

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

Memoria Práctica 4

David Criado Ramón

23 de diciembre de 2016

Índice

1	Seleccione un benchmark de Phoronix Suite, instálelo y ejecútelo, comente los resultados.	4
2	De los parámetros que le podemos pasar al comando <code>ab</code> . ¿Qué significa <code>-c 5</code> ? ¿y <code>-n 100</code> ? Monitorice la ejecución de <code>ab</code> contra alguna máquina (cualquiera), ¿cuántas “tareas” crea <code>ab</code> en el cliente?	7
3	Ejecute <code>ab</code> 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	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 meter, haga el experimento usando sus máquinas virtuales ¿coincide con los resultados de <code>ab</code> ?	11
5	Programe un benchmark usando el lenguaje que desee.	17

Índice de figuras

1.1.	Realizamos la instalación de <i>Phoronix Suite</i> con <i>apt</i> en Ubuntu Server 14.04.	4
1.2.	Listado de test de memoria en <i>Phoronix Suite</i> .	4
1.3.	Como es la primera vez que se realiza el test, se inicia la descarga del mismo.	5
1.4.	Escojo una opción del test, en mi caso, velocidad media (<i>5. Average</i>). y con datos en coma flotante (<i>2. Floating Point</i>). Podemos observar la información del sistema y escoger el nombre de los datos a guardar.	5
1.5.	Información del PC que se ha usado para el benchmark.	6
1.6.	Información de los resultados del benchmark.	6
2.1.	A la izquierda, en la parte superior, comprobamos el número de hebras de <i>ab</i> . En la parte inferior, comprobamos el número de hebras de <i>apache2</i> . A la derecha la ejecución de <i>ab</i> .	7
3.1.	Ejecución de <i>ab</i> desde CentOS contra el servidor web de Ubuntu Server.	8
3.2.	Ejecución de <i>ab</i> desde Ubuntu Server 14.04 contra el servidor web de CentOS.	9
3.3.	Ejecución de <i>ab</i> desde Ubuntu Server 14.04 contra el servidor web de Windows Server.	10
4.1.	Haciendo click derecho en el plan seleccionamos <i>Hilos(Usuarios)</i> y marcamos <i>Grupos de Hilos</i> .	11
4.2.	Grupo de Hilos en Apache JMeter.	12
4.3.	Haciendo click derecho en el <i>Grupo de Hilos</i> escogemos <i>Añadir -> Muestreador -> Petición HTTP</i> .	13

4.4.	En el campo de <i>Nombre de Servidor o IP</i> , seleccionamos la IP del servidor al que vamos a hacer el benchmark.	13
4.5.	Hacemos click derecho en el <i>Plan de Pruebas</i> escogemos <i>Añadir ->Receptor ->Gráfico de Resultados</i>	14
4.6.	Tras pulsar en el botón de ejecutar y esperar obtenemos un gráfico como este.	14
4.7.	Resultados de Apache JMeter tras ejecutarlo desde el anfitrión contra el servidor web en CentOS.	15
4.8.	Resultados de Apache JMeter tras ejecutarlo desde el anfitrión contra el servidor web en Windows Server.	16
4.9.	Resultados de Apache JMeter tras ejecutarlo desde el anfitrión contra el servidor web en Ubuntu Server.	16
5.1.	Parámetros usados para ejecutar el benchmark propio.	21
5.2.	En la parte inferior del IDE <i>Qt Creator</i> vemos el resultado de ejecutar el benchmark con los parámetros previamente expuesto.	21

Índice de tablas

3.1.	Comparación de peticiones por segundo con ab para cada sistema operativo.	10
4.1.	Comparativa de servidores web con ab y JMeter	17

1. Seleccione un benchmark de Phoronix Suite, instálelo y ejecútelo, comente los resultados.

Tras instalar Phoronix Suite en Ubuntu Server 14.04 con el comando `sudo apt install`

```
(18-12-2016 15:19:38 dcr@ubuntu1) $sudo apt 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 se instalarán, 0 para eliminar y 83 no actualizados.
Se necesita descargar 0 B/424 kB de archivos.
Se utilizarán 2.166 kB de espacio de disco adicional después de esta operación.
Seleccionando el paquete phoronix-test-suite previamente no seleccionado.
(Leyendo la base de datos ... 89961 ficheros o directorios instalados actualmente.)
Preparing to unpack .../phoronix-test-suite_4.8.3-1_all.deb ...
Unpacking phoronix-test-suite (4.8.3-1) ...
Processing triggers for nan-db (2.6.7.1-1ubuntu1) ...
Processing triggers for hicolor-icon-theme (0.13-1) ...
Processing triggers for mime-support (3.54ubuntu1.1) ...
Configurando phoronix-test-suite (4.8.3-1) ...
(18-12-2016 15:19:50 dcr@ubuntu1) $_
```

Figura 1.1: Realizamos la instalación de *Phoronix Suite* con *apt* en Ubuntu Server 14.04.

`phoronix-test-suite` consultamos la lista de benchmarks disponibles ejecutando la orden `phoronix-test-suite list-available-tests`. Como hay una lista muy grande utilizo la herramienta “*grep*” para filtrar algún test que me interese, en concreto estoy interesado en buscar alguno de memoria.

```
(18-12-2016 15:20:16 dcr@ubuntu1) $phoronix-test-suite list-available-tests | grep Memory
pts/psstop      - PSSTOP Memory test      Processor
pts/ranspeed    - RANSPEED SMP             Memory
pts/stream      - Stream                   Memory
(18-12-2016 15:20:34 dcr@ubuntu1) $_
```

Figura 1.2: Listado de test de memoria en *Phoronix Suite*.

En mi caso, he decidido que voy a realizar el test denominado *pts/ramspeed*. Para ejecutar e instalar el benchmark uso el comando `phoronix-test-suite benchmark <nombre-test>`. [1]

```
(18-12-2016 15:20:16 dcr@ubuntu) $phoronix-test-suite list-available-tests | grep Memory
pts/psstop          - PSSTOP Memory test          Processor
pts/ramspeed        - RAMspeed SMP                Memory
pts/stream          - Stream                      Memory
(18-12-2016 15:20:34 dcr@ubuntu) $phoronix-test-suite benchmark pts/ramspeed

Phoronix Test Suite v4.8.3

To Install: pts/ramspeed-1.4.0

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

1 Test To Install
1 File To Download [0.00MB]
1MB Of Disk Space Is Needed

pts/ramspeed-1.4.0:
Test Installation 1 of 1
1 File Needed [0.00 MB / 1 Minute]
Downloading: ramsmp-3.5.0.tar.gz [0.00MB]
Estimated Download Time: in .....
Installation Size: 0.72 MB
Installing Test @ 15:21:20
```

Figura 1.3: Como es la primera vez que se realiza el test, se inicia la descarga del mismo.

```
1: Copy
2: Scale
3: Add
4: Triad
5: Average
6: Test All Options
Type: 5

1: Integer
2: Floating Point
3: Test All Options
Benchmark: 2

System Information

Hardware:
Processor: Intel Core i5-6198DU @ 2.40GHz (1 Core), Motherboard: Oracle VirtualBox v1.2, Chipset: In
tel 440FX - 82441FX PMC, Memory: 1024MB, Disk: 2 x 3GB VBox HDD, Graphics: InnoTek VirtualBox, Audio:
Intel 82801AA AC 97 Audio, Network: Intel 82540EM Gigabit

Software:
OS: Ubuntu 14.04, Kernel: 4.4.0-31-generic (x86_64), Compiler: GCC 4.8, File-System: ext4, Screen Re
solution: 800x600, System Layer: VirtualBox

Would you like to save these test results (Y/n): Y
Enter a name to save these results under: resultados
Enter a unique name to describe this test run / configuration: ranset1

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: VirtualBox testing on Ubuntu 14.04 via the Phoronix Test Suite.

New Description: Test RAM 1_
```

Figura 1.4: Escojo una opción del test, en mi caso, velocidad media (5. *Average*). y con datos en coma flotante (2. *Floating Point*). Podemos observar la información del sistema y escoger el nombre de los datos a guardar.

Tras aproximadamente los 6 minutos indicados como tiempo estimado obteniendo de media **8486.04 MB/s**.

Además, puesto que, como podemos observar, he escogido que los resultados sean compartidos en la web, podemos acceder desde Internet a los resultados del test (en caso de no compartirlo, dichos resultados también son guardados en la carpeta

Resultados Performance

←

→

↺

🏠

openbenchmarking.org/result/1612187-SO-RESULTADO06

★

🔊

🖥️

🔄

📄

👤

⋮

David

22:39

18/12/2016

IOzone

RAMSpeed SiMP

SciMark

GpuTest

FS-Mark

Recently Updated Tests

The Talos Principle

Stress-NG

Netperf

iPerf

Open Porous Media Glt

Other New & Recently Updated Tests

Advertisement

Currently Popular Results

Radeon R9 290 Regression Still

Netapp ESX Volume 4

testcorei7

cloud_z-bmt-was-01

bmt-was-01

newtest

Netapp ESX Volume 4

lacasa-xen-odroidxu4-cryptography

1612187-SO-RESULTADO06: Test RAM 1

resultados

OpenBenchmarking.org

Phoronix Test Suite 6.4.0

Intel Core i5-6198DU @ 2.40GHz (1 Core)

Processor

Oracle VirtualBox v1.2

Middleware

Intel 440FX- 82441FX PMC

Chipset

1024MB

Memory

2 x 9GB VBOX HDD

Disk

InnoTek VirtualBox

Graphics

Intel 82801AA AC 97 Audio

Audio

Intel 82540EM Gigabit

Network

Ubuntu 14.04

OS

4.4.0-31-generic (x86_64)

Kernel

GCC 4.8

Compiler

x86_64

File-System

800x600

Screen Resolution

VirtualBox

System Layer

Resultados Performance

--build=x86_64-linux-gnu --disable-browser-plugin

--disable-ibmudfap --disable-werror

--enable-checking-release --enable-dcrole-gnu

--enable-gnu-unique-object --enable-gtk-cairo

--enable-java-awt-gtk --enable-java-home

--enable-languages=c-c++ java.gcc fortran obj-c-c++

--enable-libstdc++-debug --enable-libstdc++-time=yes

--enable-multifarch --enable-nls --enable-objc-gc

--enable-plugin --enable-shared --enable-threads=posix

--host=x86_64-linux-gnu --target=x86_64-linux-gnu

--with-abi=m64 --with-arch=32=i686

--with-arch-directory=amd64

Resultados Performance

openbenchmarking.org/result/1612187-SO-RESULTADO06

1612187-SO-RESULTADO06 - Results Uploaded On 18 Dec 2016

Download as CSV XML PDF

Via the OpenBenchmarking.org IDs you can analyze the results from a Phoronix Test Suite client. You can also use the export options below for external analysis.

RAMspeed SMP

RAMspeed SMP v3.5.0
Type: Average - Benchmark: Floating Point

ptsli

OpenBenchmarking.org

MB/s, More Is Better

ramtest1

SE +/- 0.00

8486.04

2000 4000 6000 8000 10000

Phoronix Test Suite 6.4.0

User Comments

Post A Comment

Pregúntame cualquier cosa

22:41 18/12/2016

6

- 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?

La opción -n indica el número de peticiones que se han de realizar (si no lo indicamos por defecto es 1).

The screenshot shows a terminal window with the following content:

```

[11-12-2016 11:59:09 dcr@centos:~$ cat /dev/urandom | base64 | nc 192.168.56.162 80
[12-12-2016 11:58:24 dcr@centos:~$ ps -eT | grep -w ab -c
1
[12-12-2016 11:59:15 dcr@centos:~$ ps -eT | grep -w ab -c
1
[12-12-2016 11:59:24 dcr@centos:~$ ps -eT | grep -w ab -c
1
[12-12-2016 12:00:07 dcr@centos:~$ ]

[12-12-2016 11:58:58 dcr@ubuntu:~$ ps -eT | grep apache2 -c
11
[12-12-2016 11:59:00 dcr@ubuntu:~$ ps -eT | grep apache2 -c
47
[12-12-2016 11:59:32 dcr@ubuntu:~$ ps -eT | grep apache2 -c
59
[12-12-2016 11:59:33 dcr@ubuntu:~$ ps -eT | grep apache2 -c
75
[12-12-2016 11:59:36 dcr@ubuntu:~$ ps -eT | grep apache2 -c
75
[12-12-2016 11:59:39 dcr@ubuntu:~$ ps -eT | grep apache2 -c
73
[12-12-2016 11:59:42 dcr@ubuntu:~$ ps -eT | grep apache2 -c
78
[12-12-2016 11:59:48 dcr@ubuntu:~$ ]
  
```

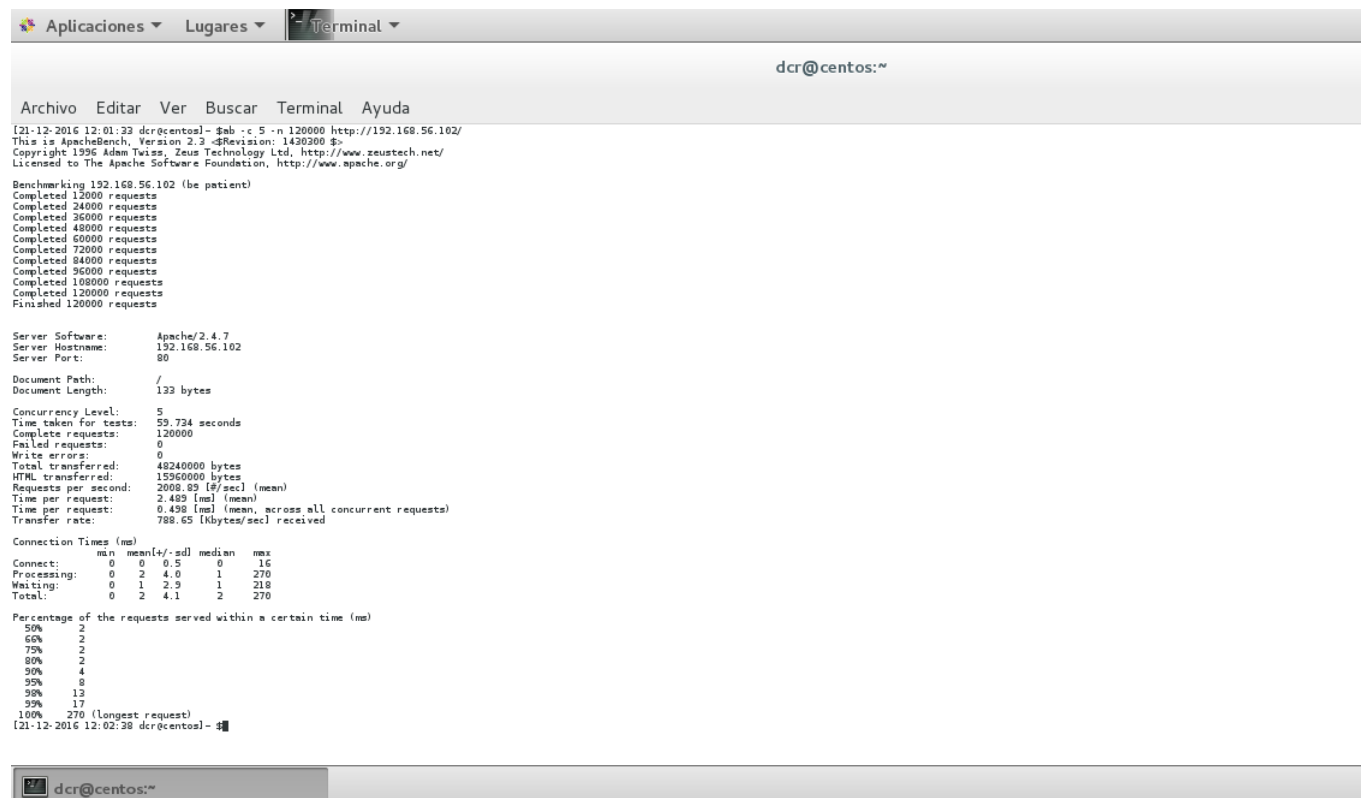
The terminal also shows the output of the Apache logs, which are being redirected to a file named 'logs'. The logs show a flood of requests from 192.168.56.102, with the error log indicating 'File does not exist: /usr/share/apache2/htdocs/index.html'.

7

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).

Todas las pruebas han sido realizadas con 120000 peticiones y un nivel de concurrencia de 5. Para que las pruebas sean válidas utilizamos el siguiente código HTML (Hypertext Markup Language).

```
<html>
<head><title>TEST AB</title></head>
<body>Esto es una página de prueba para hacer el benchmarking web de ISE.</body>
</html>
```



```
dcrc@centos:~$ ab -c 5 -n 120000 http://192.168.56.102/
This is ApacheBench, Version 2.3 <Revision: 1430300>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.56.102 (be patient)
Completed 12000 requests
Completed 24000 requests
Completed 36000 requests
Completed 48000 requests
Completed 60000 requests
Completed 72000 requests
Completed 84000 requests
Completed 96000 requests
Completed 108000 requests
Completed 120000 requests
Finished 120000 requests


Server Software: Apache/2.4.7
Server Hostname: 192.168.56.102
Server Port: 80

Document Path: /
Document Length: 133 bytes

Concurrency Level: 5
Time taken for tests: 59.734 seconds
Complete requests: 120000
Failed requests: 0
Write errors: 0
Total transferred: 482400000 bytes
HTML transferred: 156000000 bytes
Requests per second: 2008.99 [#/sec] (mean)
Time per request: 2.489 [ms] (mean)
Time per request: 0.498 [ms] (mean, across all concurrent requests)
Transfer rate: 786.65 [Kbytes/sec] received

Connection Times (ms)
  min  mean[+/-sd] median max
Connect: 0  0  0.5  0  16
Processing: 0  2  4.0  1  270
Waiting: 0  1  2.9  1  218
Total: 0  2  4.1  2  270

Percentage of the requests served within a certain time (ms)
 50%  2
 66%  2
 75%  2
 80%  2
 90%  4
 95%  8
 98%  13
 99%  17
100% 270 (longest request)
[21-12-2016 12:02:38 dcr@centos]-
```

Figura 3.1: Ejecución de *ab* desde CentOS contra el servidor web de Ubuntu Server.


```

[21-12-2016 12:03:09 dcr@ubuntu1html $ab -c 5 -n 120000 http://192.168.56.101/
This is ApacheBench, Version 2.3 <$Revision: 1528965 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.56.101 (be patient)
Completed 12000 requests
Completed 24000 requests
Completed 36000 requests
Completed 48000 requests
Completed 60000 requests
Completed 72000 requests
Completed 84000 requests
Completed 96000 requests
Completed 108000 requests
Completed 120000 requests
Finished 120000 requests


Server Software:      Apache/2.4.6
Server Hostname:      192.168.56.101
Server Port:          80

Document Path:        /
Document Length:      126 bytes

Concurrency Level:    5
Time taken for tests:  76.474 seconds
Complete requests:    120000
Failed requests:       0
Total transferred:    47760000 bytes
HTML transferred:     15120000 bytes
Requests per second:  1569.17 [#/sec] (mean)
Time per request:     3.186 [ms] (mean)
Time per request:     0.637 [ms] (mean, across all concurrent requests)
Transfer rate:        609.89 [Kbytes/sec] received


Connection Times (ms)
              min    mean[+/-sd] median    max
Connect:        0      0   0.2      0      9
Processing:      1      3  20.6      3    6517
Waiting:         0      3  19.8      3    6516
Total:           1      3  20.6      3    6517


Percentage of the requests served within a certain time (ms)
 50%      3
 66%      3
 75%      3
 80%      3
 90%      3
 95%      4
 98%      4
 99%      7
100%    6517 (longest request)
[21-12-2016 12:05:03 dcr@ubuntu1html $~_

```

Figura 3.2: Ejecución de *ab* desde Ubuntu Server 14.04 contra el servidor web de CentOS.

```

[21-12-2016 12:11:58 dcr@ubuntu1html $ab -c 5 -n 120000 http://192.168.56.103/
This is ApacheBench, Version 2.3 <$Revision: 1528965 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.56.103 (be patient)
Completed 12000 requests
Completed 24000 requests
Completed 36000 requests
Completed 48000 requests
Completed 60000 requests
Completed 72000 requests
Completed 84000 requests
Completed 96000 requests
Completed 108000 requests
Completed 120000 requests
Finished 120000 requests


Server Software:      Microsoft-IIS/8.5
Server Hostname:      192.168.56.103
Server Port:          80

Document Path:        /
Document Length:      127 bytes

Concurrency Level:     5
Time taken for tests:  35.113 seconds
Complete requests:     120000
Failed requests:        0
Total transferred:      44400000 bytes
HTML transferred:       15240000 bytes
Requests per second:    3417.53 [#/sec] (mean)
Time per request:       1.463 [ms] (mean)
Time per request:       0.293 [ms] (mean, across all concurrent requests)
Transfer rate:          1234.85 [Kbytes/sec] received


Connection Times (ms)
              min      mean[+/-sd] median   max
Connect:        0        0   0.2      0     10
Processing:      0        1   0.4      1     11
Waiting:         0        1   0.4      1     11
Total:          1        1   0.5      1     11


Percentage of the requests served within a certain time (ms)
 50%    1
 66%    1
 75%    1
 80%    2
 90%    2
 95%    2
 98%    2
 99%    3
100%   11 (longest request)
[21-12-2016 12:12:42 dcr@ubuntu1html $

```

Figura 3.3: Ejecución de *ab* desde Ubuntu Server 14.04 contra el servidor web de Windows Server.

Para comparar todos los servidores utilizamos la métrica de peticiones por segundo.

Ubuntu Server 14.04	CentOS 7	Windows Server 2012 R2
2008,89 #/segundo	1569,17 #/segundo	3417,53 #/segundo

Tabla 3.1: Comparación de peticiones por segundo con *ab* para cada sistema operativo.

Podemos observar que con diferencia Windows Server ha ofrecido los mejores resultados y que el siguiente mejor ha sido Ubuntu Server y, por último, CentOS que tiene más overhead que la otra distribución de Linux, entre otros, por el consumo que tiene la

interfaz gráfica del sistema operativo.

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 meter, haga el experimento usando sus máquinas virtuales ¿coincide con los resultados de ab?

Para instalarlo simplemente descargamos el zip y ejecutamos el archivo *jmeter.bat* en la carpeta bin (es necesario tener instalado Java en el sistema). Tras leer el tutorial, seguimos los siguientes pasos para preparar el test.

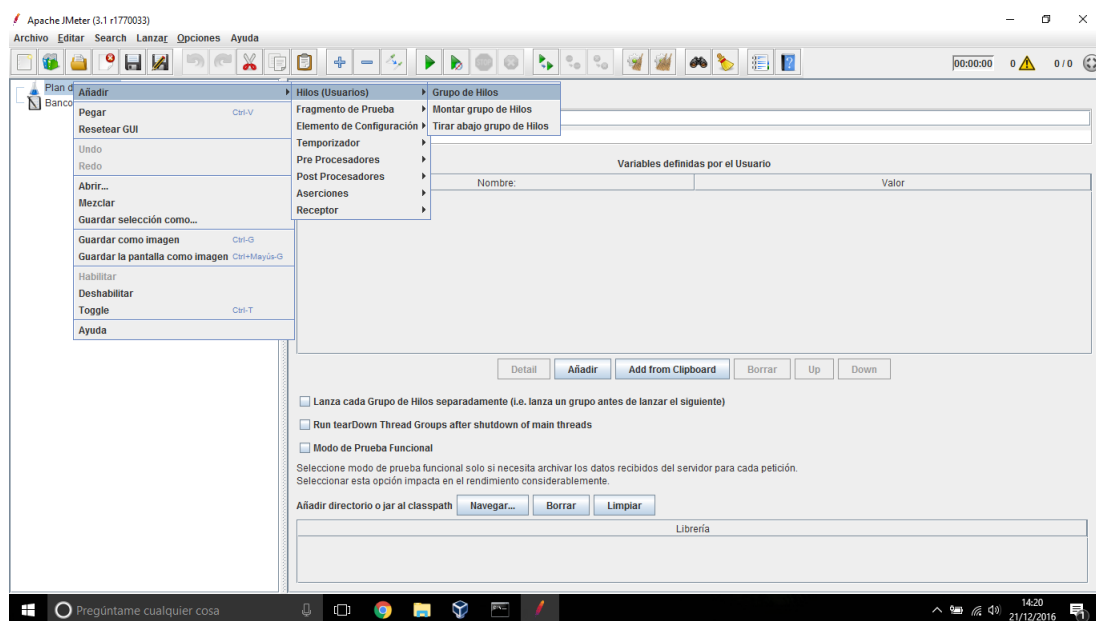


Figura 4.1: Haciendo click derecho en el plan seleccionamos *Hilos(Usuarios)* y marcamos *Grupos de Hilos*.

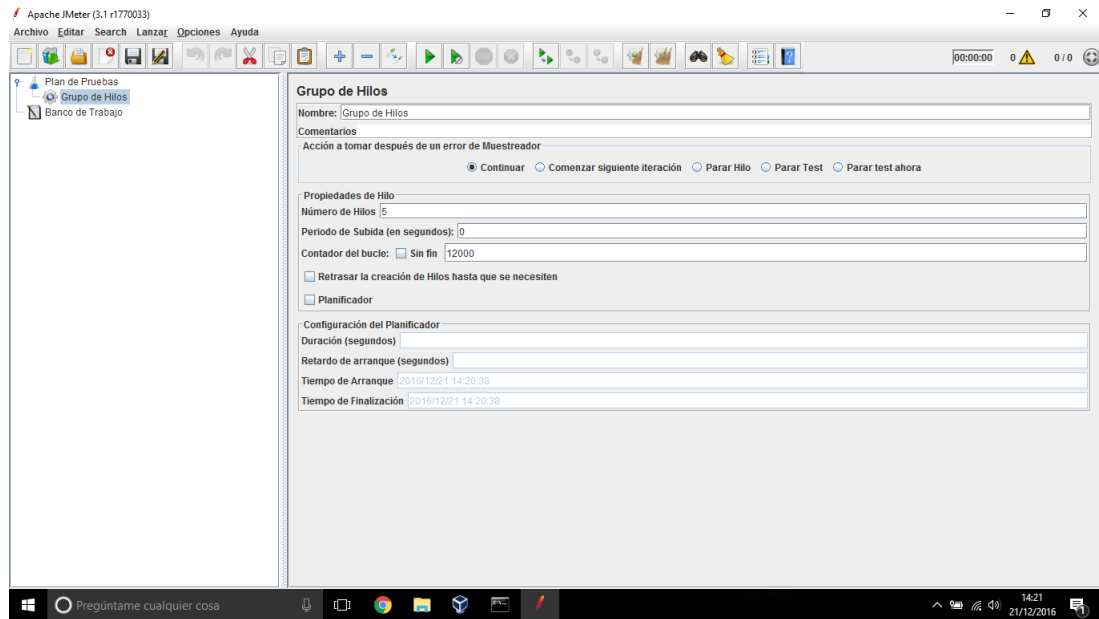


Figura 4.2: Indicamos el número de hebras que vamos a usar en *Número de Hilos* el tiempo en el que van entrando *Periodo de Subida (en segundos)* que dejamos a 0 (para que se inicien todos a la vez) y en el *Contador de bucle* ponemos el número de peticiones que ha de hacer cada hebra (al final lo dejé a 24000) para obtener una cantidad de muestras similar a las de ab.

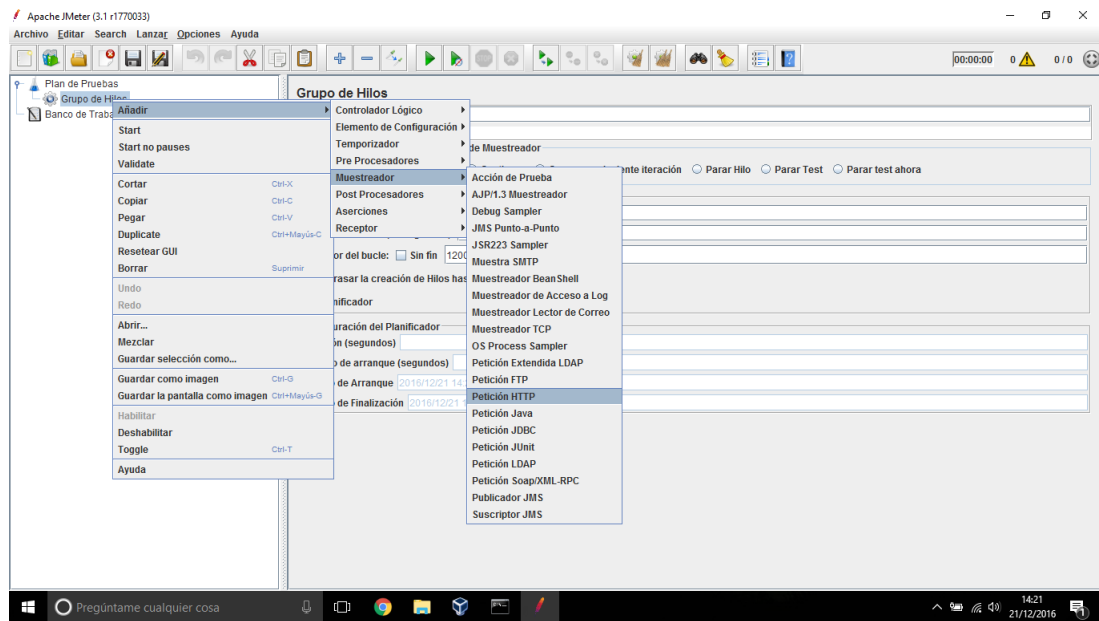


Figura 4.3: Haciendo click derecho en el *Grupo de Hilos* escogemos *Añadir ->Muestreador ->Petición HTTP*.

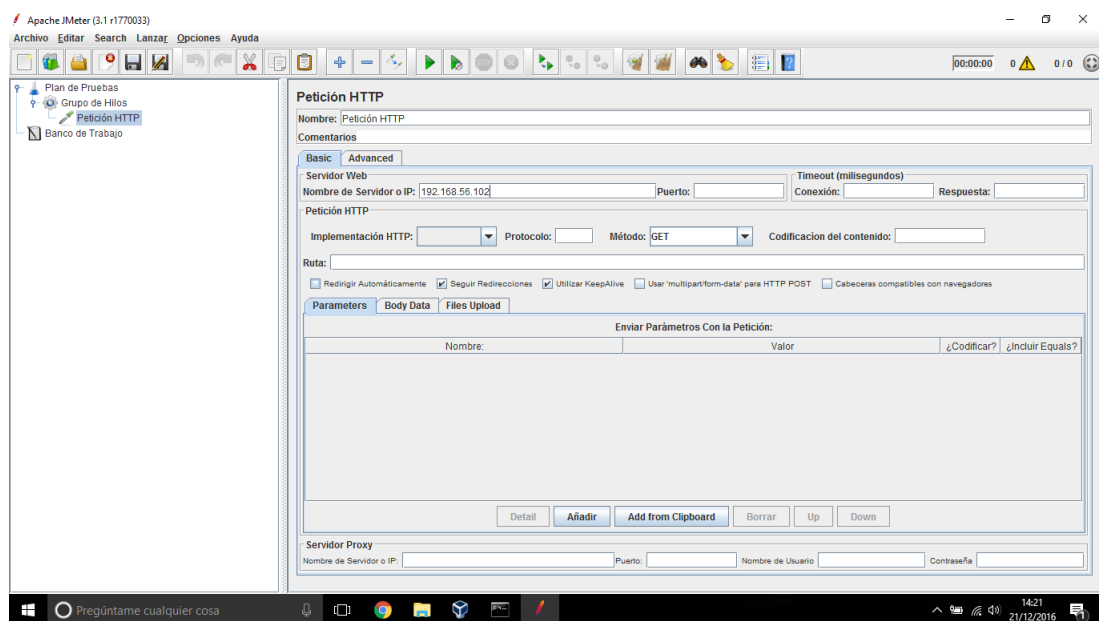


Figura 4.4: En el campo de *Nombre de Servidor o IP*. seleccionamos la IP del servidor al que vamos a hacer el benchmark.

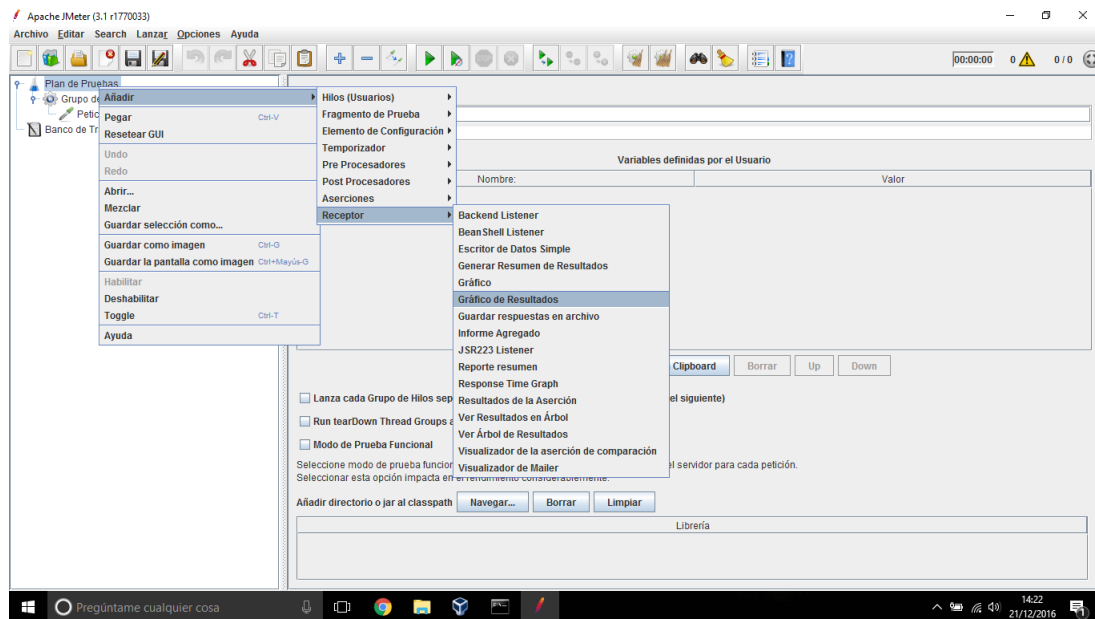


Figura 4.5: Hacemos click derecho en el *Plan de Pruebas* escogemos *Añadir ->Receptor ->Gráfico de Resultados*.

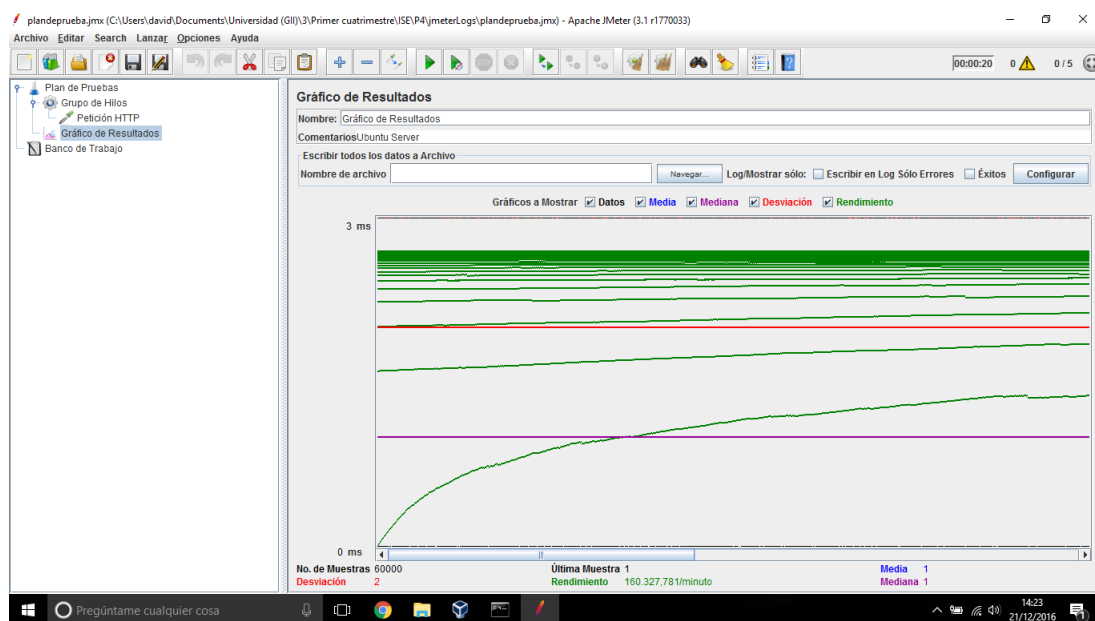


Figura 4.6: Tras pulsar en el botón de ejecutar y esperar obtenemos un gráfico como este.

Como podemos observar, en este ejemplo he realizado el benchmark sobre Ubuntu Server 14.04 indicando su dirección IP (192.168.56.102) y observamos un *Rendimiento* de

160.327,781/minuto. Para que las muestras sean más o menos similares aumentamos el contador del bucle a 24000. Tras realizarlo sobre las 3 máquinas obtenemos los siguientes resultados:

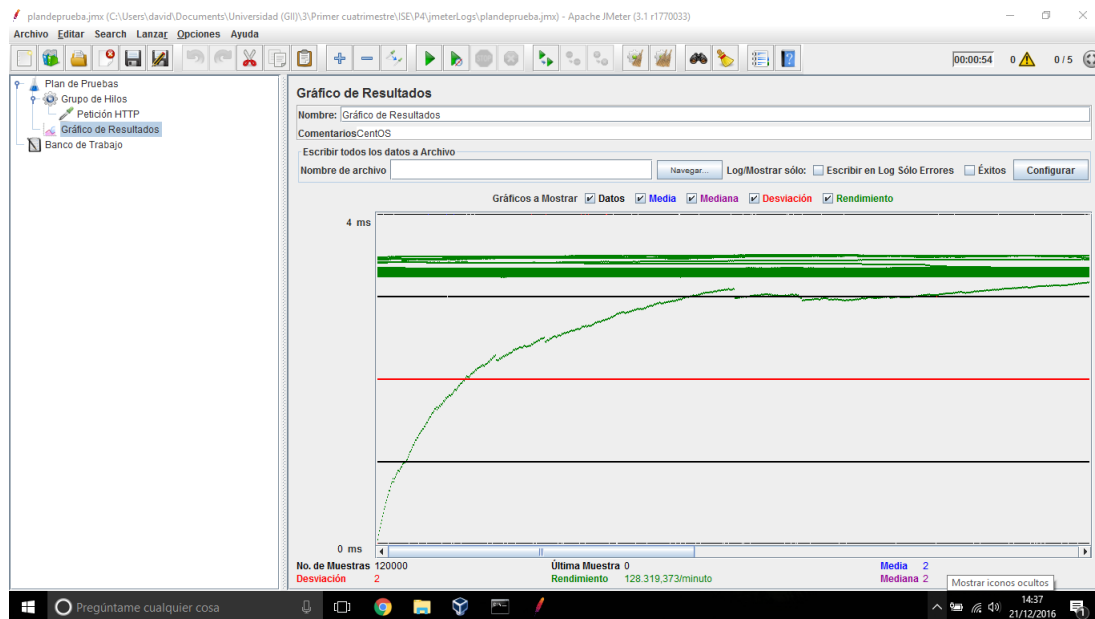


Figura 4.7: Resultados de Apache JMeter tras ejecutarlo desde el anfitrión contra el servidor web en CentOS.

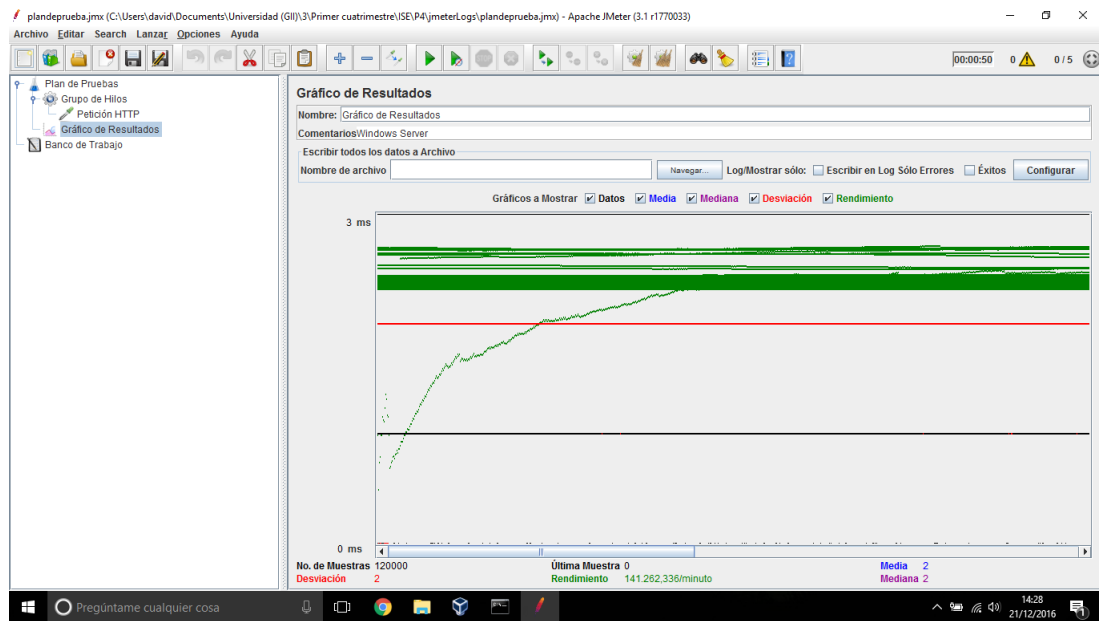


Figura 4.8: Resultados de Apache JMeter tras ejecutarlo desde el anfitrión contra el servidor web en Windows Server.

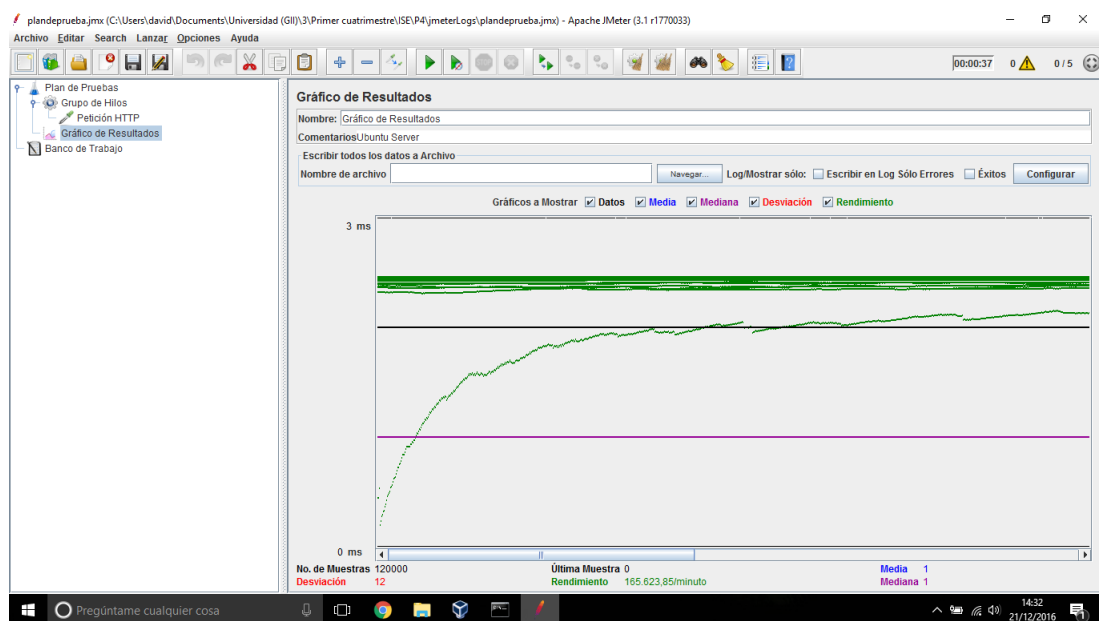


Figura 4.9: Resultados de Apache JMeter tras ejecutarlo desde el anfitrión contra el servidor web en Ubuntu Server.

Tras pasar las peticiones por segundo a peticiones por minuto de los resultados de *ab* y

el rendimiento de *Apache JMeter* (que se mide en peticiones por minuto también).

	Ubuntu Server 14.04	CentOS 7	Windows Server 2012 R2
ab	120.533,4 #/minuto	94.150,20 #/minuto	205.051,80 #/minuto
JMeter	165.623,85 #/minuto	128.319,373 #/minuto	141.266,63 #/minuto

Tabla 4.1: Comparativa de servidores web con ab y JMeter.

Podemos ver que con este test se determina que el más lento de todos es CentOS pero en este caso Ubuntu Server es más rápido que Windows Server. Es posible que el hecho de que al usar ab necesitase tener abierta otra máquina virtual añadiese algún overhead a la hora de realizar los benchmarks.

5. Programe un benchmark usando el lenguaje que desee.

El objetivo del benchmark es medir la ejecución con múltiples hebras de una CPU, para ello vamos a, en cada repetición, crear un grupo de hebras ejecutar el mismo algoritmos en todas las hebras, unirlos y medir el tiempo transcurrido. La métrica a usar será el tiempo de ejecución (medido en segundos) medio de todos los intervalos. Como el benchmark lo voy a programar con C++11 utilizaré la librería *thread* [3] para crear hebras concurrentes y la librería *chrono* [4] para realizar mediciones de tiempo lo más precisas posibles. Además, el programa permitirá pasar como parámetro si usar enteros o números en coma flotante, el número de hebras, las repeticiones, y si queremos usar un algoritmo cuadrático (ordenación por burbuja), ordenación $n \log n$ con `std::sort` o un algoritmo lineal que realiza operaciones matemáticas sobre cada número. Es importante que se compile sin optimizaciones el código para obtener resultados válidos. El código del benchmark sería el siguiente:

```
#include <algorithm>
#include <chrono>
#include <iostream>
#include <stdexcept>
#include <random>
#include <thread>
#include <vector>

// Generamos números en coma flotante aleatorios (No cuenta para el tiempo del benchmark).
void generateRandom(auto& v, unsigned size) {
    std::random_device rd;
    std::mt19937 mt(rd());
    std::uniform_real_distribution<double> dist;

    v.resize(size);
    for (auto& d: v)
```

```

d = dist(mt);
}

// Opción para complejidad cuadrática
void sortBubble(auto& v) {
for (auto i = 0u; i < v.size(); ++i)
for (auto j = 0u; j < v.size() - i; ++j)
if (v[j] > v[j+1])
std::swap(v[j], v[j+1]);
}

void sortBubbleInt(std::vector<int>& v) { std::vector<int> a;
std::copy(std::begin(v), std::end(v), std::back_inserter(a));
sortBubble(a); }
void sortBubbleFloat(std::vector<double>& v) { std::vector<double> a;
std::copy(std::begin(v), std::end(v), std::back_inserter(a));
sortBubble(a); }

// Opción para complejidad  $O(n \log n)$ 
void sortLog(auto v) {
std::sort(std::begin(v), std::end(v));
}

void sortLogInt(const std::vector<int>& v) { std::vector<int> a;
std::copy(std::begin(v), std::end(v), std::back_inserter(a));
sortLog(a); }
void sortLogFloat(const std::vector<double>& v) { std::vector<double> a;
std::copy(std::begin(v), std::end(v), std::back_inserter(a));
sortLog(a); }

// Opción para complejidad  $O(n)$ 
void linear(auto v) {
for (auto& d: v)
d = (d + 2) / std::sqrt(d);
}

void sortLinearInt(const std::vector<int>& v) { std::vector<int> a;
std::copy(std::begin(v), std::end(v), std::back_inserter(a));
linear(a); }
void sortLinearFloat(const std::vector<double>& v) { std::vector<double> a;
std::copy(std::begin(v), std::end(v), std::back_inserter(a));
linear(a); }

double averageTime(const std::vector<double>& times) {

```

```

double media = 0.0;
for (auto d: times)
media += d;
media /= times.size();

return media;
}

int main(int argc, char *argv[])
{
unsigned size, repetitions, concurrency, mode;
bool isFloat = false;
if (argc != 6) {
std::cerr << "Uso correcto: benchmark <tam_vector> <número de repeticiones> <float o int>
std::cerr << "<modo: 0 (cuadrático)|1(logarítmico)|2 lineal>\n";
return -1;
}
try {
size = std::stoi(argv[1]);
repetitions = std::stoi(argv[2]);
std::string myType = argv[3];

if (myType == "float")
isFloat = true;
else if (myType == "int")
isFloat = false;
else {
std::cerr << "Uso correcto: benchmark <tam_vector> <número de repeticiones> <float o int>
std::cerr << "<modo: 0 (cuadrático)|1(logarítmico)|2 lineal>\n";
std::cerr << "El tipo sólo puede ser float o int (en minúscula).\n";
return -1;
}
concurrency = std::stoi(argv[4]);

mode = std::stoi(argv[5]);
if (mode > 2) {
std::cerr << "Uso correcto: benchmark <tam_vector> <número de repeticiones> <float o int>
std::cerr << "<modo: 0 (cuadrático)|1(logarítmico)|2 lineal>\n";
return -1;
}

} catch (const std::invalid_argument&) {
std::cerr << "Uso correcto: benchmark <tam_vector> <número de repeticiones> <float o int>
return -1;
}

```

```

}

std::vector<std::thread> threads;
std::vector<double> repeatTimes;

std::vector<double> v;
std::vector<int> vi;

if (isFloat)
    genereteRandom(v, size);
else
    genereteRandom(vi, size);
for (auto i = 0u; i < repetitions; ++i) {
    auto trAntes = std::chrono::high_resolution_clock::now();
    for (auto j = 0u; j < concurrency; ++j)
        if (isFloat) switch (mode) {
            case 0: threads.emplace_back(&sortBubbleFloat, std::ref(v)); break;
            case 1: threads.emplace_back(&sortLogFloat, std::ref(v)); break;
            case 2: threads.emplace_back(&sortLinearFloat, std::ref(v)); break;
        } else switch (mode) {
            case 0: threads.emplace_back(sortBubbleInt, std::ref(vi)); break;
            case 1: threads.emplace_back(sortLogInt, std::ref(vi)); break;
            case 2: threads.emplace_back(sortLinearInt, std::ref(vi)); break;
        }
    for (auto& thread: threads)
        thread.join();
    auto trDespues = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> time = trDespues - trAntes;
    repeatTimes.push_back(time.count());
    threads.clear();
}
double mediaConcurrente = averageTime(repeatTimes);

std::cout << "El tiempo medio de ejecución concurrente de una repetición con " <<
concurrency << " hebras es " << mediaConcurrente << " segundos.\n";

}

```

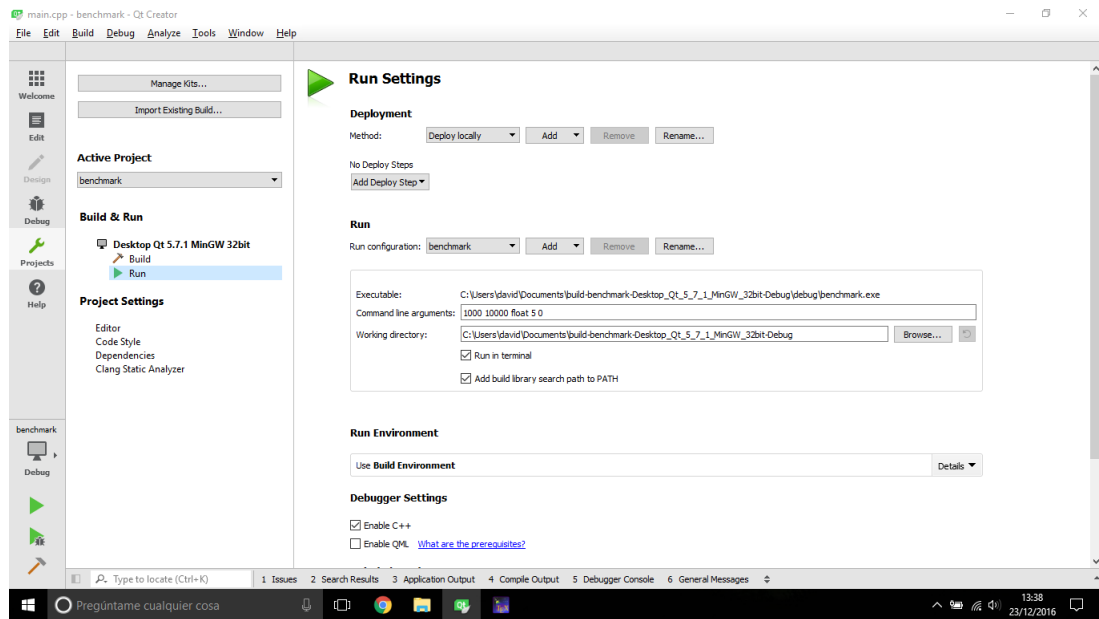


Figura 5.1: Parámetros usados para ejecutar el benchmark propio.

Tras ejecutarlo en mi máquina anfitriona con los siguientes parámetros que se pueden ver en la figura obtengo los resultados que se ven en la siguiente captura de pantalla.

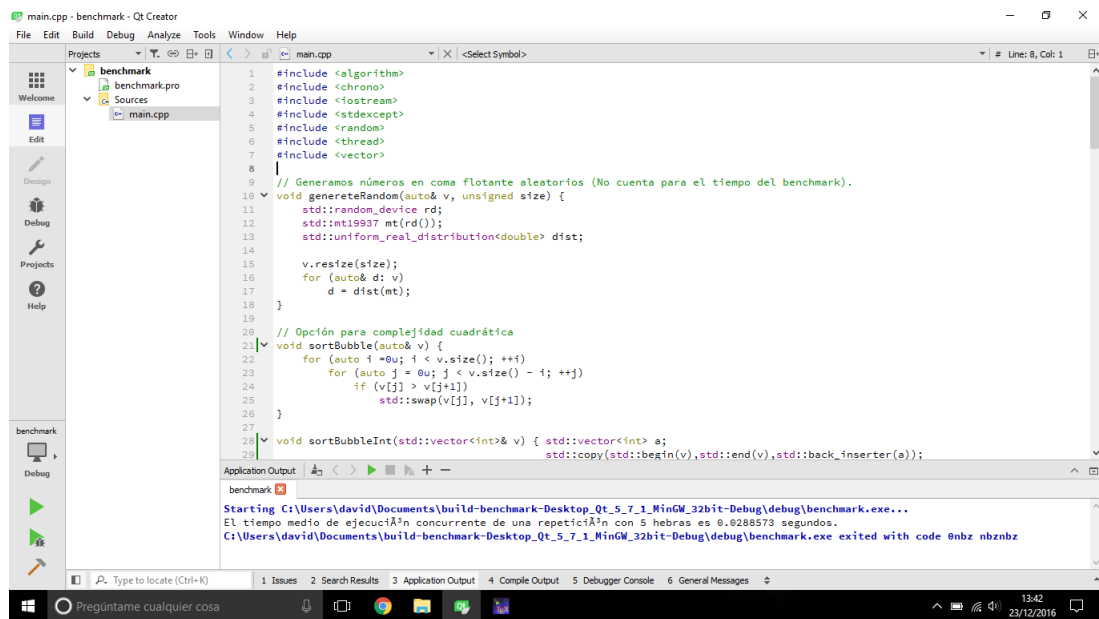


Figura 5.2: En la parte inferior del IDE *Qt Creator* vemos el resultado de ejecutar el benchmark con los parámetros previamente expuesto.

Como podemos observar el tiempo media que mi máquina anfitriona, con un Intel Core i5-6198DU tarda de media aproximadamente 0,03 segundos en que todas las hebras ejecuten una vez el algoritmo de ordenación burbuja sobre un vector de números en coma flotante de tamaño 1000.

Referencias

- [1] “Página de manual para el comando “phoronix-test-suite” en Ubuntu Server 14.04.”
- [2] “Página de manual para el comando “ab” en Ubuntu Server 14.04.”
- [3] “C++ Reference: Class Thread (since c++11).” <http://www.cplusplus.com/reference/thread/thread/>, consultado el 20 de Diciembre de 2016.
- [4] “C++ Reference: High Resolution Clock Now (since c++11).” http://www.cplusplus.com/reference/chrono/high_resolution_clock/now/, consultado el 20 de Diciembre de 2016.