición, indicamos la ruta del archivo HTML, en mi caso he elegido la raíz de mi servidor, /, tal y como podemos ver en la figura 4.3.
6. Finalmente, añadimos un *Listener* para ver y almacenar los resultados. Para ello seleccionamos *Usuarios*, el grupo que hemos creado en los pasos anteriores, le damos click derecho y elegimos *Añadir*, luego elegimos *Receptor* y finalmente *Gráfico de Resultados*.
7. A continuación, guardamos nuestro plan de pruebas. Finalmente, lo ejecutamos pulsando el botón *play* verde en el programa.

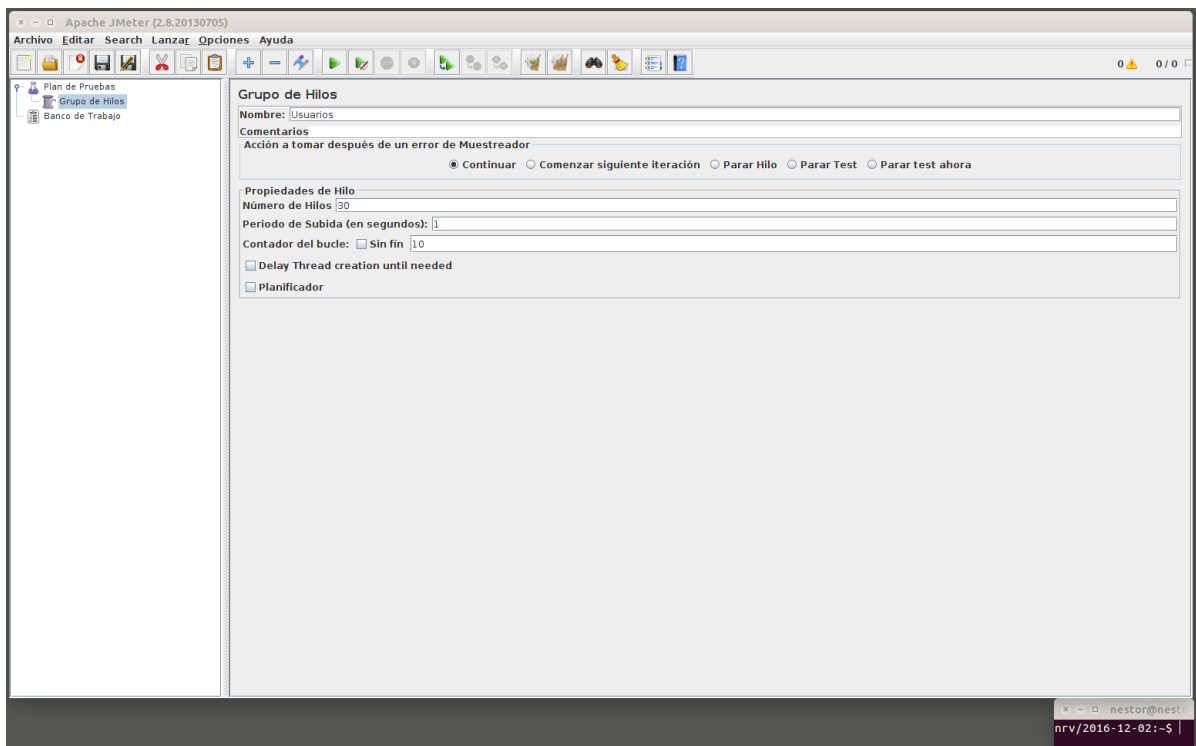


Figura 4.1: Creación del grupo de hilos.

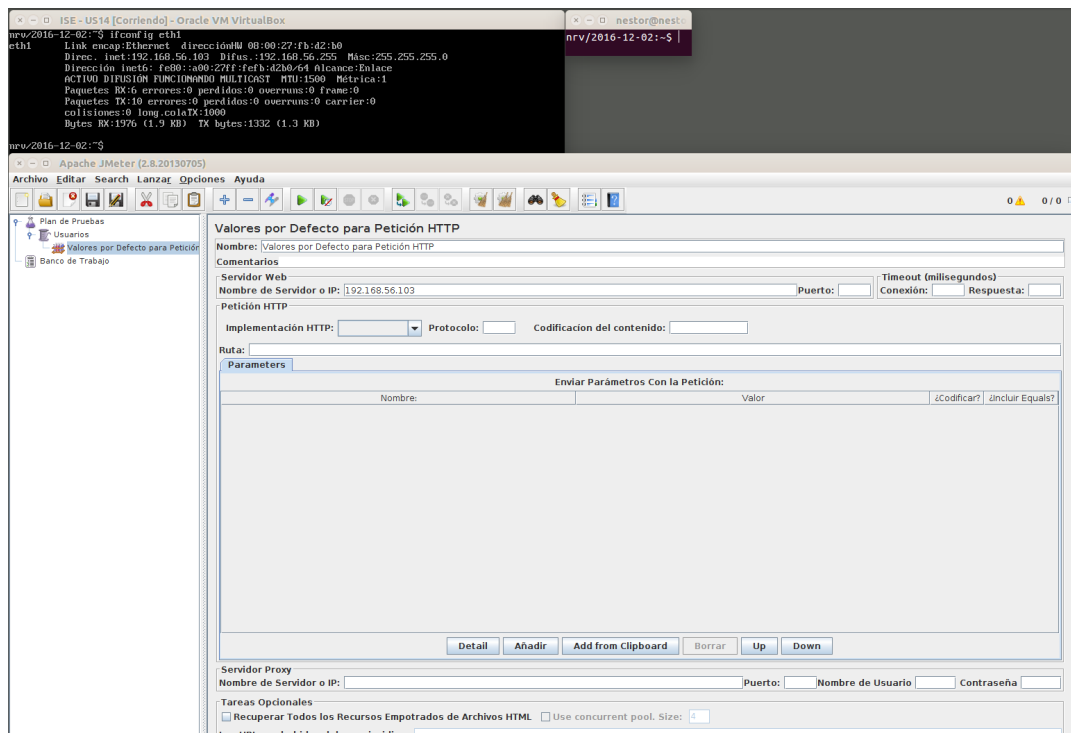


Figura 4.2: Configuración de la tarea a realizar por los usuarios.

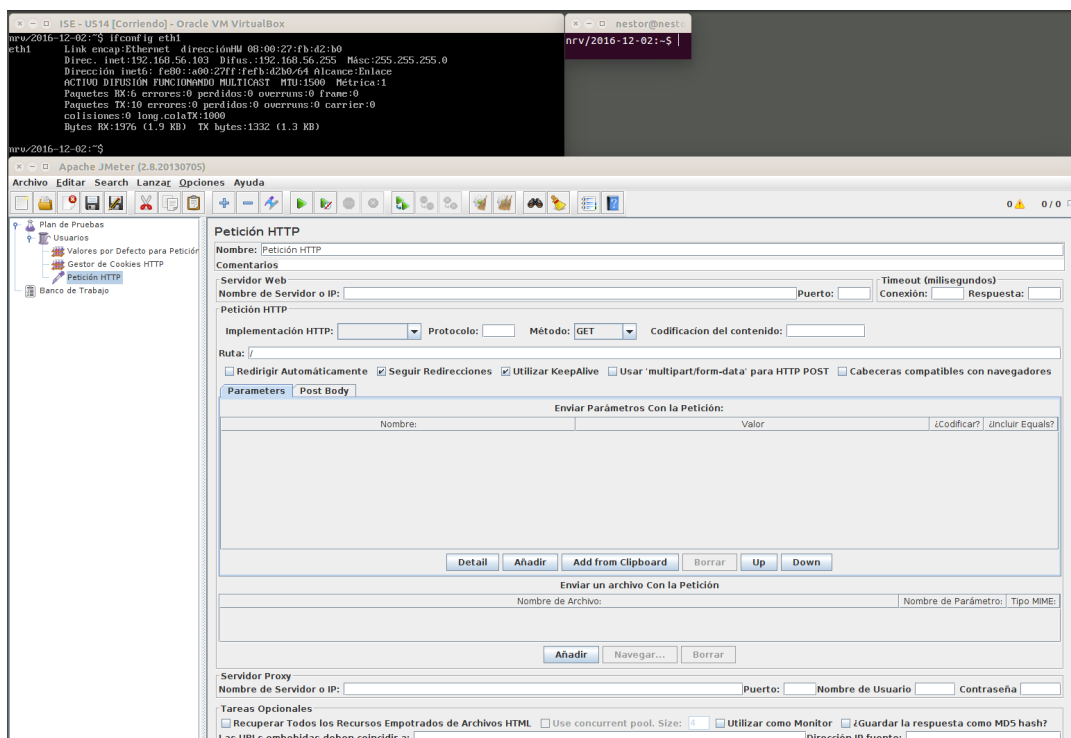


Figura 4.3: Configuración de la petición HTTP.

Una vez ejecutado, el resultado lo podemos ver en la figura 4.4. En dicha figura podemos una representación gráfica de los datos recogidos. Los resultados son distintos a los proporcionados por *ab*. Desde mi punto de vista, me gusta más como *ab* representa los datos, ya que veo la información de manera más clara.

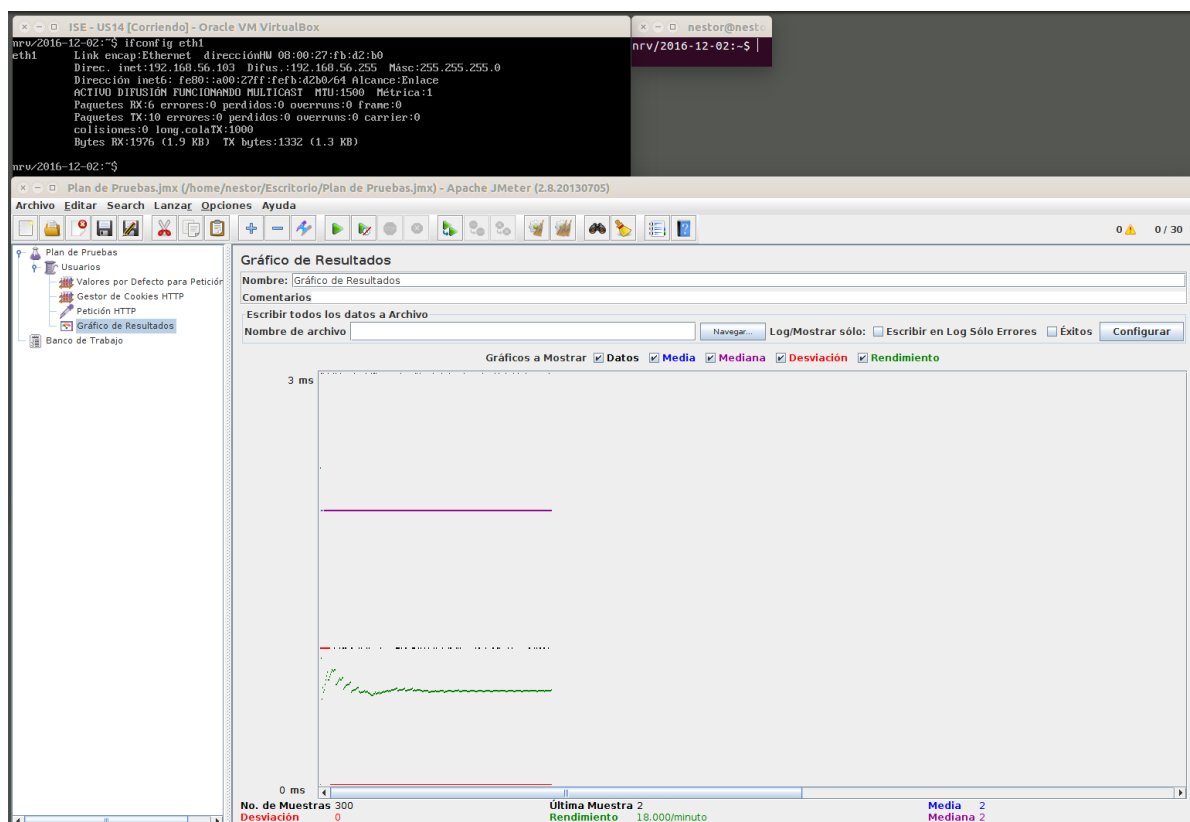
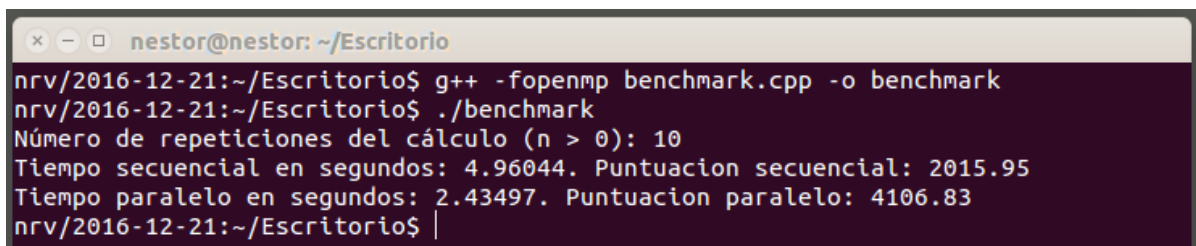


Figura 4.4: Resultado de *JMeter*.

5. **Cuestión 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. 4) Ejemplo de uso analizando los resultados.**
1. El objetivo del benchmark es conseguir un tiempo de ejecución para una tarea predeterminada para poder comparar distintas CPU. El benchmark consiste en la multiplicación de dos matrices y la escritura del resultado en una tercera matriz. La multiplicación se ha realizado de dos maneras: de forma secuencial y usando paralelización.

2. La ejecución del benchmark devuelve dos tiempos, uno para la ejecución secuencial y otro para la ejecución en paralelo. El tiempo de la ejecución secuencial, nos indica el tiempo empleado en la multiplicación usando una hebra mientras que el tiempo de ejecución usando paralelización nos indica el tiempo empleado en la multiplicación usando todas las hebras que permita la arquitectura de la máquina. A parte de los tiempos, nos devuelve una puntuación, ya que tradicionalmente los benchmarks suelen darte una puntuación para tener una referencia más visual. Dicha puntuación se calcula como $10000/\text{tiempo}$. Una mayor puntuación indica un mejor resultado.
3. Para usar este benchmark lo compilamos ejecutando `g++ -fopenmp benchmark.cpp -o benchmark`. Lo ejecutamos mediante el comando `./benchmark`. A continuación nos pide el número de veces que se va a calcular la multiplicación de las matrices, para así luego poder calcular el tiempo medio de dichas ejecuciones.
4. El resultado de la ejecución del benchmark en mi máquina se puede ver en la figura 5.1. Podemos ver que en mi caso, el tiempo medio de ejecución para una ejecución secuencial es de 4.96044, obteniendo así una puntuación de 2015.95. En el caso de la ejecución paralela, el tiempo medio de ejecución es de 2.43497, obteniendo así una puntuación de 4106.83.



```

nestor@nestor: ~/Escritorio
nrv/2016-12-21:~/Escritorio$ g++ -fopenmp benchmark.cpp -o benchmark
nrv/2016-12-21:~/Escritorio$ ./benchmark
Número de repeticiones del cálculo (n > 0): 10
Tiempo secuencial en segundos: 4.96044. Puntuacion secuencial: 2015.95
Tiempo paralelo en segundos: 2.43497. Puntuacion paralelo: 4106.83
nrv/2016-12-21:~/Escritorio$

```

Figura 5.1: Resultado del benchmark.

El benchmark usado lo podemos ver a continuación ²:

```

#include <iomanip>
#include <iostream>
#include <omp.h>
#include <vector>

using namespace std;

double t1secuencial, t2secuencial, t1paralelo, t2paralelo, t1, t2;
double M1[1000][1000], M2[1000][1000], M3[1000][1000];

double media(const vector<double> &elementos){
    double suma=0;

```

²El benchmark “*benchmark.cpp*” se encuentra dentro de la carpeta *Archivos auxiliares*.

```

        for(int i=0; i<elementos.size(); i++)
            suma+=elementos[i];
        return suma/elementos.size();
    }

    void producto_secuencial() {
        int i, j, k;
        t1secuencial = omp_get_wtime();
        for (i=0; i<1000; i++){
            for(j=0; j<1000; j++){
                for (k=0; k<1000; k++){
                    M1[i][j] = M1[i][j] + (M2[i][k]*M3[k][j]);
                }
            }
        }
        t2secuencial=omp_get_wtime();
    }

    void producto_paralelo() {
        int i, j, k;
        #pragma omp parallel shared(M1,M2,M3) private(i,j,k)
        {
            //Medida de tiempo
            #pragma omp single
            {
                t1paralelo = omp_get_wtime();
            }
            #pragma omp for schedule(runtime)
            for(i=0; i<1000; i++){
                for(j=0; j<1000; j++){
                    for(k=0; k<1000; k++){
                        M1[i][j] = M1[i][j] + M2[i][k] *
                        M3[k][j];
                    }
                }
            }
            //Medida de tiempo
            #pragma omp single
            {
                t2paralelo = omp_get_wtime();
            }
        }
    }

    int main(int argc, char *argv[]) {
        vector<double> vtsecuencia, vtparalelo;
        int i, j, iteraciones = 0;
        cout << "Número de repeticiones del cálculo (n>=0): ";
        cin >> iteraciones;

        for (i=0; i<1000; i++){
            for (j=0; j<1000; j++){

```



```

        M1[i][j] = 0;
        M2[i][j] = 2;
        M3[i][j] = 2;
    }
}

for(int i=0; i<iteraciones; i++){
    producto_secuencial();
    vtsecuencia.push_back(t2secuencial - t1secuencial);
}

for (i=0; i<1000; i++){
    for (j=0; j<1000; j++){
        M1[i][j] = 0;
        M2[i][j] = 2;
        M3[i][j] = 2;
    }
}

for(int i=0; i<iteraciones; i++){
    producto_paralelo();
    vtparalelo.push_back(t2paralelo - t1paralelo);
}

t1 = media(vtsecuencia);
t2 = media(vtparalelo);

//Para evitar que el compilador suprima codigo:
M1[0][0]++; M1[0][0]--;
M2[0][0]++; M2[0][0]--;
M3[0][0]++; M3[0][0]--;

cout << "Tiempo_secuencial_en_segundos:_" << t1 << "._Puntuacion_
    secuencial:_" << 10000/t1 << endl;
cout << "Tiempo_paralelo_en_segundos:_" << t2 << "._Puntuacion_
    paralelo:_" << 10000/t2 << endl;

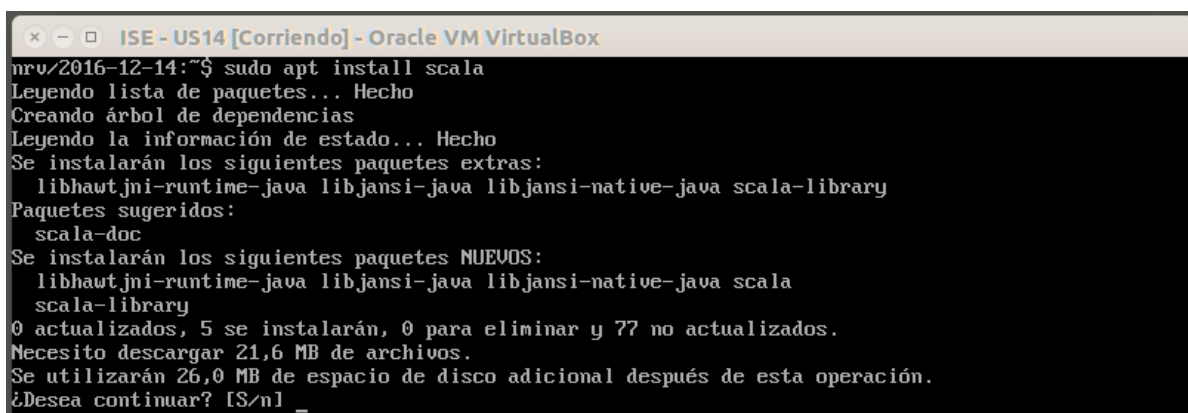
return 0;
}

```

6. Cuestión opcional 1: ¿Qué es Scala? Instale Gatling y pruebe los escenarios por defecto.

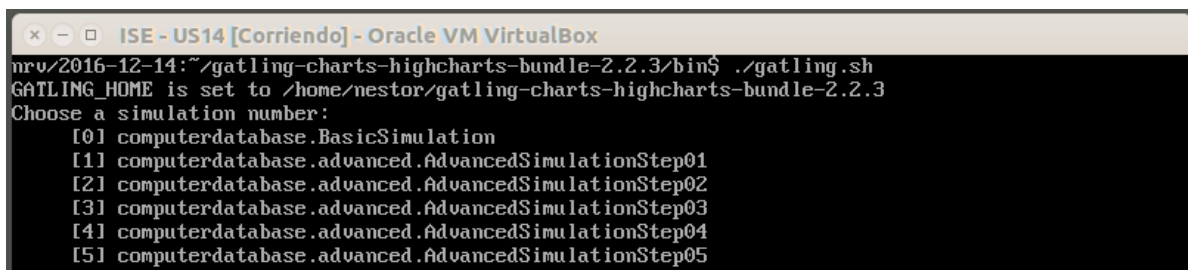
Para saber que es *Scala* no hay nada mejor que verlo en la página oficial [5]. *Scala* es un lenguaje de programación. *Scala* es el acrónimo de “*Scalable Language*”. Podemos usar *Scala* para programación orientada a objetos o para una programación más funcional. Se ejecuta sobre la máquina virtual de Java. Las clases de *Java* y *Scala* pueden combinarse sin ningún problema.

Como podemos ver en la página de *Gatling* [6], *Gatling* se basa en *Scala*, así que primero tenemos que instalar *Scala*, en mi caso en Ubuntu Server. Para ello ejecutamos `sudo apt install scala`, como podemos ver en la figura 6.1. Una vez instalado, pasamos a instalar *Gatling* siguiendo las instrucciones que podemos ver en la página oficial [7]. Debemos descargar un fichero zip y luego extraerlo. Una vez extraído, dentro de la carpeta de *Gatling* nos encontramos la carpeta *bin* y dentro de ella el fichero *gatling.sh*. Lo ejecutamos mediante el comando `./gatling.sh` como podemos ver en la figura 6.2. A continuación nos pedirá que escenario queremos usar como podemos ver en la figura 6.2, yo he elegido el 0. A continuación comenzará la prueba. El resultado lo podemos ver en la figura 6.3.



```
ISE - US14 [Corriendo] - Oracle VM VirtualBox
nrv/2016-12-14:~$ sudo apt install scala
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes extras:
  libhawtjni-runtime-java libjansi-java libjansi-native-java scala-library
Paquetes sugeridos:
  scala-doc
Se instalarán los siguientes paquetes NUEVOS:
  libhawtjni-runtime-java libjansi-java libjansi-native-java scala
  scala-library
0 actualizados, 5 se instalarán, 0 para eliminar y 77 no actualizados.
Necesito descargar 21,6 MB de archivos.
Se utilizarán 26,0 MB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n] _
```

Figura 6.1: Instalación de *Scala*.



```
ISE - US14 [Corriendo] - Oracle VM VirtualBox
nrv/2016-12-14:~/gatling-charts-highcharts-bundle-2.2.3/bin$ ./gatling.sh
GATLING_HOME is set to /home/nestor/gatling-charts-highcharts-bundle-2.2.3
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 6.2: Ejecución de *Gatling* y posibles escenarios.

```

Simulation computerdatabase.BasicSimulation completed in 23 seconds
Parsing log file(s)...
Parsing log file(s) done
Generating reports...

=====
--- Global Information ---
> request count                13 (OK=13    KO=0    )
> min response time            58 (OK=58    KO=-    )
> max response time            224 (OK=224   KO=-    )
> mean response time           93 (OK=93    KO=-    )
> std deviation                 46 (OK=46    KO=-    )
> response time 50th percentile 76 (OK=76    KO=-    )
> response time 75th percentile 80 (OK=80    KO=-    )
> response time 95th percentile 181 (OK=181   KO=-    )
> response time 99th percentile 215 (OK=215   KO=-    )
> mean requests/sec            0.542 (OK=0.542 KO=-    )
--- Response Time Distribution ---
> t < 800 ms                   13 (100%)
> 800 ms < t < 1200 ms         0 ( 0%)
> t > 1200 ms                  0 ( 0%)
> failed                       0 ( 0%)
=====

Reports generated in 1s.
Please open the following file: /home/nestor/gatling-charts-highcharts-bundle-2.2.3/results/basicsim
ulation-1481711704035/index.html
nrv/2016-12-14:~/gatling-charts-highcharts-bundle-2.2.3/bin$ _

```

Figura 6.3: Resultado de la ejecución de *Gatling*.

Para poder ver los resultados de una mejor forma, copiamos el contenido de la carpeta `/results/basicsimulation-1481711704035` a `/var/www/html`, para poder acceder desde el navegador de mi máquina anfitriona. Para ello, desde la carpeta `/results/basicsimulation-1481711704035` ejecutamos `cp -r * /var/www/html`, como podemos ver en la figura 6.4. A continuación, accedemos desde la máquina anfitriona. Para ello introducimos la dirección IP de nuestro servidor seguido de `/index.html`. Como podemos ver en la figura 6.4 la dirección IP de mi servidor es `192.168.56.103` (máquinas conectas en modo *host-only*), así que debemos introducir `192.168.56.103/index.html`. Podemos ver los resultados en la figura 6.5. Como podemos ver, se han respondido todas las solicitudes de manera correcta y todas en menos de 800ms.

```

ISE - US14 [Corriendo] - Oracle VM VirtualBox
nrv/2016-12-14:~/gatling-charts-highcharts-bundle-2.2.3/results/basicimulation-1481711704035$ sudo
cp -r * /var/www/html/
nrv/2016-12-14:~/gatling-charts-highcharts-bundle-2.2.3/results/basicimulation-1481711704035$ ifcon
fig eth1
eth1      Link encap:Ethernet  direcciónHW 08:00:27:fb:d2:b0
          Direc. inet:192.168.56.103  Difus.:192.168.56.255  Másc:255.255.255.0
          Dirección inet6: fe80::a00:27ff:fefb:d2b0/64 Alcance:Enlace
          ACTIVO DIFUSIÓN FUNCIONANDO MULTICAST MTU:1500 Métrica:1
          Paquetes RX:638 errores:0 perdidos:0 overruns:0 frame:0
          Paquetes TX:444 errores:0 perdidos:0 overruns:0 carrier:0
          colisiones:0 long.colaTX:1000
          Bytes RX:120434 (120.4 KB)  TX bytes:347208 (347.2 KB)

nrv/2016-12-14:~/gatling-charts-highcharts-bundle-2.2.3/results/basicimulation-1481711704035$ _

```

Figura 6.4: Copia de los resultados y dirección IP del servidor.

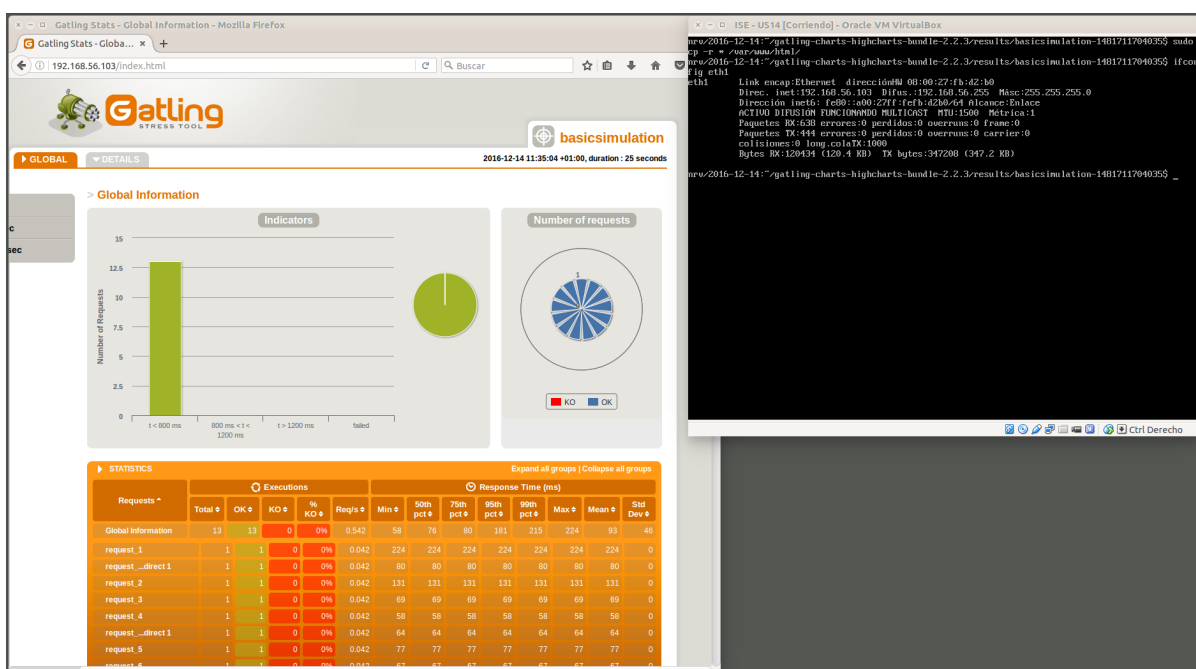


Figura 6.5: Resultado de Gatling desde el navegador.

Referencias

- [1] <http://www.phoronix-test-suite.com/>. Fecha de acceso: 02/12/2016.
- [2] <https://openbenchmarking.org/tests/pts>. Fecha de acceso: 02/12/2016.
- [3] <http://httpd.apache.org/docs/2.4/programs/ab.html>. Fecha de acceso: 02/12/2016.
- [4] <http://jmeter.apache.org/usermanual/build-web-test-plan.html>. Fecha de acceso: 02/12/2016.
- [5] <https://www.scala-lang.org/what-is-scala.html>. Fecha de acceso: 14/12/2016.
- [6] <http://gatling.io/#/>. Fecha de acceso: 14/12/2016.
- [7] <http://gatling.io/#/resources/download>. Fecha de acceso: 14/12/2016.