

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

Memoria Práctica 4



Iván Rodríguez Millán

23 de diciembre de 2016

Índice

1 Seleccione, instale y execute uno, comente los resultados. Atención: no es lo mismo un benchmark que una suite, instale un benchmark.	4
1.1 Respuesta :	4
2 Para Apache Benchmark, de los parámetros que le podemos pasar al comando ab, ¿Qué significa -c 5? ¿Y -n 100? Monitorice la ejecución de ab contra alguna máquina (cualquiera)¿Cuántas "tareas" crea ab en el cliente?	10
2.1 Respuesta :	10
3 Ejecute ab contra las tres máquinas virtuales (desde el SO anfitrión a las máquinas virtuales de la red local) una a una (arrancadas por separado). ¿Cuál es la que proporciona mejores resultados? Muestre y coméntelos. (Use como máquina de referencia Ubuntu Server para la comparativa).	13
3.1 Respuesta :	13
4 Instale y siga el tutorial en la página de JMeter 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?	20
4.1 Respuesta :	20
5 Programe un benchmark usando el lenguaje que deseé. El benchmark debe incluir: Objetivo del benchmark, métricas (unidades, variables, puntuaciones, etc.), instrucciones para su uso y ejemplo de uso analizando los resultados.	35
5.1 Respuesta :	35
5.2 Objetivo del Benchmark:	35
5.3 Métricas :	35
5.4 Instrucciones para su uso :	35
5.5 Ejemplo de uso analizando los resultados :	51

Índice de figuras

1.1. Instalación de Phoronix-test-suite.	4
1.2. Listando benchmark disponibles para instalar en nuestro equipo.	5
1.3. Instalando benchmark.	6
1.4. Fin de la instalación del benchmark.	6
1.5. Listando información del test elegido.	7
1.6. Ejecución del benchmark elegido.	7
1.7. Fin de la ejecución del benchmark elegido.	8
1.8. Gráfica de OpenBenchmarking.	8
1.9. Datos resultantes del benchmark.	9
1.10. Datos de las máquinas a comparar.	9
1.11. Gráfica de la comparativa.	9

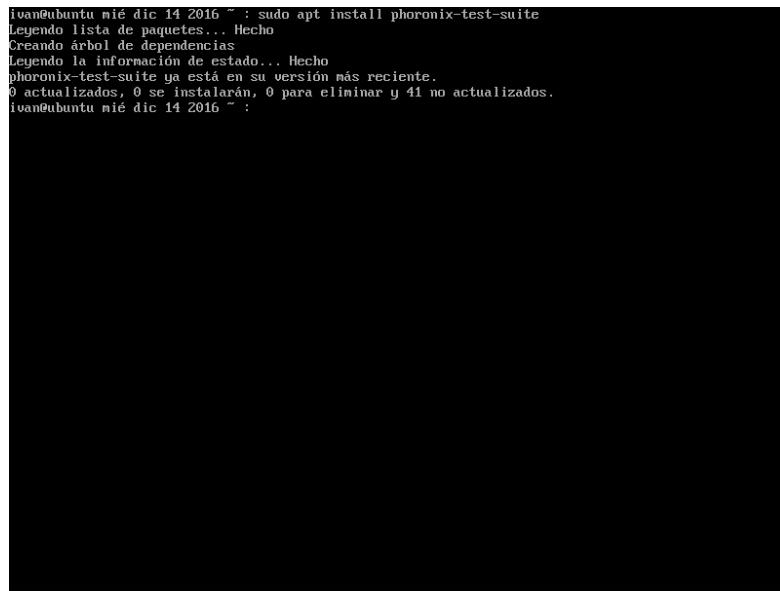
2.1.	Salida de la ejecución del comando ab.	11
2.2.	Mostrando la cantidad de procesos creados en el cliente con el lanzamiento del comando ab contra otra máquina.	12
3.1.	Nuevo index.html en Ubuntu Server.	13
3.2.	Nuevo index.html en CentOS.	14
3.3.	Nuevo index.html en Windows Server.	15
3.4.	Conexión desde el anfitrión a las tres páginas de Apache, Httpd e IIS (Ubuntu, CentOS 7 y Windows Server).	16
3.5.	Lanzamiento del comando ab contra Ubuntu Server.	17
3.6.	Lanzamiento del comando ab contra CentOS 7.	18
3.7.	Lanzamiento del comando ab contra Windows Server 2012.	19
4.1.	Ejecución de JMeter desde la terminal.	20
4.2.	Añadiendo grupo de usuarios a nuestro test.	21
4.3.	Añadiendo valores por defecto para petición HTTP.	22
4.4.	Añadiendo el tipo de peticiones HTTP.	23
4.5.	Añadiendo el gráfico de resultados.	23
4.6.	Finalización del lanzamiento del test.	24
4.7.	Ejecución de JMeter desde línea de comandos para la obtención del index.html.	25
4.8.	Inicio Apache JMeter Dashboard.	25
4.9.	Estadísticas de la ejecución del test.	26
4.10.	Tiempo total empleado para la generación de solicitudes.	26
4.11.	Distribución del tiempo de respuesta en milisegundos.	27
4.12.	Media de tiempo de conexión en milisegundos.	27
4.13.	Media de latencia en respuestas en milisegundos.	28
4.14.	Creación de grupo de hilos y valores por defecto para Windows Server 2012.	29
4.15.	Añadiendo solicitud HTTP para Windows Server 2012.	29
4.16.	Añadiendo gráfica para Windows Server 2012.	30
4.17.	Añadiendo reporte de datos para Windows Server 2012.	31
4.18.	Creación de grupo de hilos y valores por defecto para CentOS 7.	32
4.19.	Añadiendo solicitud HTTP para CentOS 7.	32
4.20.	Gráfica de los resultados del benchmark creado y ejecutado contra CentOS 7.	33
4.21.	Datos del resumen con respecto a la ejecución del benchmark contra CentOS 7.	34
5.1.	Gráfica de los resultados de ejecución de Fibonacci calculando desde el 0 hasta el 40.	51
5.2.	Gráfica de los resultados de ejecución de QuickSort ordenando un vector desde 2 componentes hasta 20.	52

Índice de tablas

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

1.1. Respuesta :

En primer lugar instalamos phoronix-test-suite: [1]



```
ivan@ubuntu mié dic 14 2016 ~ : sudo apt install phoronix-test-suite
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
phoronix-test-suite ya está en su versión más reciente.
0 actualizados, 0 se instalarán, 0 para eliminar y 41 no actualizados.
ivan@ubuntu mié dic 14 2016 ~ :
```

Figura 1.1: Instalación de Phoronix-test-suite.

En segundo lugar listaremos los benchmark disponibles para instalar en nuestro equipo. Lo que hacemos es llevarnos la salida a un fichero, ya que la salida es muy larga, y nos será más cómodo elegirlo desplazándonos desde el fichero. Para listar los benchmark disponibles necesitamos el siguiente comando:

- phoronix-test-suite list-available-tests

```
ivan@ubuntu:~$ phoronix-test-suite list-available-tests > lista_tests.txt  
ivan@ubuntu:~$
```

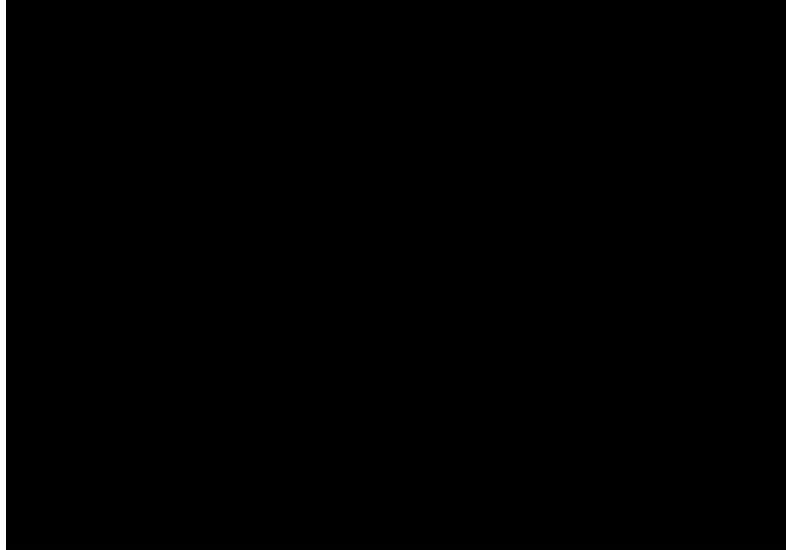


Figura 1.2: Listando benchmark disponibles para instalar en nuestro equipo.

En nuestro caso elegimos pts/build-apache, que consiste en medir el tiempo que tarda en compilarse un programa http apache(Timed http apache compilation). Para instalar un benchmark necesitamos el siguiente comando:

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

```

ivan@ubuntu mié dic 14 2016 ~ : phoronix-test-suite install pts/build-apache
The following dependencies are needed and will be installed:
- build-essential
- autoconf
- perl
- perl-base
- perl-modules
- libssl-perl
- libperl-dev
- libpcre3-dev
- mesa-utils
- unzip
This process may take several minutes.

```

Figura 1.3: Instalando benchmark.

```

Configurando libfile-which-perl (1.09-1) ...
Configurando libalien-sdl-perl (1.440-4) ...
Configurando libfile-fcntllock-perl (0.14-2build1) ...
Configurando libpcre3-dev:amd64 (1:8.31-2ubuntu2.3) ...
Configurando libperl5.18 (5.18.2-2ubuntu1.1) ...
Configurando libperl-dev (5.18.2-2ubuntu1.1) ...
Configurando libtie-simple-perl (1.03-1) ...
Configurando libssl-perl (2.540-5) ...
Configurando nmpages-dev (3.54-1ubuntu1) ...
Configurando musescore-soundfont-gm (1.3+dfsg-1) ...
Configurando unzip (6.0-9ubuntu1.5) ...
Configurando mesa-utils (8.1.0-2) ...
Procesando disparadores para libc-bin (2.19-0ubuntu6.9) ...

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: ..... [0.78MB]
  Estimated Download Time: 1m ..... .
  Downloading: apr-1.5.0.tar.bz2 [0.66MB]
  Estimated Download Time: 1m ..... .
  Installing Test @ 11:13:15

ivan@ubuntu mié dic 14 2016 ~ :

```

Figura 1.4: Fin de la instalación del benchmark.

A continuación se muestra con el siguiente comando información de cualquier paquete, en nuestro caso del elegido anteriormente:

- `phoronix-test-suite info <nombre-test>`

```

ivan@ubuntu mié dic 14 2016 ~ : phoronix-test-suite info pts/build-apache

Phoronix Test Suite v4.8.3
Timed Apache Compilation 2.4.7

Run Identifier: pts/build-apache-1.5.1
Profile Version: 1.5.1
Maintainer: Michael Larabel
Test Type: Processor
Software Type: Utility
License Type: Free
Test Status: Verified
Project Web-Site: http://www.apache.org/
Estimated Run-Time: 403 Seconds
Download Size: 6.21 MB

Description: This test times how long it takes to build the Apache HTTP Server.

Test Installed: Yes
Last Run: 2016-12-14
Latest Run-Time: 6 Minutes, 43 Seconds
Times Run: 1

Software Dependencies:
- Compiler / Development Libraries

ivan@ubuntu mié dic 14 2016 ~ :

```

Figura 1.5: Listando información del test elegido.

Para finalizar, ejecutamos el benchmark con el siguiente comando y analizamos los resultados:

- `phoronix-test-suite benchmark <nombre-test>`

```

ivan@ubuntu mié dic 14 2016 ~ : phoronix-test-suite benchmark pts/build-apache

Phoronix Test Suite v4.8.3

    Installed: pts/build-apache-1.5.1

System Information

Hardware:
Processor: Intel Core i5-5257U @ 2.70GHz (1 Core), Motherboard: Oracle VirtualBox v1.2, Chipset: Intel 440FX-82441FX PMC, Memory: 2048MB, Disk: 2 x 9GB VBOX HDD, Graphics: InnoTek VirtualBox, Audio: Intel 82801AA AC 97 Audio, Network: Intel 82540EM Gigabit

Software:
OS: Ubuntu 14.04, Kernel: 4.4.0-47-generic (x86_64), Compiler: GCC 4.8, File-System: ext4, Screen Resolution: 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: run

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:

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: 10 Minutes
Running Pre-Test Script @ 11:14:26

```

Figura 1.6: Ejecución del benchmark elegido.

```

Enter a unique name to describe this test run / configuration: run
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:

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: 10 Minutes
    Running Pre-Test Script @ 11:14:26
    Started Run 1 @ 11:14:57
    Running Interim Test Script @ 11:17:05
    Started Run 2 @ 11:17:07
    Running Interim Test Script @ 11:19:04
    Started Run 3 @ 11:19:07 [Std. Dev: 2.76%]
    Running Post-Test Script @ 11:21:09

Test Results:
123.83170890808
117.29145002365
121.70164179802

Average: 120.94 Seconds

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/1612145-S0-RESULTADO30
juan@ubuntu mié dic 14 2016 ~ : -

```

Figura 1.7: Fin de la ejecución del benchmark elegido.

De este modo hemos obtenido que el promedio de tiempo de compilación del servidor apache http es de 120.94 segundos. Podemos comprobar como el tiempo mas rápido es de 117.30 segundos. El tiempo más lento es de 123.83 segundos.

OpenBenchmarking también nos permite subir los datos a su página, de tal forma que nos muestra una gráfica de los resultados, y podemos comparar dichos resultados con los de otras máquinas que hayan sido previamente analizadas.

En el siguiente enlace puede verse los resultados del benchmark que se realizó. [2]

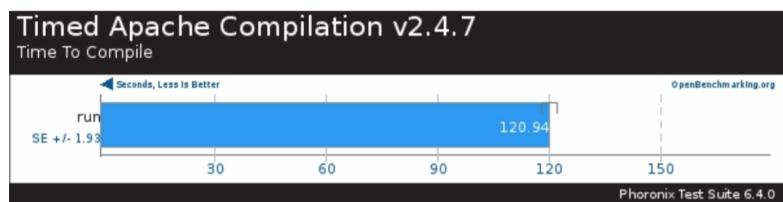


Figura 1.8: Gráfica de OpenBenchmarking.

resultados	
	
run	
build-apache: Time To Compile	120.94
Standard Error	1.93
Standard Deviation	2.76%
OpenBenchmarking.org	

Figura 1.9: Datos resultantes del benchmark.

Comparativa con otra máquina:

Test Systems:

run

Processor: Intel Core i5-5257U @ 2.70GHz (1 Core), Motherboard: Oracle VirtualBox v1.2, Chipset: Intel 440FX- 82441FX PMC, Memory: 2048MB, Disk: 2 x 9GB VBOX HDD, Graphics: InnoTek VirtualBox, Audio: Intel 82801AA AC 97 Audio, Network: Intel 82540EM Gigabit

OS: Ubuntu 14.04, Kernel: 4.4.0-47-generic (x86_64), Compiler: GCC 4.8, File-System: ext4, Screen Resolution: 800x600, System Layer: VirtualBox

build-apache

Processor: Intel Core 2 4300 @ 1.80GHz (2 Cores), Motherboard: HP OA68h, Chipset: Intel 82975X MCH + ICH7, Memory: 1024MB, Disk: 2 x 250GB Seagate ST3250820AS, Graphics: AMD FireGL V3350 256MB, Audio: Realtek ALC262, Monitor: HP L1706, Network: Broadcom Limited NetXtreme BCM5755 Gigabit PCI

OS: Fedora 25, Kernel: 4.8.10-300.fc25.x86_64 (x86_64), Desktop: Xfce 4.12, Display Driver: modesetting 1.19.0, OpenGL: 2.1 Mesa 12.0.3 Gallium 0.4, Compiler: Clang 3.8.0, File-System: ext4, Screen Resolution: 1280x1024

Figura 1.10: Datos de las máquinas a comparar.

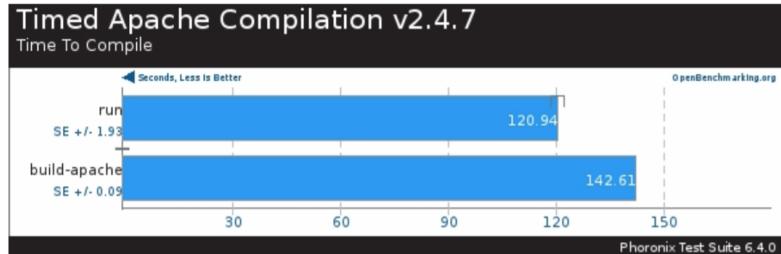


Figura 1.11: Gráfica de la comparativa.

2. **Para Apache Benchmark, de los parámetros que le podemos pasar al comando ab, ¿Qué significa -c 5? Y -n 100? Monitorice la ejecución de ab contra alguna máquina (cualquiera)¿Cuántas “tareas” crea ab en el cliente?**

2.1. Respuesta :

El comando -c 5 significa que se lanzarán 5 solicitudes simultáneamente. Y el comando -n 100 significa el número de solicitudes que serán lanzadas en el benchmark. [3]

Probaremos el comando ab contra una máquina Ubuntu server 14, desde mi ordenador personal.

```
ivan@MacBook-Pro-de-Ivan jue dic 15 2016 ~ : ab -c 5 -n 1000 http://192.168.56.105/
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
```

```
Benchmarking 192.168.56.105 (be patient)
```

```
Completed 100 requests
```

```
Completed 200 requests
```

```
Completed 300 requests
```

```
Completed 400 requests
```

```
Completed 500 requests
```

```
Completed 600 requests
```

```
Completed 700 requests
```

```
Completed 800 requests
```

```
Completed 900 requests
```

```
Completed 1000 requests
```

```
Finished 1000 requests
```

```
Server Software:      Apache/2.4.7
```

```
Server Hostname:     192.168.56.105
```

```
Server Port:         80
```

```
Document Path:       /
```

```
Document Length:    11510 bytes
```

```
Concurrency Level:   5
```

```
Time taken for tests: 0.752 seconds
```

```
Complete requests:  1000
```

```
Failed requests:    0
```

```
Total transferred: 11783000 bytes
```

```
HTML transferred:   11510000 bytes
```

```
Requests per second: 1329.15 [#/sec] (mean)
```

```
Time per request:   3.762 [ms] (mean)
```

```
Time per request:   0.752 [ms] (mean, across all concurrent requests)
```

```
Transfer rate:      15294.30 [Kbytes/sec] received
```

```
Connection Times (ms)
```

	min	mean	[+/-sd]	median	max
--	-----	------	---------	--------	-----

Connect:	0	0	0.1	0	1
----------	---	---	-----	---	---

Processing:	1	3	1.8	3	24
-------------	---	---	-----	---	----

Waiting:	0	3	0.9	3	10
----------	---	---	-----	---	----

Total:	1	4	1.8	3	25
--------	---	---	-----	---	----

```
Percentage of the requests served within a certain time (ms)
```

50%	3
-----	---

66%	4
-----	---

75%	4
-----	---

80%	4
-----	---

90%	6
-----	---

95%	7
-----	---

98%	9
-----	---

99%	10
-----	----

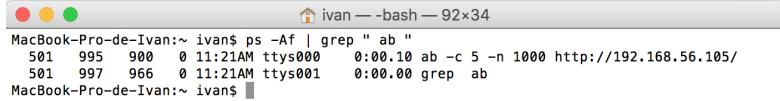
100%	25 (longest request)
------	----------------------

```
ivan@MacBook-Pro-de-Ivan jue dic 15 2016 ~ : █
```

11

Figura 2.1: Salida de la ejecución del comando ab.

Para mostrar la cantidad de tareas (procesos) que se crean en el cliente cuando lanzamos el comando ab contra la máquina Ubuntu server, usamos el comando ps -Af | grep " ab ", nos mostrará dos procesos, pero uno es grep, que se lanza con este último comando mencionado, luego solamente hacemos caso al otro que aparece.



```
 MacBook-Pro-de-Ivan:~ ivan$ ps -Af | grep " ab "
 501  995  900  0 11:21AM ttys000  0:00.10 ab -c 5 -n 1000 http://192.168.56.105/
 501  997  966  0 11:21AM ttys001  0:00.00 grep ab
MacBook-Pro-de-Ivan:~ ivan$
```

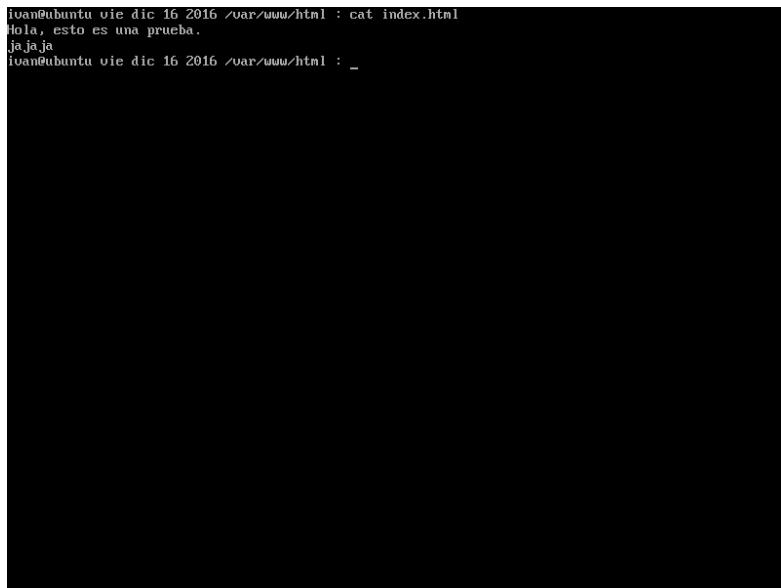
Figura 2.2: Mostrando la cantidad de procesos creados en el cliente con el lanzamiento del comando ab contra otra máquina.

- 3. Ejecute ab contra las tres máquinas virtuales (desde el SO anfitrión a las máquinas virtuales de la red local) una a una (arrancadas por separado). ¿Cuál es la que proporciona mejores resultados? Muestre y coméntelos. (Use como máquina de referencia Ubuntu Server para la comparativa).**

3.1. Respuesta :

En primer lugar debemos modificar los ficheros index.html de cada servidor, pues es obvio que no es lo mismo cargar la página principal de apache en Ubuntu Server, con texto solamente, que cargar la página principal de IIS en Windows server, que a parte de texto contiene una imagen.

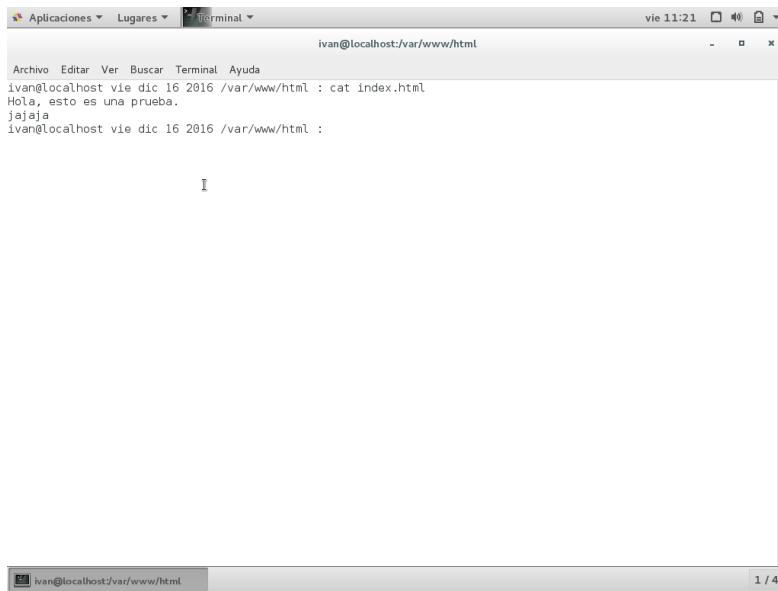
Para modificar los ficheros .html, en Ubuntu debemos ir a la ruta “/var/www/html” y nos encontraremos con un fichero llamado index, podemos hacer dos cosas, en primer lugar una copia de ese fichero para cuando acabemos de hacer nuestras comprobaciones poder volver a lo que teníamos por defecto (Es lo que yo haría), ó directamente manipular este fichero sin hacer copia. En nuestro caso hacemos copia de seguridad y cambiamos el fichero por el siguiente:



```
ivan@ubuntu vie dic 16 2016 /var/www/html : cat index.html
Hola, esto es una prueba.
ja.ja.ja
ivan@ubuntu vie dic 16 2016 /var/www/html : _
```

Figura 3.1: Nuevo index.html en Ubuntu Server.

Para el caso de CentOS 7, es más sencillo ya que cuando nos sale la típica página por defecto del servidor httpd (Testing 123....) es porque no existe ningún index.html, luego nos bastará con crearlo.



A screenshot of a terminal window titled "Terminal". The window shows the command "ivan@localhost:~\$ cat index.html" followed by the content "Hola, esto es una prueba." and "jajaja". The window has a standard Linux desktop interface with a title bar, menu bar, and status bar indicating the date and time.

Figura 3.2: Nuevo index.html en CentOS.

Para el caso de Windows Server 2012 es aún más sencillo con la interfaz, ya que debemos irnos a Server Manager, desde allí entramos a IIS, y una vez dentro nos dirigimos hacia tools(IIS manager), cuando estemos dentro en sites-Default Web Site botón derecho y explorar. La ruta tal vez sea difícil de recordar, pero se hace trivial como se ha dicho antes con la interfaz. Cuando estemos dentro haremos copia de seguridad de los archivos, y los cambiaremos por un index.html de nuestro puño y letra.

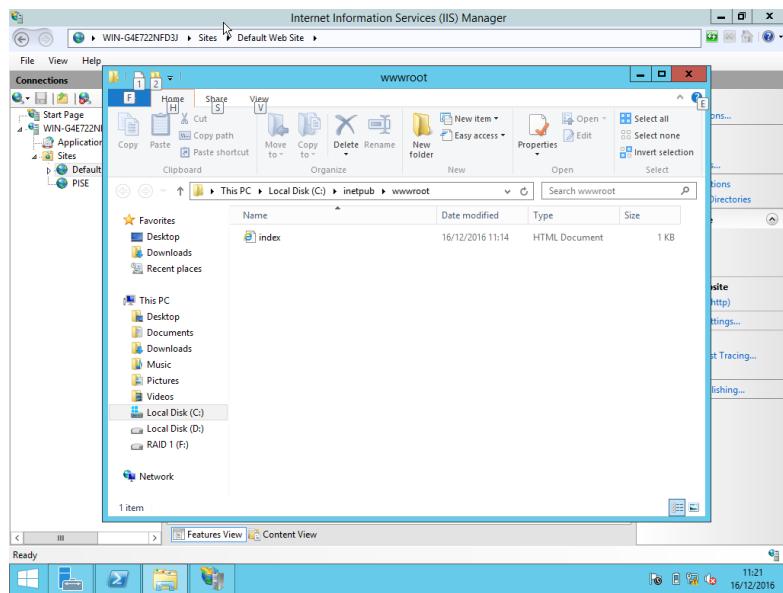


Figura 3.3: Nuevo index.html en Windows Server.

Para terminar con esta configuración previa, se muestra una imagen desde el anfitrión conectándose a las tres páginas:

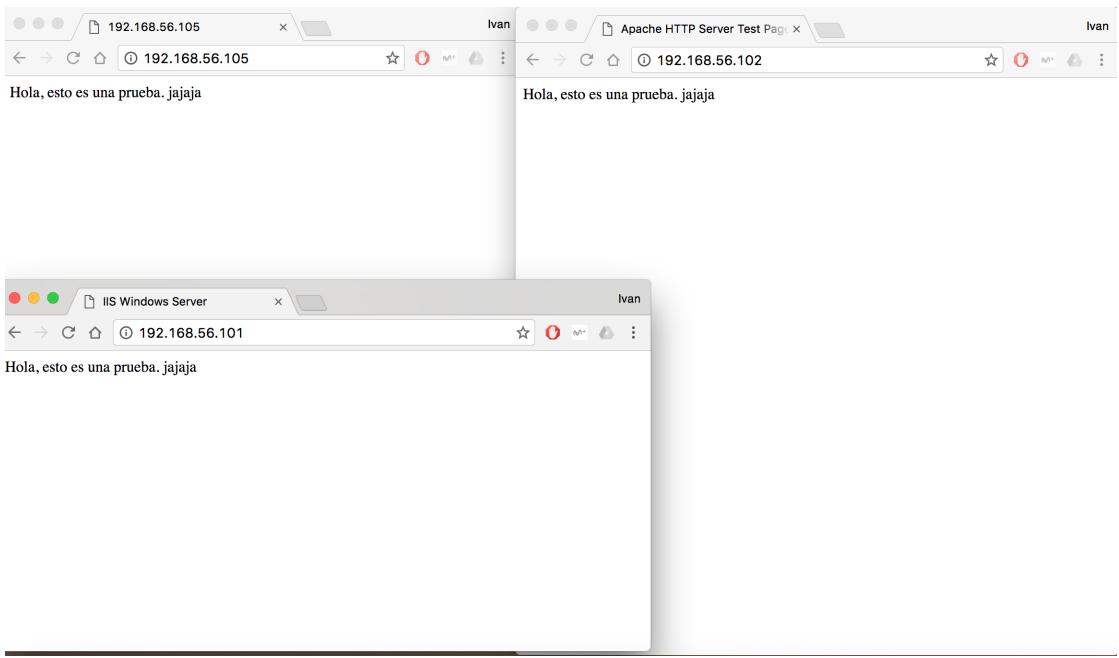


Figura 3.4: Conexión desde el anfitrión a las tres páginas de Apache, Httpd e IIS (Ubuntu, CentOS 7 y Windows Server).

Ahora procederé a mostrar las imágenes del lanzamiento del comando ab contra cada máquina (Servidor). [3]

Lanzamiento del comando ab contra Ubuntu Server:

```

ivan@MacBook-Pro-de-Ivan ~ dic 16 2016 : ab -c 5 -n 1000 http://192.168.56.105/
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.56.105 (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests

Server Software:      Apache/2.4.7
Server Hostname:     192.168.56.105
Server Port:          80

Document Path:        /
Document Length:     33 bytes

Concurrency Level:    5
Time taken for tests: 0.869 seconds
Complete requests:   1000
Failed requests:     0
Total transferred:   278000 bytes
HTML transferred:    33000 bytes
Requests per second: 1150.59 [#/sec] (mean)
Time per request:   4.346 [ms] (mean)
Time per request:   0.869 [ms] (mean, across all concurrent requests)
Transfer rate:       312.37 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0  0.1     0      1
Processing:     0    1  0.2     1      2
Waiting:        0    0  0.1     0      2
Total:          1    1  0.2     1      3

Percentage of the requests served within a certain time (ms)
 50%      1
 66%      1
 75%      1
 80%      1
 90%      1
 95%      1
 98%      1
 99%      2
100%      3 (longest request)
ivan@MacBook-Pro-de-Ivan ~ dic 16 2016 ~ : █

```

Figura 3.5: Lanzamiento del comando ab contra Ubuntu Server.

Lanzamiento del comando ab contra CentOS 7:

```

ivan@MacBook-Pro-de-Ivan ~ dic 16 2016 : ab -c 5 -n 1000 http://192.168.56.102/
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.56.102 (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests

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

Document Path:        /
Document Length:     33 bytes

Concurrency Level:    5
Time taken for tests: 0.968 seconds
Complete requests:   1000
Failed requests:     0
Total transferred:   304000 bytes
HTML transferred:    33000 bytes
Requests per second: 1032.54 [#/sec] (mean)
Time per request:    4.842 [ms] (mean)
Time per request:    0.968 [ms] (mean, across all concurrent requests)
Transfer rate:        306.54 [Kbytes/sec] received

Connection Times (ms)
              min  mean [+/-sd] median   max
Connect:        0    0   0.1     0     1
Processing:     0    1   0.2     1     3
Waiting:        0    1   0.1     1     2
Total:          1    1   0.2     1     3

Percentage of the requests served within a certain time (ms)
 50%    1
 66%    1
 75%    1
 80%    1
 90%    1
 95%    1
 98%    2
 99%    2
100%   3 (longest request)
ivan@MacBook-Pro-de-Ivan ~ dic 16 2016 ~ :

```

Figura 3.6: Lanzamiento del comando ab contra CentOS 7.

Lanzamiento del comando ab contra Windows Server 2012:

```

[ivan@MacBook-Pro-de-Ivan ~ : ab -c 5 -n 1000 http://192.168.56.101/
This is ApacheBench, Version 2.3 <Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.56.101 (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests

Server Software:      Microsoft-IIS/8.5
Server Hostname:     192.168.56.101
Server Port:          80

Document Path:        /
Document Length:     32 bytes

Concurrency Level:    5
Time taken for tests: 0.782 seconds
Complete requests:   1000
Failed requests:      0
Total transferred:   273000 bytes
HTML transferred:    32000 bytes
Requests per second: 1279.15 [#/sec] (mean)
Time per request:    3.909 [ms] (mean)
Time per request:    0.782 [ms] (mean, across all concurrent requests)
Transfer rate:        341.02 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       0    0.1      0       1
Processing:    0    0.2      0       4
Waiting:      0    0.2      0       4
Total:         0    0.3      1       5
ERROR: The median and mean for the processing time are more than twice the standard
       deviation apart. These results are NOT reliable.

Percentage of the requests served within a certain time (ms)
  50%    1
  66%    1
  75%    1
  80%    1
  90%    1
  95%    1
  98%    2
  99%    2
 100%   5 (longest request)
ivan@MacBook-Pro-de-Ivan ~ :

```

Figura 3.7: Lanzamiento del comando ab contra Windows Server 2012.

Podemos observar como para Ubuntu tenemos un tiempo total de 0,869 segundos, promedio de solicitudes 1150,59 solicitudes / segundo, promedio de tiempo por solicitud de 4,346 ms / solicitud y una velocidad de transferencia de 312,37 KB / segundo.

En cuanto a CentOS 7 tenemos un tiempo total de 0,968 segundos, promedio de solicitudes 1032,54 solicitudes / segundo, promedio de tiempo por solicitud de 4,842 ms / solicitud y una velocidad de transferencia de 306,54 KB / segundo.

Para terminar en Windows Server 2012 tenemos un tiempo total de 0,782 segundos, promedio de solicitudes 1279,15 solicitudes / segundo, promedio de tiempo por solicitud de 3,909 ms / solicitud y una velocidad de transferencia de 341,02 KB / segundo.

Con estos datos en la mano podemos decir que el más rápido ha sido Windows server 2012. Ya que por ejemplo tardó 0,182 segundos menos que CentOS 7 y 0,082 segundos menos que Ubuntu, con lo que podemos ver que Ubuntu y Windows Server están casi a la par, contando Ubuntu con una velocidad de transferencia bastante menor que Windows Server 2012.

4. Instale y siga el tutorial en la página de JMeter 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?

4.1. Respuesta :

En primer lugar vamos a la página oficial de JMeter y en la parte de la izquierda nos encontraremos con descargas [4], una vez dentro nos descargamos el “.zip” de la sección binarios. Cuando lo hayamos descargado pasamos a descomprimirlo y entramos con la terminal en la carpeta “/bin”.

```
ivan@cv1099038 vie dic 16 2016 ~/Desktop/apache-jmeter-3.1/bin : ls
ApacheJMeter.jar          hc.parameters      jmeter-server.bat      logkit.xml           stoptest.cmd
BeanShellAssertion.bshrc   heapdump.cmd     jmeter-t.cmd        mirror-server       stoptest.sh
BeanShellFunction.bshrc    heapdump.sh      jmeter.bat         mirror-server.cmd  system.properties
BeanShellListeners.bshrc   httpclient.parameters  jmeter.log          mirror-server.sh  templates
BeanShellSampler.bshrc    jaas.conf        jmeter.properties   report-template   upgrade.properties
Prueba Ubuntu Server.jmx   jmeter          jmeter.sh          reportgenerator.properties
Result1.jmx                jmeter-n-r.cmd   jmeterw.cmd       saveservice.properties
UNIX.jmx                  jmeter-n.cmd    krb5.conf        shutdown.cmd
examples                  jmeter-server   log4j.conf       shutdown.sh
ivan@cv1099038 vie dic 16 2016 ~/Desktop/apache-jmeter-3.1/bin : ./jmeter.sh
Writing log file to: /Users/ivan/Desktop/apache-jmeter-3.1/bin/jmeter.log
=====
Don't use GUI mode for load testing, only for Test creation and Test debugging !
For load testing, use NON GUI Mode & adapt Java Heap to your test requirements
=====
```

Figura 4.1: Ejecución de JMeter desde la terminal.

Ahora pasaremos a crearnos nuestro propio Web Test Plan. En primer lugar pasamos a añadir un grupo de usuarios, que son los que harán las solicitudes.

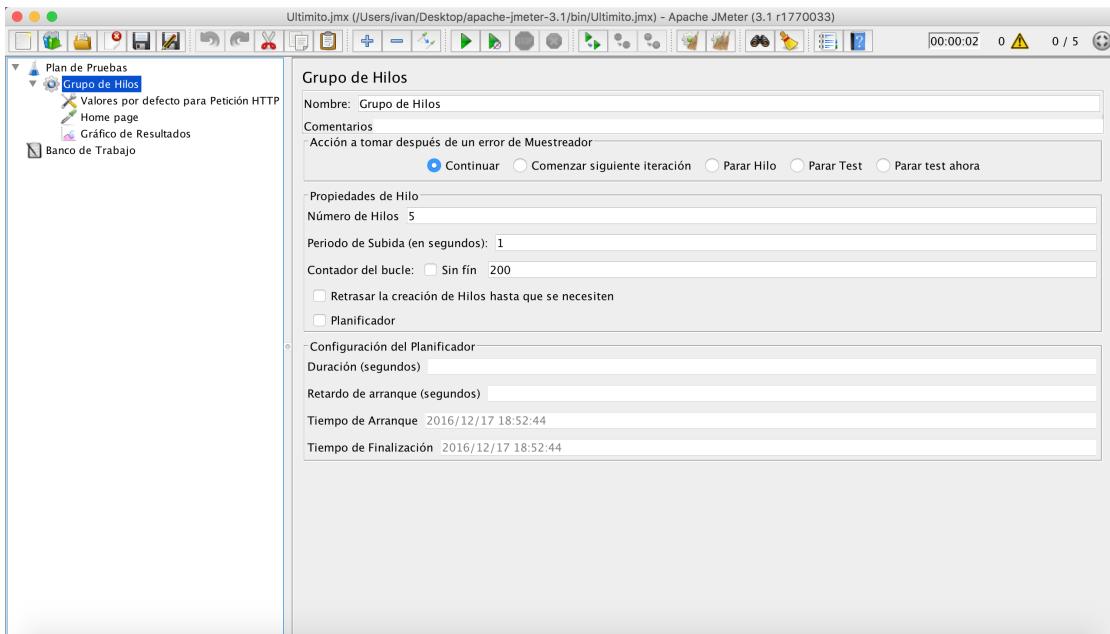


Figura 4.2: Añadiendo grupo de usuarios a nuestro test.

Como puede verse indicamos que el número de usuarios sea de 5, con un período de subida de 1 segundo y que se repita 200 veces entre otras cosas para llegar a las 1000 muestras como en el caso del comando ab lanzado anteriormente.

En segundo lugar pasamos a añadir lo que van a hacer dichos usuarios creados anteriormente. En nuestro caso peticiones HTTP:

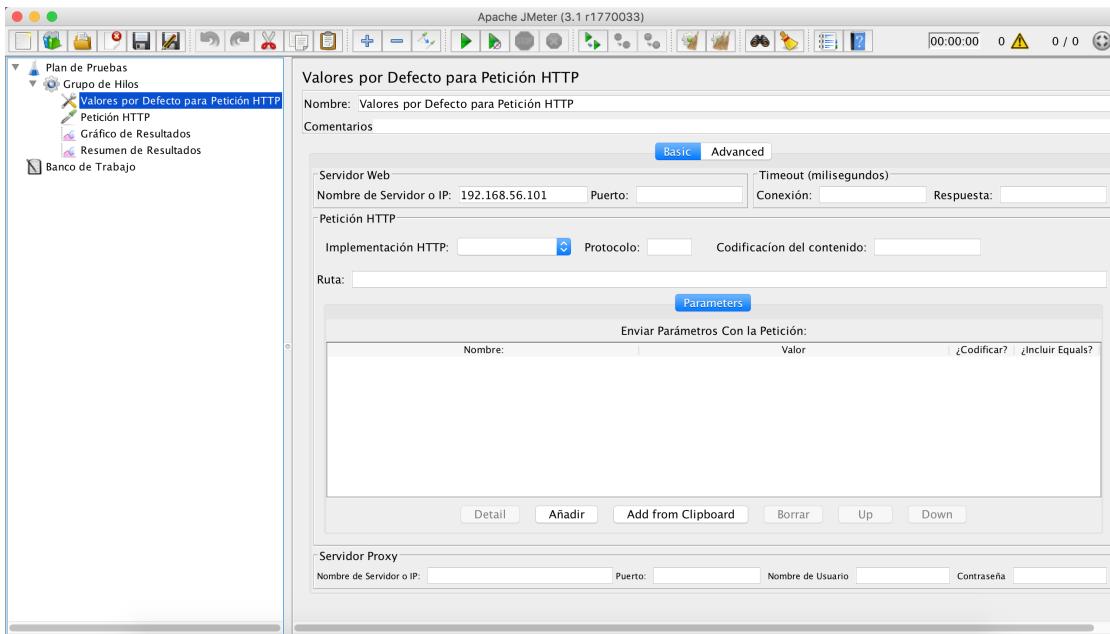


Figura 4.3: Añadiendo valores por defecto para petición HTTP.

En este apartado debemos indicar la IP de la máquina virtual con la que vayamos a hacer la prueba, en nuestro caso con Ubuntu Server.

En tercer lugar añadimos las peticiones HTTP, aquí solamente debemos cambiar la ruta a “/” y no hace falta indicar la IP del servidor, pues con que se haya indicado en la imagen 4.3 es suficiente según la documentación de JMeter.[4].

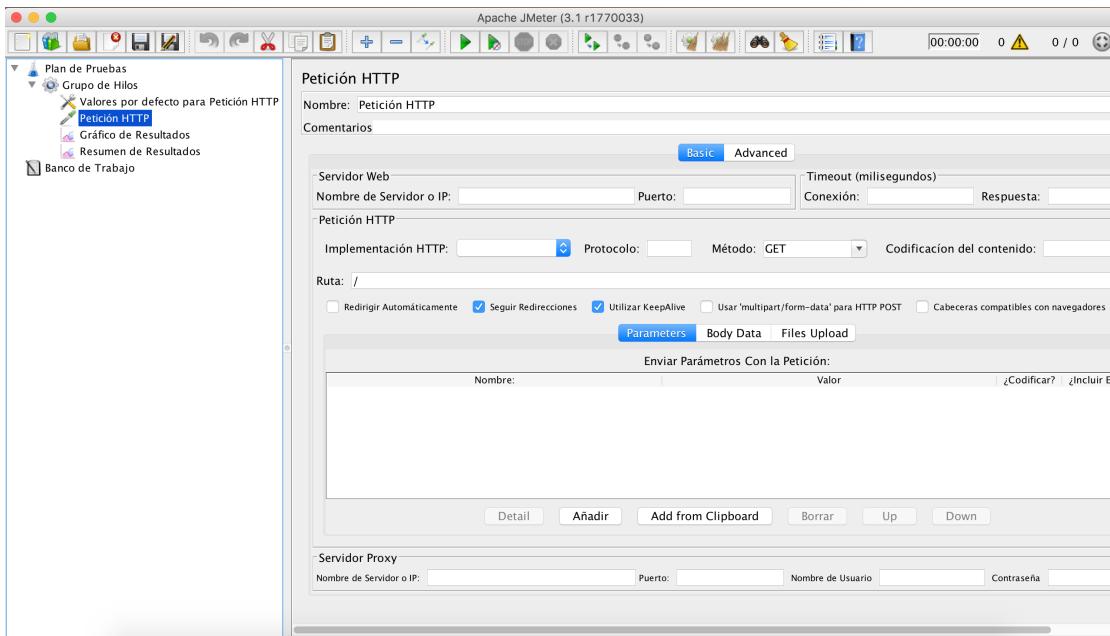


Figura 4.4: Añadiendo el tipo de peticiones HTTP.

Y por último lugar debemos añadir el gráfico de los resultados para poder interpretar los datos.

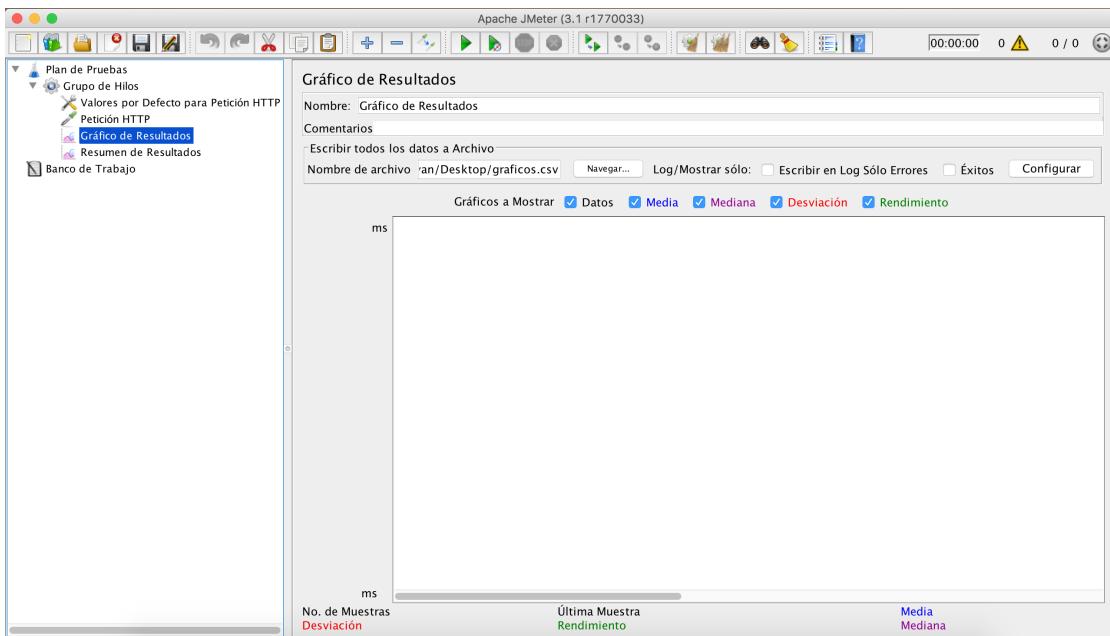


Figura 4.5: Añadiendo el gráfico de resultados.

Una vez hecho todo el proceso, podemos iniciar el test, esto se hace lanzar->arrancar. Y nos debería quedar algo parecido a la siguiente imagen:

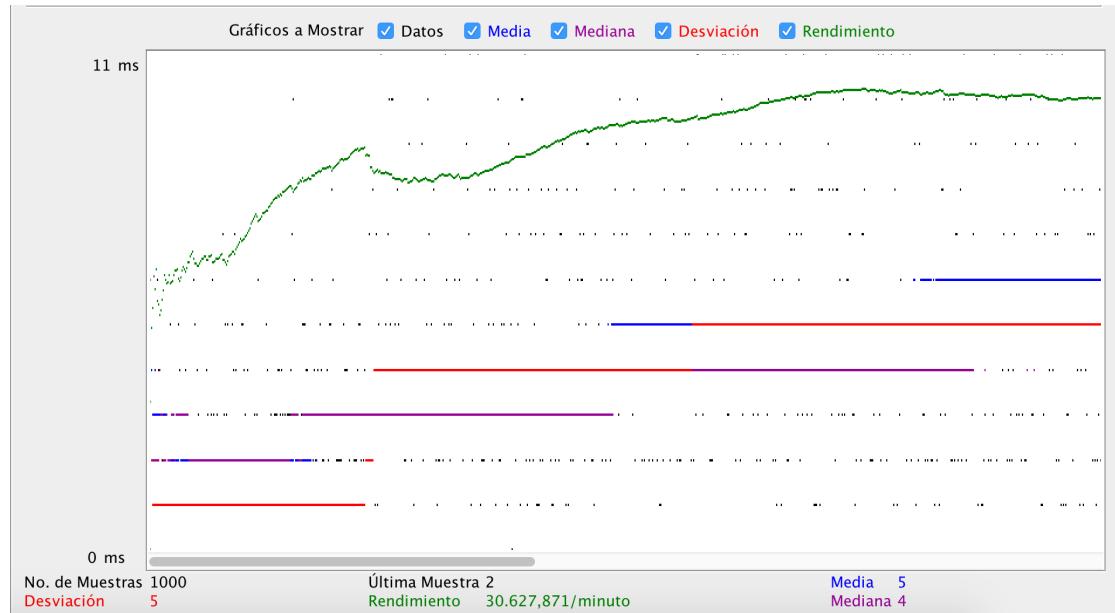


Figura 4.6: Finalización del lanzamiento del test.

Para mostrar los datos tenemos varias opciones, o bien que se nos guarde en un fichero “.csv”, observarlos desde la misma gráfica 4.6 ó tenemos otra opción que es el Apache JMeter Dashboard [5] , que en comparación con los otros dos métodos es bastante más sencillo y a la vez tenemos más gráficas en donde poder ver todos los datos reflejados. Para que JMeter nos proporcione un “.html” que nos de acceso al Dashboard debemos lanzarlo con el siguiente comando una vez que estemos dentro de la carpeta descargada en /bin:

- sh jmeter -n -t <test creado.jmx>-l <ruta donde quieras que se guarde el .csv>-e -o <ruta donde quieras que se guarde el html>

En mi caso:

```

ivang@MacBook-Pro-de-Ivan dom dic 18 2016 ~/Desktop/apache-jmeter-3.1/bin : sh jmeter -n -t Ultimito.jmx -l /Users/ivan/Desktop/test1.csv -e -o /Users/ivan/Desktop/HTMLrep/
Writing results file to: /Users/ivan/Desktop/apache-jmeter-3.1/bin/jmeter.log
Creating summariser <summary>
Created the tree successfully using Ultimito.jmx
Starting the test @ Sun Dec 18 11:32:04 CET 2016 (1482057124940)
Waiting for possible Shutdown/StopTestNow/Heapdump message on port 4445
summary + 1 in 00:00:00 = 4,5/s Avg: 102 Min: 102 Max: 102 Err: 0 (0,0%) Active: 1 Started: 1 Finished: 0
summary + 999 in 00:00:01 = 1117,4/s Avg: 1 Min: 0 Max: 27 Err: 0 (0,0%) Active: 0 Started: 5 Finished: 5
summary = 1000 in 00:00:01 = 896,1/s Avg: 1 Min: 0 Max: 102 Err: 0 (0,0%)
Tidying up ... @ Sun Dec 18 11:32:06 CET 2016 (1482057126137)
... end of run
ivan@MacBook-Pro-de-Ivan dom dic 18 2016 ~/Desktop/apache-jmeter-3.1/bin :

```

Figura 4.7: Ejecución de JMeter desde línea de comandos para la obtención del index.html.

Así podremos acceder mediante un navegador al Apache JMeter Dashboard clicando sobre el index.html. Así se nos abrirá la siguiente ventana en el navegador:

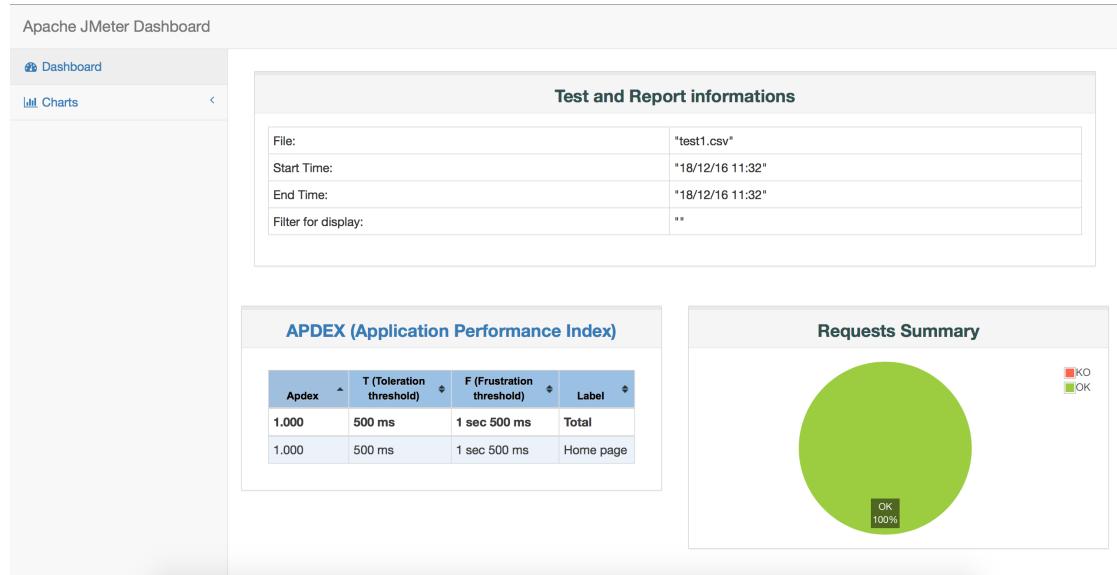


Figura 4.8: Inicio Apache JMeter Dashboard.

En la misma página podemos ver también las estadísticas de la ejecución del test creado previamente.

Statistics													
Label	#Samples	KO	Error %	Average response time	90th pct	95th pct	99th pct	Throughput	Received KB/sec	Sent KB/sec	Min	Max	
Total	1000	0	0.00%	1.71	2.00	3.00	4.00	1006.04	308.24	179.79	0	102	
Home page	1000	0	0.00%	1.71	2.00	3.00	4.00	1006.04	308.24	179.79	0	102	

Figura 4.9: Estadísticas de la ejecución del test.

En esta tabla podemos ver que para unas 1000 muestras no ha habido ningún error. Hemos tenido una productividad de 1006.04 transacciones(solicitudes) / segundo. También vemos otros datos como los KB recibidos y enviados por segundo, en donde tenemos 308.24 KB/segundo recibidos y 179 KB / segundo enviados.

Por último comentar que hay multitud de gráficas como por ejemplo:

Aquí vemos que todas las solicitudes están dentro de los 500 ms.

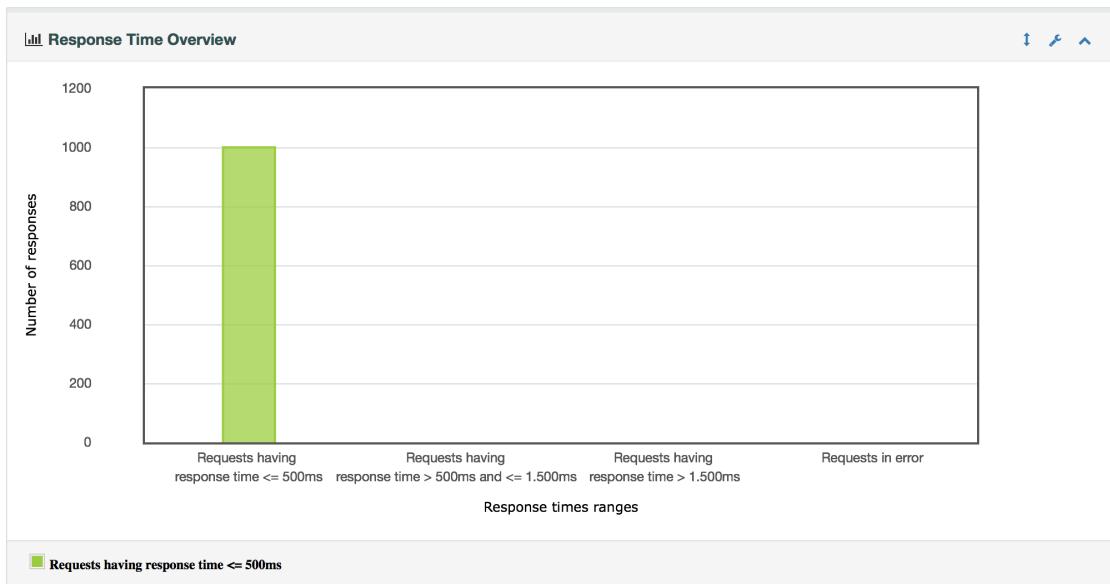


Figura 4.10: Tiempo total empleado para la generación de solicitudes.

En la siguiente gráfica vemos que las 1000 respuestas se distribuyen a lo largo de los 500 ms.

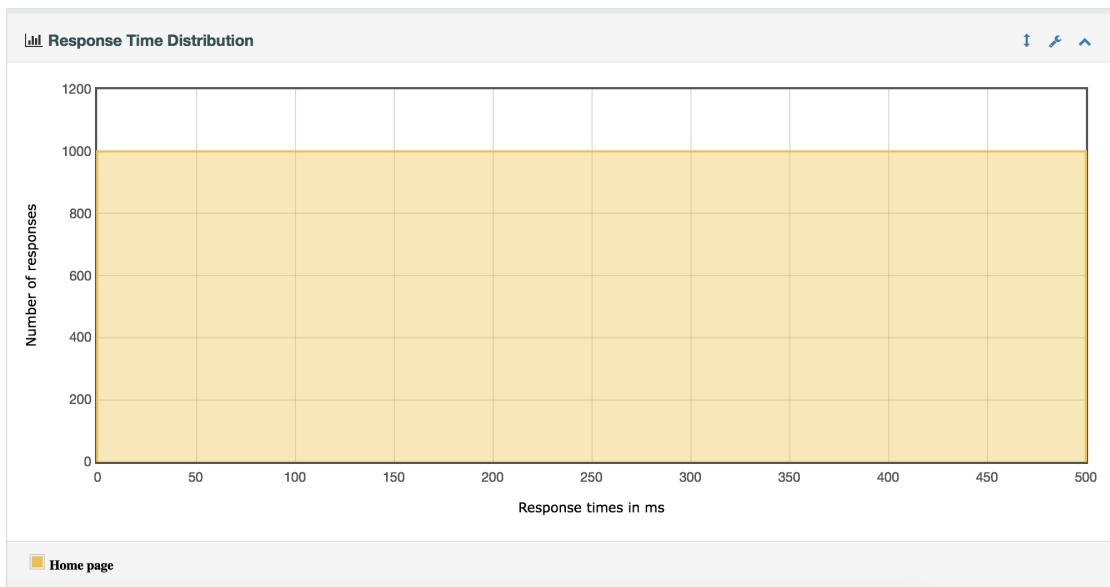


Figura 4.11: Distribución del tiempo de respuesta en milisegundos.

A continuación vemos que la media de tiempo de conexión fue de 0.097 ms.

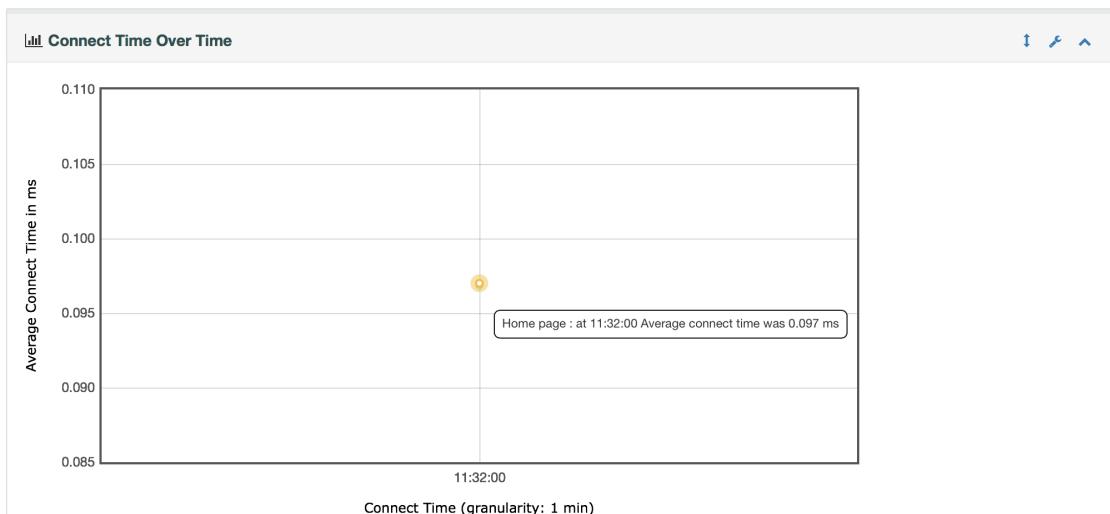


Figura 4.12: Media de tiempo de conexión en milisegundos.

En la siguiente gráfica se ve como la media de latencia en las respuesta es de 1.664 ms.

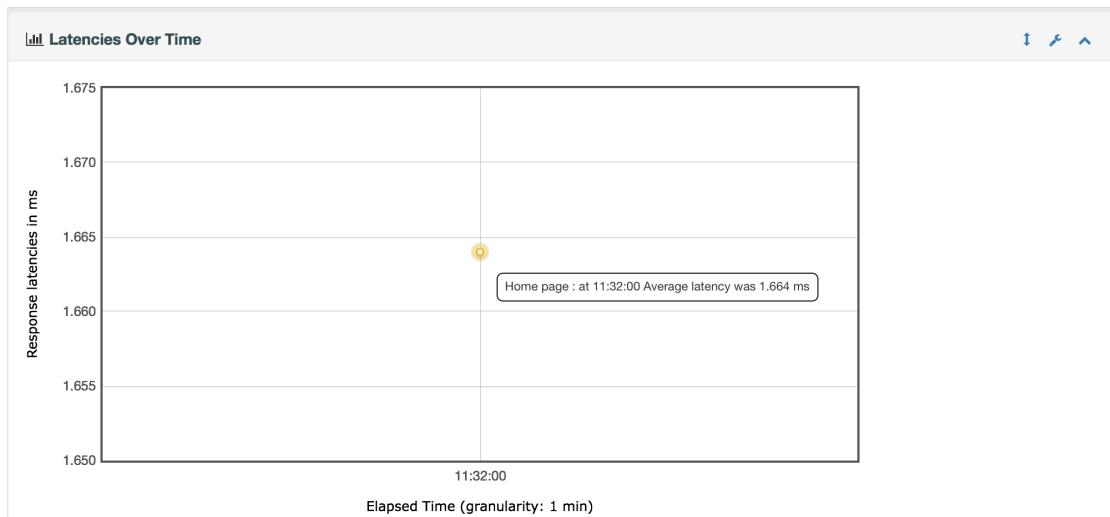


Figura 4.13: Media de latencia en respuestas en milisegundos.

Por último indicar que los datos obtenidos no tienen nada que ver con los mostrados por el comando ab.

Para que se valide todo lo mostrado anteriormente se adjunta con todo el trabajo la carpeta con el index.html y además el documento test.csv.

Para el caso de Windows Server hemos realizado el mismo trabajo, pero esta vez vamos a analizar los datos desde otro componente, para así mostrar los diferentes modos de visualizar los datos.

En primer lugar nos creamos nuestro grupo de hilos como antes (con 5 hilos y 200 peticiones) e introducimos el componente valores por defecto de HTTP.

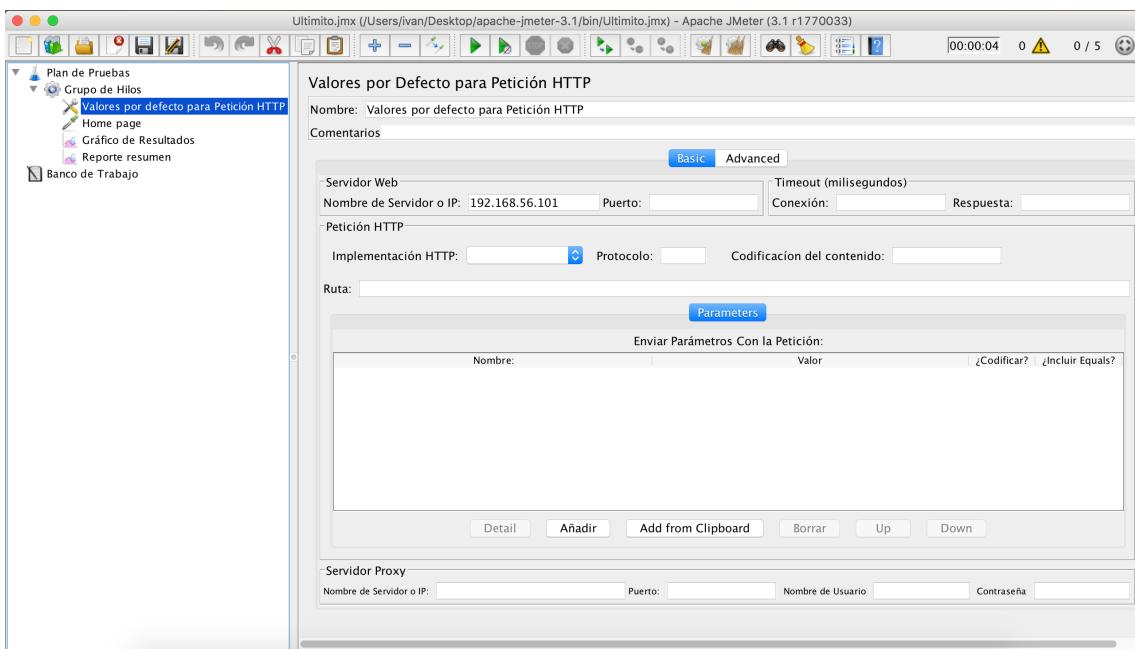


Figura 4.14: Creación de grupo de hilos y valores por defecto para Windows Server 2012.

En segundo lugar añadimos la solicitud HTTP.

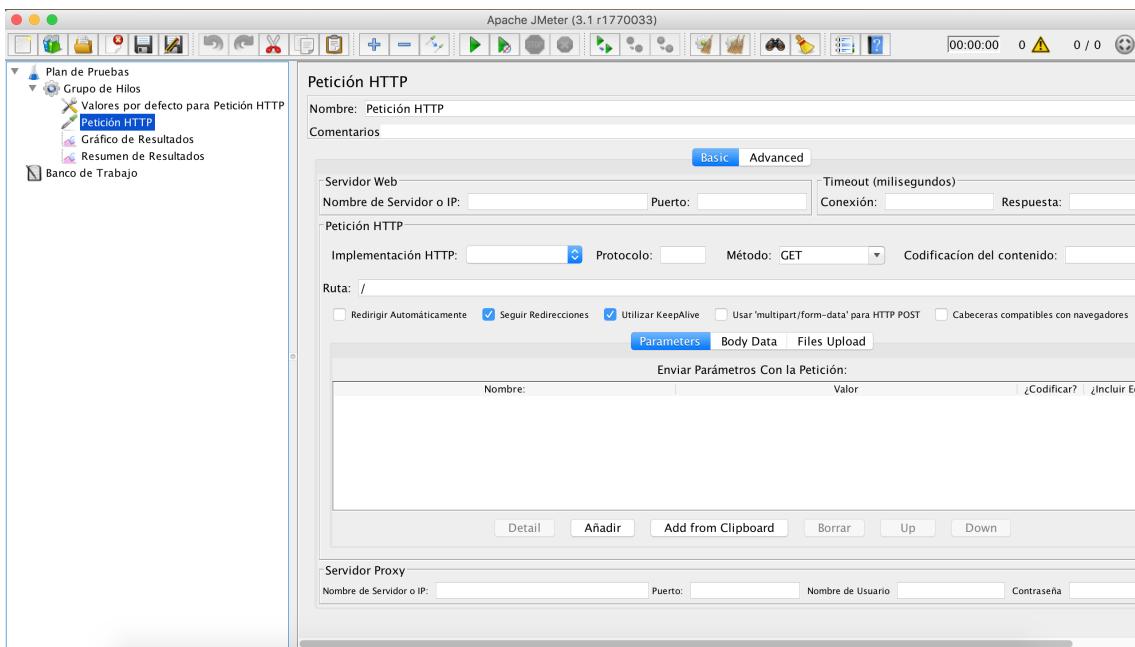


Figura 4.15: Añadiendo solicitud HTTP para Windows Server 2012.

En tercer lugar añadimos la gráfica para mostrar los datos.

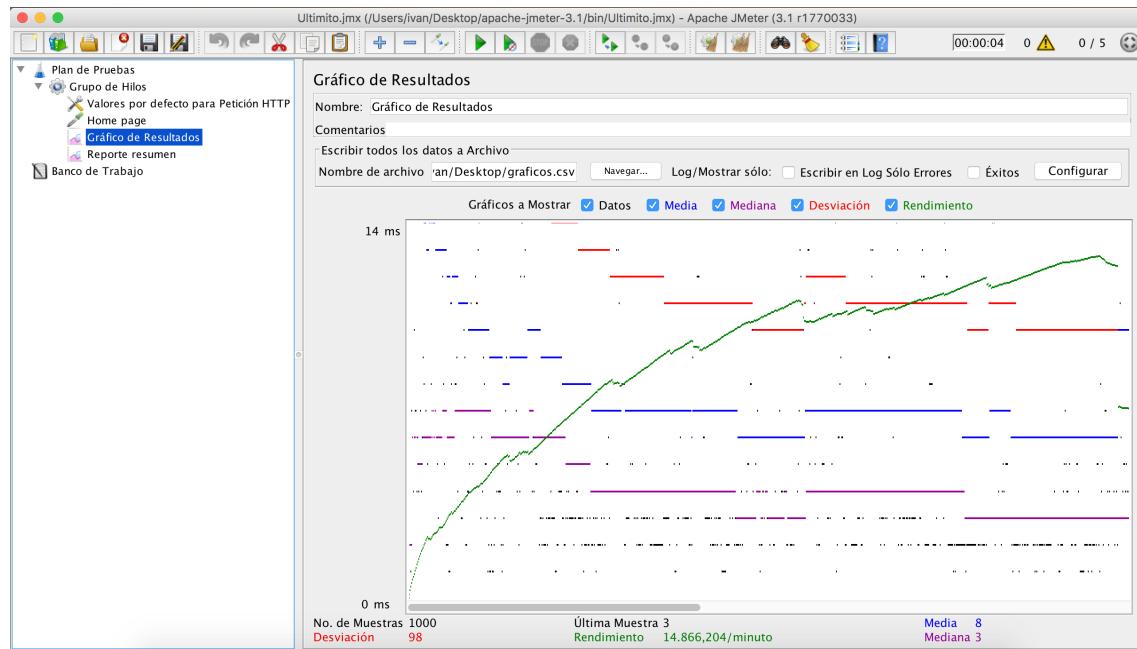


Figura 4.16: Añadiendo gráfica para Windows Server 2012.

Y por último lugar añadimos un componente nuevo, llamado reporte de datos, aquí se visualizarán los datos mas importantes de la prueba realizada.

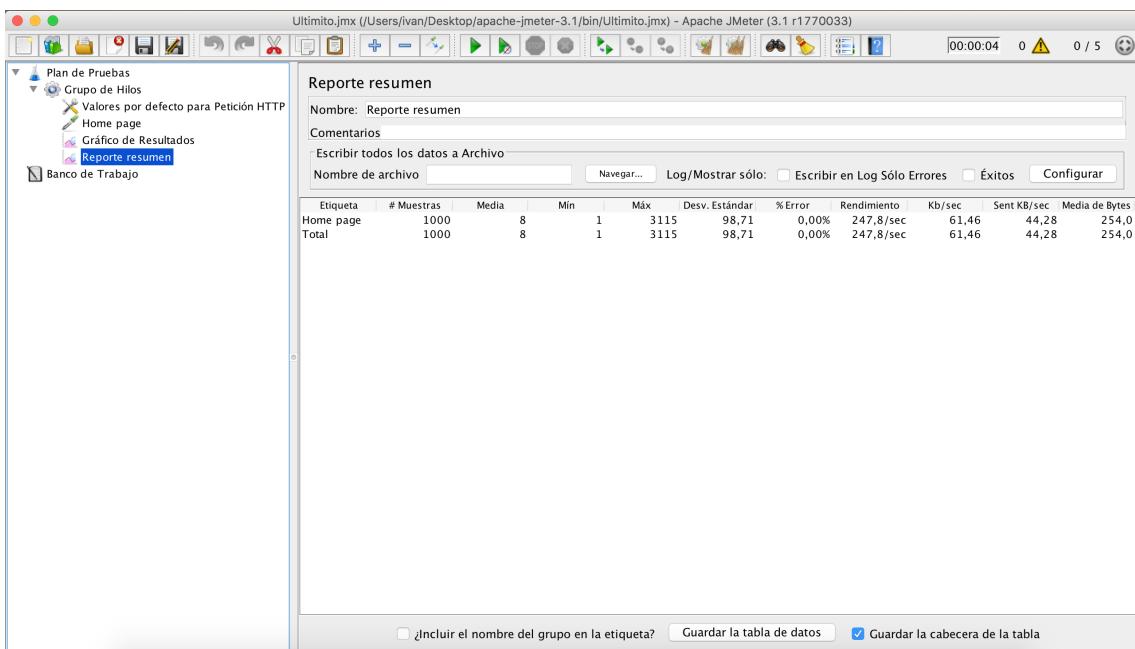


Figura 4.17: Añadiendo reporte de datos para Windows Server 2012.

Aquí podemos ver datos como el rendimiento que en este caso es de 247.8 solicitudes/-segundo, los KB por segundo que se envían que es de 44.28 KB / segundo, las muestras realizadas que es de 1000. Podemos ver que los resultados no tienen nada que ver con los obtenidos por el comando ab.

Para el caso de CentOS todo se ha montado de igual manera que para Ubuntu Server y Windows Server 2012.

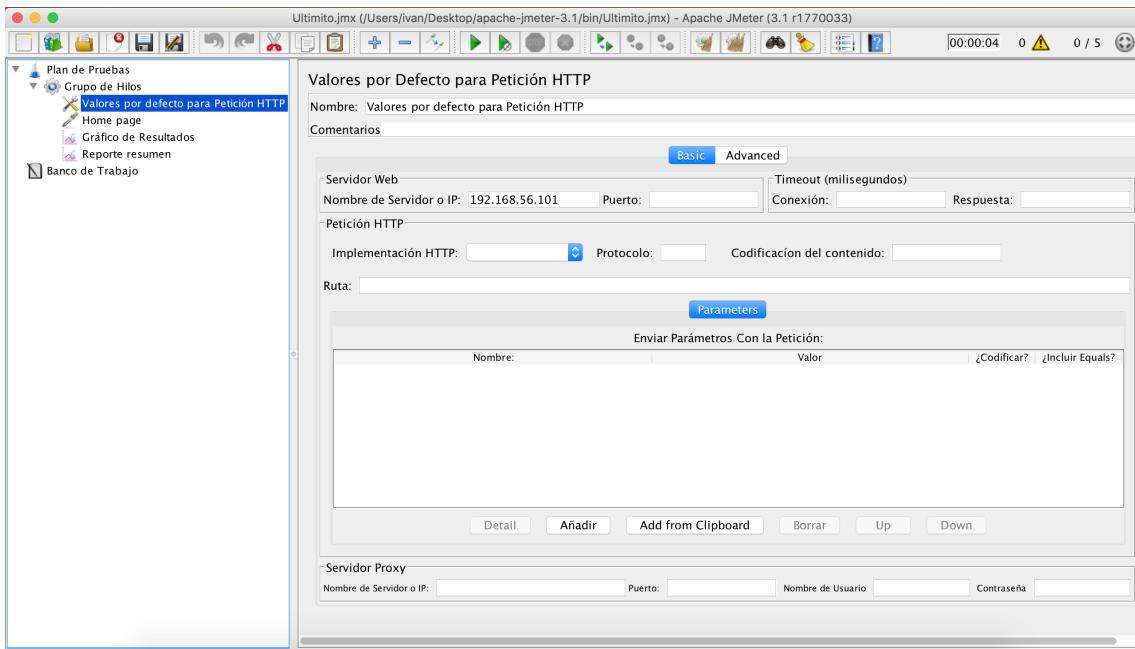


Figura 4.18: Creación de grupo de hilos y valores por defecto para CentOS 7.

En segundo lugar añadimos la solicitud HTTP.

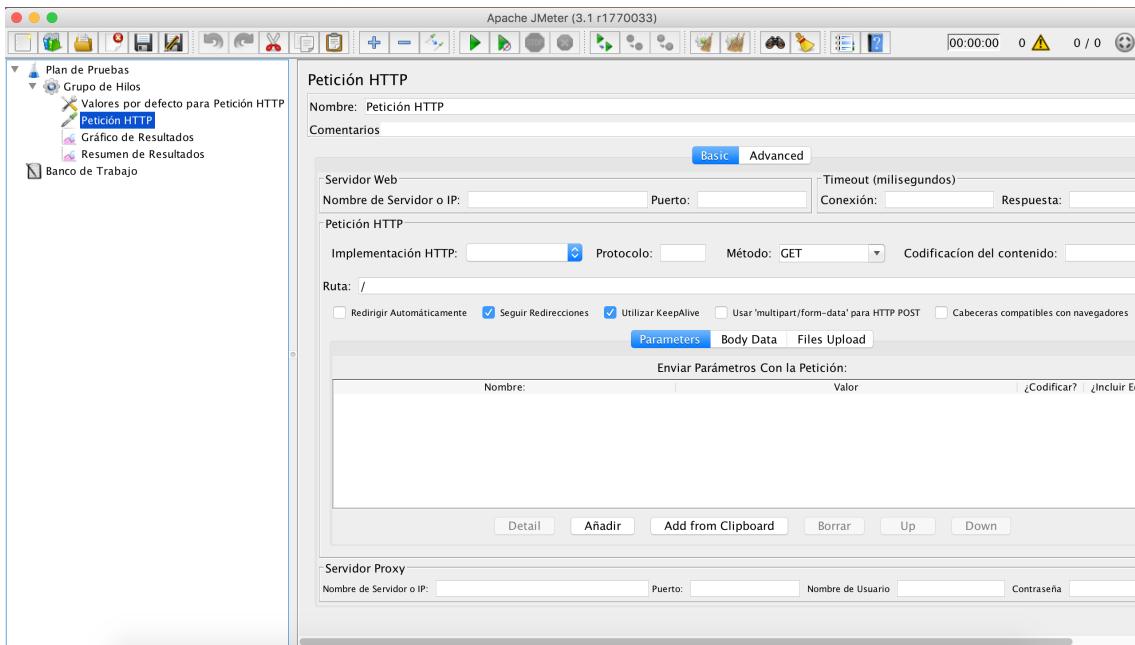


Figura 4.19: Añadiendo solicitud HTTP para CentOS 7.

En tercer lugar añadimos la gráfica para mostrar los datos.

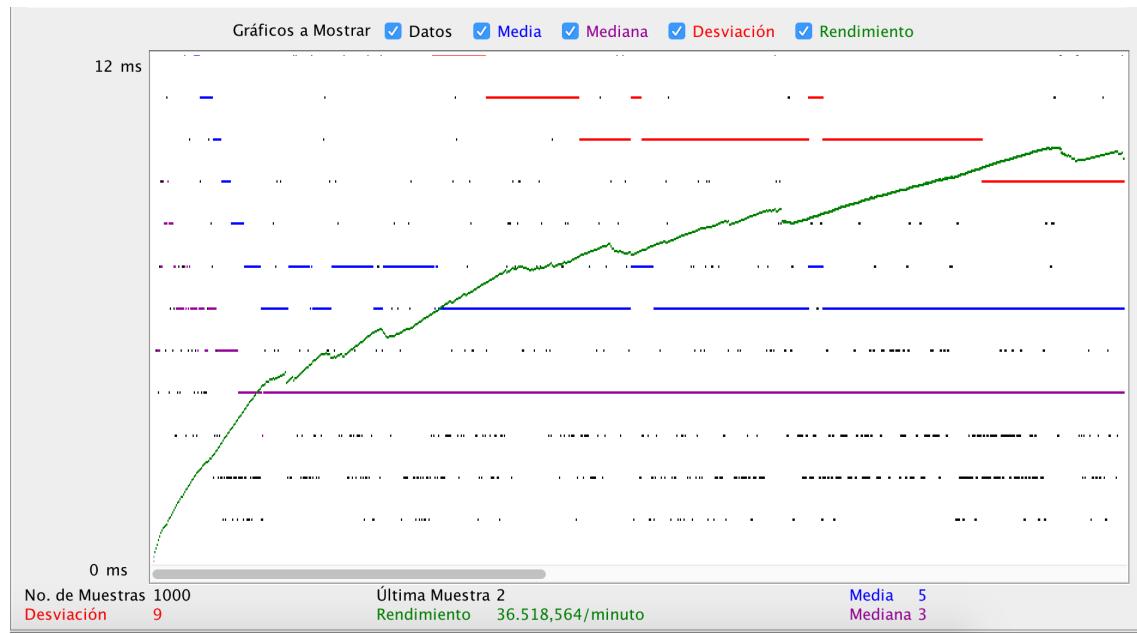


Figura 4.20: Gráfica de los resultados del benchmark creado y ejecutado contra CentOS 7.

Y por último añadimos un reporte de datos:

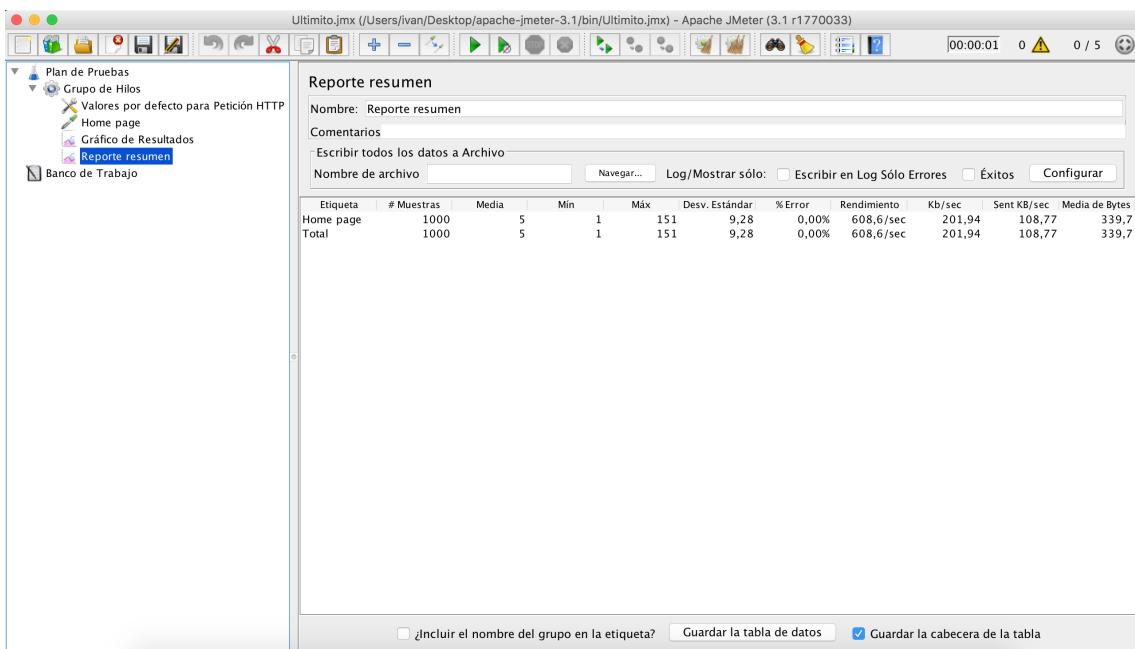


Figura 4.21: Datos del resumen con respecto a la ejecución del benchmark contra CentOS 7.

Se puede observar un rendimiento de 608.6 solicitudes / segundo. También se ve que se envían unos 108.77 KB / segundo. Mientras que la desviación estándar total con el comando ab era de 0.2, el mínimo para el tiempo de conexión en total de 1, una media de 1 y un máximo de 3 en total. Con JMeter nos sale una desviación típica de 9.28 , una media de 5, un mínimo de 1 y un máximo de 151. Sobre todo para el máximo y desviación estándar se ven valores muy diferentes.

**5. Programe un benchmark usando el lenguaje que deseé.
El benchmark debe incluir: Objetivo del benchmark,
métricas (unidades, variables, puntuaciones, etc.),
instrucciones para su uso y ejemplo de uso analizando
los resultados.**

5.1. Respuesta :

5.2. Objetivo del Benchmark:

En mi caso voy a desarrollar un benchmark para la comparación de 3 lenguajes de programación, Python, Ruby y Go. Fijándonos en la comparativa que se realiza en la página oficial de Julia [6], en donde se usan varios lenguajes de programación para resolver 5 algoritmos y se miden los tiempos que emplea cada algoritmo para ello. En mi caso voy a hacer algo más simple, usando los 3 lenguajes de programación mencionados anteriormente, vamos a medir el tiempo que tarda cada uno de ellos para resolver el mítico problema de Fibonacci y el algoritmo QuickSort para ordenar un vector de números, y así poder validar los tiempos que se muestran en la página de Julia. De esta forma si nosotros queremos desarrollar algún programa en uno de estos lenguajes, podremos sacar conclusiones para escoger uno u otro.

5.3. Métricas :

Las métricas que utiliza el benchmark son segundos para la ejecución tanto del algoritmo Fibonacci como para el algoritmo QuickSort. Además para el Fibonacci tendremos el dato de la sucesión de Fibonacci que hemos calculado, y para el QuickSort tendremos el tamaño del vector de números que hemos ordenado. En el caso de Fibonacci calculamos desde el 0 hasta el 40, y cada sucesión la calculamos 5 veces para hacer la media sumando los tiempos y dividiéndolos entre 5 (el número de veces que se ha calculado esa sucesión). Y para el caso de QuickSort hemos realizado el mismo proceso, ordenamos un vector de tamaño 7, desde 2 elementos hasta los 7 elementos. Y cada ordenación lo hacemos 5 veces, para obtener la media.

5.4. Instrucciones para su uso :

Para usarlo, mayormente se debe tener en cuenta diversas instalaciones que serán explicadas a continuación:

- Tener instalado Python 3.x. [7]
- Tener instalado Ruby 2.3.3. [8]
- Tener instalado Go 1.7.4. [9]
- Tener instalado gnuplot 5.0. [10]

De tal forma que una vez instaladas todos los componentes anteriormente descritos, debemos ejecutar el script en Python que se mostrará a continuación. Después de ejecutar el script en Pyhton, tendremos unos ficheros con los resultados de las ejecuciones, si queremos que se muestren en una gráfica, debemos ejecutar con gnuplot el script creado para las gráficas.

La ejecución del script en Python se realiza de la siguiente manera:

- `python <nombre del fichero .py>`

La ejecución del script para gnuplot se realiza de la siguiente manera:

- `gnuplot <nombre del script .gnu>`

A continuación se irá mostrando el código con el que se ha realizado la práctica. En primer lugar tenemos un script que realiza las correspondientes llamadas a los programas escritos en Python, Ruby y Go. Y además saca los datos hacia unos ficheros que después serán usados por otro script para sacar las correspondientes gráficas en gnuplot.

```
#!/usr/bin/env python

import sys
import subprocess
import time

print ("Benchmark for Python, Ruby and Go language programming Web")

subprocess.call(['echo', '>', "fibonacciPython.dat"])

subprocess.call(['echo', '>', "fibonacciRuby.dat"])

subprocess.call(['echo', '>', "fibonacciGo.dat"])

p = open('fibonacciPython.dat', 'w')

r = open('fibonacciRuby.dat', 'w')

g = open('fibonacciGo.dat', 'w')

medium_python = []
medium_ruby = []
medium_go = []

for i in range(0, 41, 1):
    medium_python.append(0)
    medium_ruby.append(0)
    medium_go.append(0)

print ("Executing all programs to measure time for fibonacci")

for i in range(0, 5, 1):

    value = 0

    for i in range(0, 41, 1):

        print("PYTHON\n")
```

```

start = time.time()
subprocess.call(['python', 'fibonacciPython.py', str(i)])
end = time.time()
medium_python[value] = medium_python[value] + (end - start)

print("RUBY\n")
start = time.time()
subprocess.call(['ruby', 'fibonacciRuby.rb', str(i)])
end = time.time()
medium_ruby[value] = medium_ruby[value] + (end - start)

print("GO\n")
start = time.time()
subprocess.call(['go', 'run', 'fibonacciGo.go', str(i)])
end = time.time()
medium_go[value] = medium_go[value] + (end - start)

value = value + 1

value = 0

print ("Saving data for fibonacci")

for i in range(0, 41, 1):
    p.write(str(i)+"\n")
    r.write(str(i)+"\n")
    g.write(str(i)+"\n")

    p.write(str(medium_python[value]/5) + "\n")
    r.write(str(medium_ruby[value]/5) + "\n")
    g.write(str(medium_go[value]/5) + "\n")

    value = value + 1

p.close()
r.close()
g.close()

print ("----- Finished Fibonacci, now start
QuickSort -----")
subprocess.call(['echo', '>', "quicksortPython.dat"])

subprocess.call(['echo', '>', "quicksortRuby.dat"])

```

```

subprocess . call ([ 'echo ' , '>' , "quicksortGo . dat" ])

p = open( 'quicksortPython . dat' , 'w')

r = open( 'quicksortRuby . dat' , 'w')

g = open( 'quicksortGo . dat' , 'w')

for i in range(2 , 6 , 1):
    medium_python.append(0)
    medium_ruby.append(0)
    medium_go.append(0)

print ( "Executing all programs to measure time for quicksort" )

for i in range(0 , 5 , 1):

    value = 0

    for i in range(2 , 6 , 1):

        print("PYTHON\n")
        start = time . time()
        subprocess . call ([ 'python ' , 'quicksortPython . py' , str(i)])
        end = time . time()
        medium_python[value] = medium_python[value] + (end - start)

        print("RUBY\n")
        start = time . time()
        subprocess . call ([ 'ruby ' , 'quicksortRuby . rb' , str(i)])
        end = time . time()
        medium_ruby[value] = medium_ruby[value] + (end - start)

        print("GO\n")
        start = time . time()
        subprocess . call ([ 'go ' , 'run ' , 'quicksortGo . go' , str(i)])
        end = time . time()
        medium_go[value] = medium_go[value] + (end - start)

    value = value + 1

value = 0

```

```
print ("Saving_data_for_quicksort")

for i in range(2, 6,1):
    p.write(str(i)+"_")
    r.write(str(i)+"_")
    g.write(str(i)+"_")

    p.write(str(medium_python[value]/5) + "\n")
    r.write(str(medium_ruby[value]/5) + "\n")
    g.write(str(medium_go[value]/5) + "\n")

    value = value + 1

p.close()
r.close()
g.close()

print ("_____Finished_Quicksort ,"
      "now you can view all data_____")
```

En segundo lugar tenemos los distintos algoritmos programados en Python:

```
#Author: Ivan Rodriguez Millan .
#Date: 18/12/2016
#Universidad de Granada.
#Description: Program to calculate the succession of
#fibonacci (Python Language).

#!/usr/bin/env python

import math
import sys

def fib_recursive(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib_recursive(n-1) + fib_recursive(n-2)

fib_recursive(int(sys.argv[1]))

#print(fib_recursive(int(sys.argv[1])))
```

```

#Author: Ivan Rodriguez Millan .
#Date: 18/12/2016
#Universidad de Granada.
#Description: Program to sort by QuickSort (Python Language).

#!/usr/bin/env python

import math
import sys

def quicksort(array , inicio , final):

    pivot = 0

    if inicio >= final:
        return final

    pivot1 = array[inicio]
    limite = inicio
    temp=0

    for i in range(inicio+1,final+1, 1):
        if array[i] < pivot1:
            limite = limite + 1

            temp=array[i]
            array[i]=array[limite]
            array[limite]=temp

    temp=array[inicio]
    array[inicio]=array[limite]
    array[limite]=temp

    pivot=limite

    quicksort(array , inicio , pivot)
    quicksort(array , pivot+1, final)

    return array

array = [102,90,98,93,77,82,83,2,45,0,550,1102,75,74,33,29,44,12,5,3,89]

quicksort(array , 0, int(sys.argv[1]))

```

```
#print(quicksort(array, 0, int(sys.argv[1])))
```

A continuación se muestran los distintos algoritmos desarrollados en Ruby:

```
#Author: Ivan Rodriguez Millan .
#Date: 18/12/2016
#Universidad de Granada.
#Description: Program to calculate the succession of fibonacci
#(Ruby Language).

def fibonacci(n)

    if n == 0
        0
    elsif n == 1
        1
    else
        fibonacci(n-1) + fibonacci(n-2)
    end

end

fibonacci(ARGV[0].to_i)

#puts "#{fibonacci(ARGV[0].to_i)} "
```

```

#Author: Ivan Rodriguez Millan .
#Date: 18/12/2016
#Universidad de Granada.
#Description: Program to sort by QuickSort (Ruby Language).

def quicksort(array , from, to)
  if to == nil

    to = array .count - 1
  end

  if from >= to

    return
  end

  pivot = array [from]

  min = from
  max = to

  free = min

  while min < max
    if free == min
      if array [max] <= pivot
        array [ free ] = array [max]
        min += 1
        free = max
      else
        max -= 1
      end
    elsif free == max
      if array [min] >= pivot
        array [ free ] = array [min]
        max -= 1
        free = min
      else
        min += 1
      end
    else
      "Estado_inconsistente"
    end
  end
end

```

```
end

array[free] = pivot

quicksort array, from, free - 1
quicksort array, free + 1, to
end

a = [102,90,98,93,77,82,83,2,45,0,550,1102,75,74,33,29,44,12,5,3,89]

quicksort a,0,ARGV[0].to_i

#puts a
```

A continuación se muestran los distintos algoritmos desarrollados en Ruby:

```
//Author: Ivan Rodriguez Millan .
//Date: 18/12/2016
//Universidad de Granada.
//Description: Program to calculate the succession of fibonacci
//(Go Language).

package main

import "os"
import "strconv"

func main() {

    //Leemos un valor desde linea de comandos.
    args := os.Args[1]

    //Convertimos ese valor a entero.
    argint, _ := strconv.Atoi(args)

    //Llamamos a la funcion Fibonacci con el valor leido desde linea
    //de comandos. No olvidarse de importar fmt para imprimir el
    //resultado
    //fmt.Println(fibonacci(argint))

    fibonacci(argint)
}

func fibonacci(n int) int {

    if n == 0 {
        return 0
    }else if n == 1 {
        return 1
    }else {
        return fibonacci(n-1) + fibonacci(n-2)
    }
}
```

```

//Author: Ivan Rodriguez Millan.
//Date: 18/12/2016
//Universidad de Granada.
//Description: Program to sort by QuickSort (Go Language).

package main

import "os"
import "strconv"
//import "fmt"

func quickSort (A []int , inicio int , final int) int {
    pivot := 0;

    if(inicio >= final){
        return final;
    }

    pivot1 := A[inicio];
    limite := inicio;
    temp:=0;

    for i:=inicio+1; i <= final ; i++ {
        if(A[ i ] < pivot1){
            limite++;

            temp=A[ i ];
            A[ i ]=A[ limite ];
            A[ limite ]=temp;
        }
    }

    temp=A[ inicio ];
    A[ inicio ]=A[ limite ];
    A[ limite ]=temp;

    pivot=limite;

    quickSort(A, inicio , pivot)
    quickSort(A, pivot+1, final)

    return 1;
}

```

```
func main() {  
    args := os.Args[1]  
  
    //Convertimos ese valor a entero.  
    argint, _ := strconv.Atoi(args)  
  
    A := []int{102,90,98,93,77,82,83,2,45,0,550,1102,75,74,33,29,44,12  
    ,5,3,89};  
  
    quickSort(A,0,argint);  
    //No olvidarse de importar fmt para imprimir el resultado  
    //fmt.Println(A);  
}
```

Y por último se mostrará el script realizado para sacar las gráficas con gnuplot:

```
set terminal svg
set title 'Fibonacci'
set output 'FibonacciBenchmarkLines.svg'
set xlabel 'size_of_data_to_calculate'
set ylabel 'execution_time_(s)'
set xrange [0:45]
plot 'fibonacciPython.dat' using 1:2 title 'Python' with lines ,
'fibonacciRuby.dat' using 1:2 title 'Ruby' with lines ,
'fibonacciGo.dat' using 1:2 title 'Go' with lines

set terminal svg
set title 'Quicksort'
set output 'QuicksortBenchmarkLines.svg'
set xlabel 'size_of_data_to_sort'
set ylabel 'execution_time_(s)'
set xrange [0:25]
plot 'quicksortPython.dat' using 1:2 title 'Python' with lines ,
'quicksortRuby.dat' using 1:2 title 'Ruby' with lines ,
'quicksortGo.dat' using 1:2 title 'Go' with lines
```

Nota: Por último cabe decir que se ha intentado que los códigos de los programas en cada lenguaje fueran parecidos para no alterar la comparativa entre ellos, aunque obviamente me he encontrado con los problemas de no conocer nada de Go, y muy poco de Ruby y Python.

5.5. Ejemplo de uso analizando los resultados :

Primero ejecutamos el script en Python con el comando que hemos dejado arriba. Este script primero ejecuta los tres programas de Fibonacci en los diferentes lenguajes, el primero en ejecutar será el escrito en Python ya que como el propio script está hecho en Python, el interprete estará cargado y como sabemos que el primero será el que más carga tenga pues lo compensamos.

Después de ejecutar los programas de Fibonacci, se pasará a ejecutar los programas de QuickSort siguiendo los mismos pasos que antes. El script nos irá diciendo en cada momento lo que se está realizando.

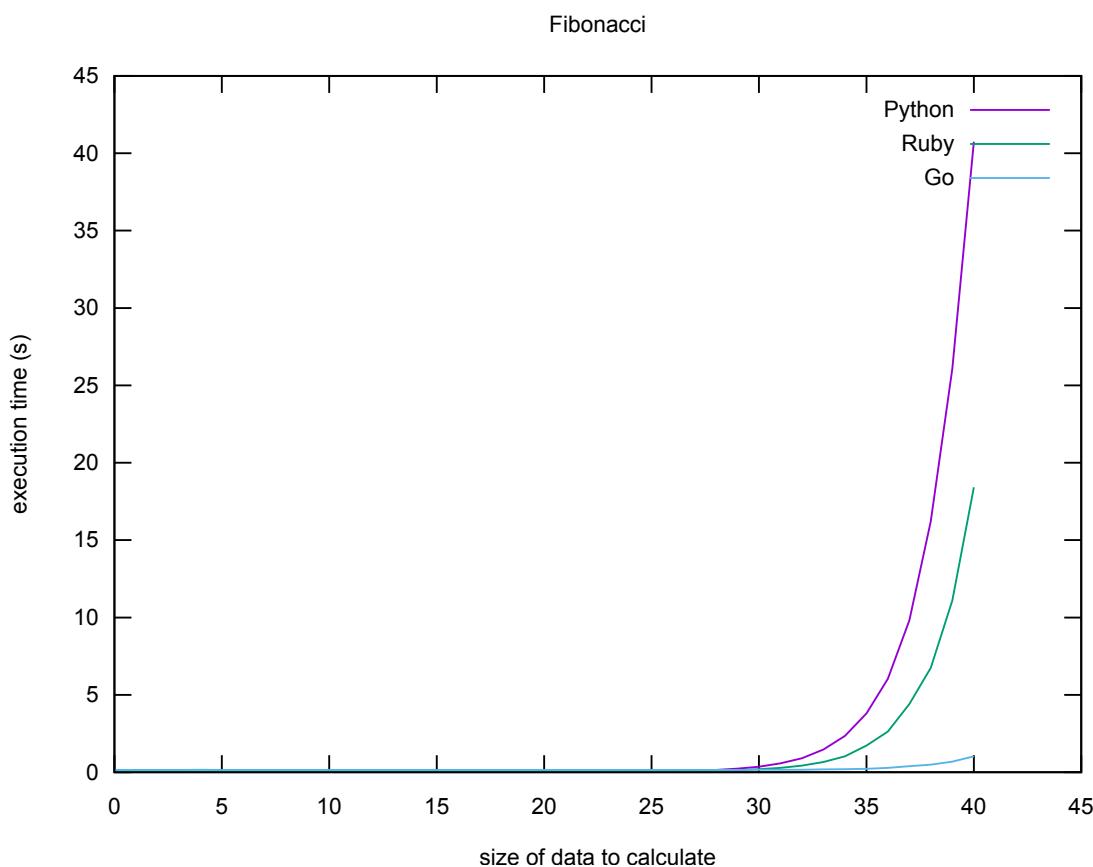


Figura 5.1: Gráfica de los resultados de ejecución de Fibonacci calculando desde el 0 hasta el 40.

En la gráfica podemos ver como la ejecución en Go es notablemente inferior en cuanto a tiempo se refiere. Tanto es así que al calcular el Fibonacci de 40 en Python sube hasta los 40 segundos y en Go se mantiene en los 2.5 segundos. En Ruby el tiempo al calcular

el Fibonacci de 40 aumenta hasta los 20 segundos.
 Para el cálculo de Fibonacci anterior a 35 que es donde Python verdaderamente empeora sustancialmente, los tres programas se comportan de forma similar.

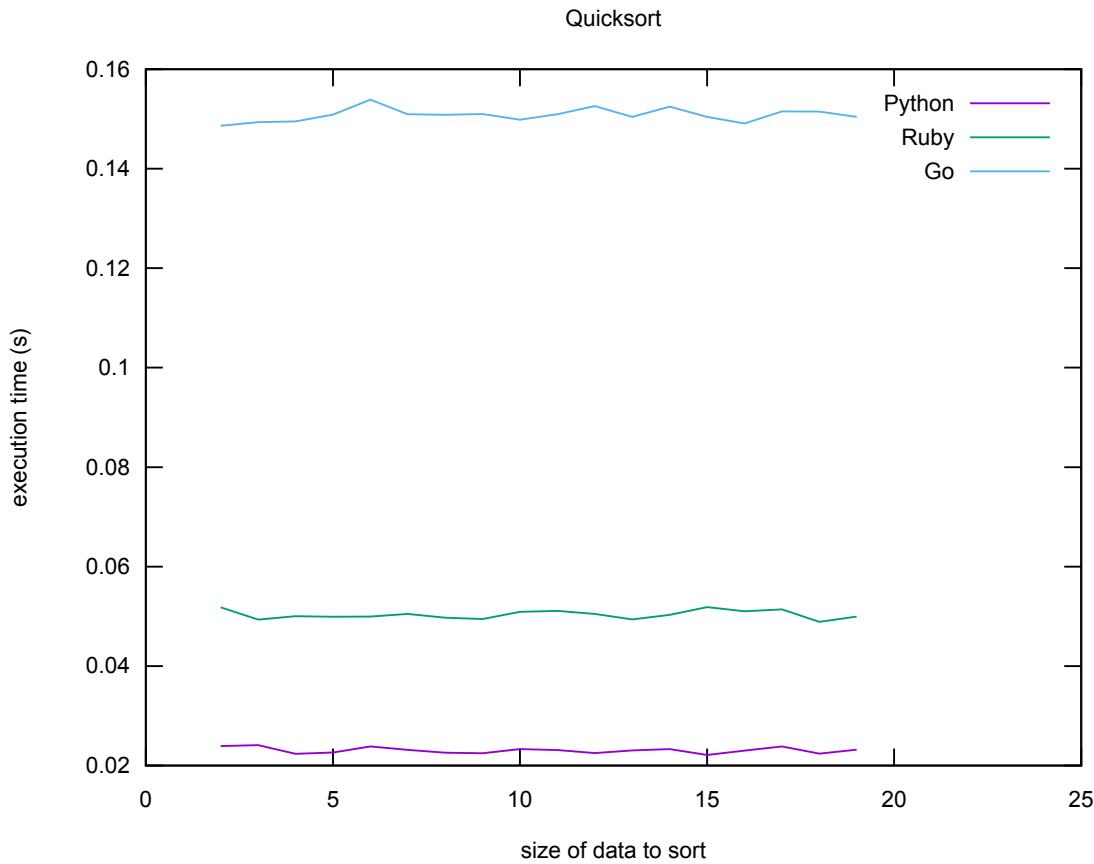


Figura 5.2: Gráfica de los resultados de ejecución de QuickSort ordenando un vector desde 2 componentes hasta 20.

En la gráfica 5.2 se observa como ocurre lo contrario que en la gráfica , ya que ahora es Python quién apenas supera los 0.02 segundos y Go llega hasta los 0.15 segundos. En este caso ocurre algo simbólico, pues en la referencia [6] vemos que Python empeoraba bastante en comparación con Go en cuanto al algoritmo QuickSort se refiere.

En el caso de Ruby se mantiene bastante cerca de Python con unos 0.06 segundos aproximadamente. Si es verdad hablando de la comparativa entre Python y Go, que las diferencias en el algoritmo QuickSort experimentadas en este estudio, son bastante inferiores en comparación con el algoritmo de Fibonacci.

Luego podemos dar un gran voto de confianza a que Go en un futuro de la mano de Google se imponga a lenguajes como Python en donde hay detrás una gran comunidad

hoy en día. Aunque viendo los resultados tanto de este benchmark como los de la página oficial de Julia, no parece nada descabellado usar actualmente Go.

Referencias

- [1] “<http://www.phoronix-test-suite.com/documentation/phoronix-test-suite.pdf>,” consultado el 14 de Diciembre de 2016.
- [2] “<http://openbenchmarking.org/result/1612145-S0-RESULTADO30>,” consultado el 14 de Diciembre de 2016.
- [3] “<https://httpd.apache.org/docs/2.4/programs/ab.html>,” consultado el 14 de Diciembre de 2016.
- [4] “http://jmeter.apache.org/download_jmeter.cgi,” consultado el 16 de Diciembre de 2016.
- [5] “<http://jmeter.apache.org/usermanual/generating-dashboard.html>,” consultado el 18 de Diciembre de 2016.
- [6] “<http://julialang.org/>,” consultado el 17 de Diciembre de 2016.
- [7] “<https://www.python.org/downloads/>,” consultado el 18 de Diciembre de 2016.
- [8] “<https://www.ruby-lang.org/es/documentation/installation/>,” consultado el 18 de Diciembre de 2016.
- [9] “<https://golang.org/>,” consultado el 18 de Diciembre de 2016.
- [10] “<http://www.gnuplot.info/>,” consultado el 18 de Diciembre de 2016.