

Ingeniería de Servidores (2015-2016)
GRADO EN INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE GRANADA

Memoria Práctica 4

Javier Izquierdo Vera

21 de diciembre de 2015

Índice

1. Instale la aplicación. ¿Qué comando permite listar los benchmarks disponibles?	3
2. Seleccione, instale y ejecute uno, comente los resultados. Atención: no es lo mismo un benchmark que una suite, instale un benchmark.	4
3. De los parámetros que le podemos pasar al comando ¿Qué significa -c 5 ? ¿y -n 100? Monitoree la ejecución de ab contra alguna máquina (cualquiera) ¿cuántos procesos o hebras crea ab en el cliente?	5
4. 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) y muestre y comente las estadísticas. ¿Cuál es la que proporciona mejores resultados? Fíjese en el número de bytes transferidos, ¿es igual para cada máquina?	7
5. ¿Qué es Scala? Instale Gatling y pruebe los escenarios por defecto.	12
5.1. ¿Qué es Scala?	12
5.2. Instale Gatling y pruebe los escenarios por defecto.	13
6. Lea el artículo y elabore un breve resumen.	15
7. 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 alguna de sus máquinas virtuales (Puede hacer una página sencilla, usar las páginas de phpmyadmin, instalar un CMS, etc.).	15
8. Seleccione un benchmark entre SisoftSandra y Aida. Ejecútelo y muestre capturas de pantalla comentando los resultados.	20
9. 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	24
9.1. Objetivo del benchmark	24
9.2. Métricas	24
9.3. Instrucciones para su uso	25
9.4. Ejemplo de uso analizando los resultados	27
9.5. Implementación	27

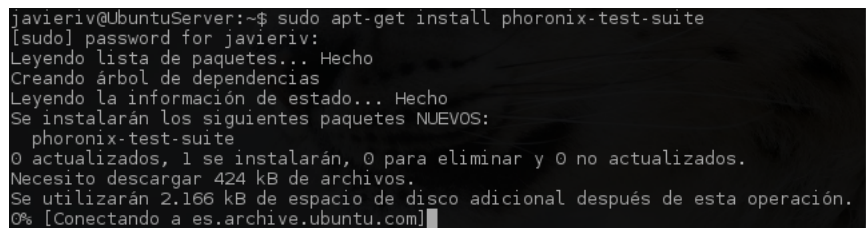
Índice de figuras

1.1. Instalando Phoronix Test Suite en Ubuntu Server mediante apt	3
1.2. Benchmarks disponibles en Phoronix Test Suite	3
2.1. Instalando benchmark pts/openssl en Phoronix Test Suite	4
2.2. Faltan dependencias en un benchmark en Phoronix Test Suite	4
2.3. Resultados del benchmark pts/openssl en Phoronix Test Suite	5
3.1. Parámetros que acepta Apache Benchmark	5
3.2. Número de hebras que genera el proceso ab para conexiones simulando distintos usuarios	6
5.1. Instalando Scala en Kali Linux 2.0 mediante el gestor de paquetes apt . .	12
5.2. Ejemplo sencillo del lenguaje Scala	13
5.3. Iniciando Gatling, este compila los escenarios y los pregunta cuál deseamos utilizar	13
5.4. Probando un escenario de Gatling	14
5.5. Resultados de una prueba realizada con Gatling	14
7.1. Instalando JMeter mediante el gestor de paquetes apt en Kali Linux 2.0 .	15
7.2. Configurando grupo de hilos en JMeter	16
7.3. Configurando petición HTTP que tiene por defecto JMeter	16
7.4. Configurando petición HTTP a JMeter	17
7.5. Gráfico generado por JMeter de una prueba realizada al login de phpm- yadmin en Ubuntu Server	17
7.6. Reporte del resumen generado por JMeter de una prueba realizada al login de phpmyadmin en Ubuntu Server	18
7.7. Gráfico generado por JMeter de una prueba realizada al login de phpm- yadmin en Ubuntu Server	18
7.8. Reporte del resumen generado por JMeter de una prueba realizada al login de phpmyadmin en Ubuntu Server	19
7.9. Errores mostrados en el gráfico generado por JMeter de una prueba reali- zada al login de phpmyadmin en Ubuntu Server	19
8.1. Seleccionando directorio en el que se instalará Aida64 en Windows Server 2008 R2	20
8.2. Instalación finalizada de Aida64 en Windows Server 2008 R2	21
8.3. Instalación finalizada de Aida64 en Windows Server 2008 R2	21
8.4. Tipos de benchmarks que ofrece Aida64	22
8.5. Benchmarks de disco que ofrece Aida64	22
8.6. Realizando benchmark de lecturas en disco utilizando Aida64	23
8.7. Resultado del benchmark de lecturas en disco realizao con Aida64	23
9.1. Pantalla inicial del benchmark mostrando advertencia	25
9.2. Pantalla inicial del benchmark	26
9.3. Realizando el benchmark al dispositivo	26
9.4. Resultados obtenidos por el benchmark	27

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

Se realizará la instalación de Phoronix Test Suite siguiendo el manual de Ubuntu [1]. Tal como nos indica, es posible instalarlo mediante el gestor de paquetes apt, el paquete ya se encuentra en los repositorios.

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



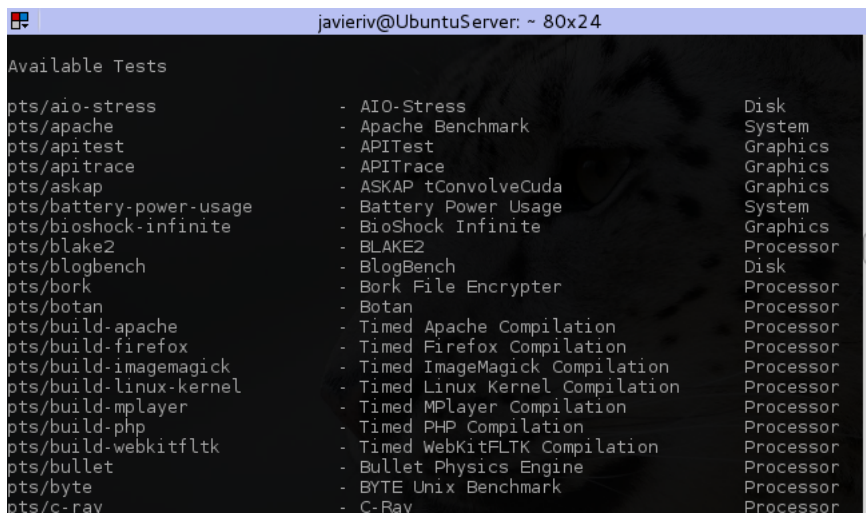
```
javieriv@UbuntuServer:~$ sudo apt-get install phoronix-test-suite
[sudo] password for javieriv:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes NUEVOS:
  phoronix-test-suite
0 actualizados, 1 se instalarán, 0 para eliminar y 0 no actualizados.
Necesito descargar 424 kB de archivos.
Se utilizarán 2.166 kB de espacio de disco adicional después de esta operación.
0% [Conectando a es.archive.ubuntu.com]
```

Figura 1.1: Instalando Phoronix Test Suite en Ubuntu Server mediante apt

Para listar los benchmarks disponibles se usa el siguiente comando (tal como se indica en el manual anteriormente citado):

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

Al utilizarlo nos muestra una larga lista de benchmarks.



```
javieriv@UbuntuServer: ~ 80x24

Available Tests

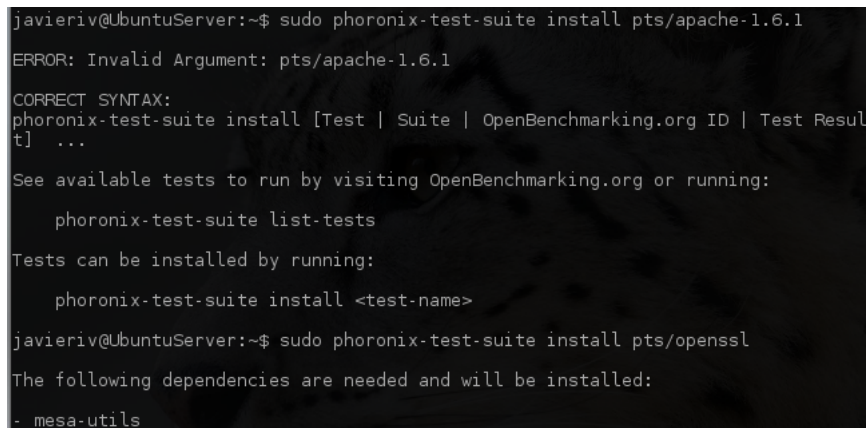
pts/aio-stress          - AIO-Stress          Disk
pts/apache              - Apache Benchmark    System
pts/apitest             - APITest             Graphics
pts/apitrace            - APITrace            Graphics
pts/askap               - ASKAP tConvolveCuda  Graphics
pts/battery-power-usage - Battery Power Usage  System
pts/bioshock-infinite   - BioShock Infinite   Graphics
pts/blake2              - BLAKE2              Processor
pts/blogbench           - BlogBench           Disk
pts/bork                - Bork File Encrypter  Processor
pts/botan              - Botan               Processor
pts/build-apache        - Timed Apache Compilation Processor
pts/build-firefox       - Timed Firefox Compilation Processor
pts/build-imagemagick   - Timed ImageMagick Compilation Processor
pts/build-linux-kernel - Timed Linux Kernel Compilation Processor
pts/build-mplayer       - Timed MPlayer Compilation Processor
pts/build-php           - Timed PHP Compilation Processor
pts/build-webkitgtk     - Timed WebKitGTK Compilation Processor
pts/bullet              - Bullet Physics Engine Processor
pts/byte                - BYTE Unix Benchmark  Processor
pts/c-ray               - C-Ray               Processor
```

Figura 1.2: Benchmarks disponibles en Phoronix Test Suite

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

Se ha seleccionado *pts/openssl*, para medir el rendimiento del cifrado de OpenSSL. [2] Una vez más, siguiendo el manual [1], utilizaremos la orden *install* en Phoronix Test Suite.

```
sudo phoronix-test-suite install pts/openssl
```



```
javieriv@UbuntuServer:~$ sudo phoronix-test-suite install pts/apache-1.6.1
ERROR: Invalid Argument: pts/apache-1.6.1

CORRECT SYNTAX:
phoronix-test-suite install [Test | Suite | OpenBenchmarking.org ID | Test Result] ...

See available tests to run by visiting OpenBenchmarking.org or running:

    phoronix-test-suite list-tests

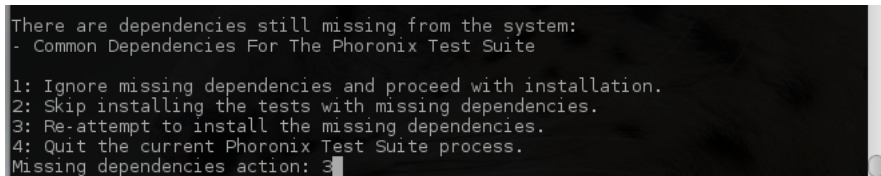
Tests can be installed by running:

    phoronix-test-suite install <test-name>

javieriv@UbuntuServer:~$ sudo phoronix-test-suite install pts/openssl
The following dependencies are needed and will be installed:
- mesa-utils
```

Figura 2.1: Instalando benchmark pts/openssl en Phoronix Test Suite

Durante la instalación nos indica que faltan dependencias, le indicamos que las instale.



```
There are dependencies still missing from the system:
- Common Dependencies For The Phoronix Test Suite

1: Ignore missing dependencies and proceed with installation.
2: Skip installing the tests with missing dependencies.
3: Re-attempt to install the missing dependencies.
4: Quit the current Phoronix Test Suite process.
Missing dependencies action: 3
```

Figura 2.2: Faltan dependencias en un benchmark en Phoronix Test Suite

Una vez instalado procedemos a ejecutarlo, para ello utilizaremos la siguiente orden:

```
sudo phoronix-test-suite run pts/openssl
```

Tras ejecutarlo nos pregunta si deseamos guardar los resultados, seleccionamos que sí e indicamos un nombre para el archivo con los resultados. Tras lo anterior ejecutamos el benchmark indicando una estimación de tiempo.

Este es el resultado:

```

OpenSSL 1.0.1g:
pts/openssl-1.9.0
Test 1 of 1
Estimated Trial Run Count:    3
Estimated Time To Completion: 2 Minutes
Started Run 1 @ 18:34:23

Started Run 2 @ 18:34:45
Started Run 3 @ 18:35:07 [Std. Dev: 1.26%]

Test Results:
95.8
97.9
95.8

Average: 96.50 Signs Per Second

```

Figura 2.3: Resultados del benchmark pts/openssl en Phoronix Test Suite

Se muestran tres resultados diferentes, lo que quiere decir que se han efectuado tres pruebas. En las tres se puestran resultados parecidos. La máquina en la que se ha probado el benchmark es capaz de realizar alrededor de 96.5 firmas por segundo.

3. De los parámetros que le podemos pasar al comando ¿Qué significa `-c 5` ? ¿y `-n 100`? Monitoree la ejecución de `ab` contra alguna máquina (cualquiera) ¿cuántos procesos o hebras crea `ab` en el cliente?

Escribiendo `textitab` podemos ver la lista de parámetros que acepta la orden.

```

javieriv@UbuntuServer:~$ ab
ab: wrong number of arguments
Usage: ab [options] [http[s]://[hostname[:port]]/path]
Options are:
  -n requests      Number of requests to perform
  -c concurrency   Number of multiple requests to make at a time
  -t timelimit      Seconds to max. to spend on benchmarking
                   This implies -n 50000
  -s timeout        Seconds to max. wait for each response
                   Default is 30 seconds
  -b window size    Size of TCP send/receive buffer, in bytes
  -B address        Address to bind to when making outgoing connections
  -p postfile       File containing data to POST. Remember also to set -T
  -u putfile        File containing data to PUT. Remember also to set -T
  -T content-type   Content-type header to use for POST/PUT data, eg.
                   'application/x-www-form-urlencoded'
                   Default is 'text/plain'
  -v verbosity      How much troubleshooting info to print
  -w               Print out results in HTML tables
  -i               Use HEAD instead of GET
  -x attributes     String to insert as table attributes
  -y attributes     String to insert as tr attributes
  -z attributes     String to insert as td or th attributes
  -C attribute      Add cookie, eg. 'Apache=1234' (repeatable)

```

Figura 3.1: Parámetros que acepta Apache Benchmark

Según podemos ver en la ayuda, la orden `ab -c 5` permite usar el benchmark con una concurrencia de 5, es decir, realizando 5 peticiones a la vez. Según se indica, con la orden `-n` es posible indicar el número de peticiones que se van a realizar, de modo que la orden

`ab -n 100` realizaría 100 peticiones.

Para determinar el número de hebras que genera la orden `ab`, la ejecutaremos para un número de peticiones elevado, de modo que tengamos tiempo para ver el número de hebras que pertenecen activas.

Ejecutamos la orden:

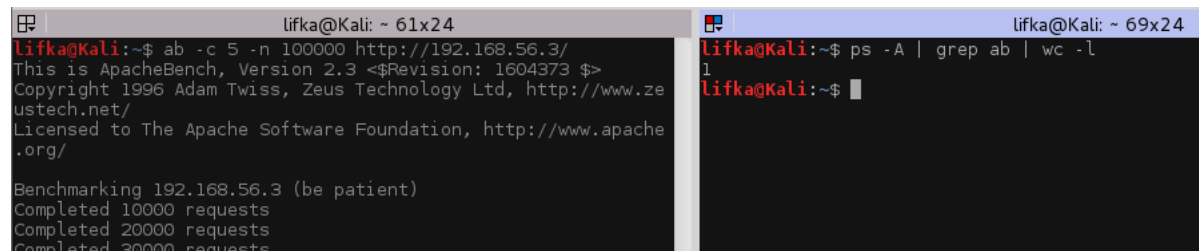
```
ab -c 5 -n 100000 http://192.168.56.3/
```

La dirección es la de la máquina Ubuntu Server, en la que se encuentra Apache 2 en ejecución.

Para comprobar el número de hebras activas utilizaremos el comando `ps` [13], utilizando la orden `grep`, la cual vimos en la práctica anterior. También utilizaremos `wc`, para contar el número de líneas [12].

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

El resultado es el siguiente:



The image shows two side-by-side terminal windows from a Kali Linux machine. The left window, titled 'lifka@Kali: ~ 61x24', shows the output of the command 'ab -c 5 -n 100000 http://192.168.56.3/'. The output includes the ApacheBench version (2.3), copyright information (1996 Adam Twiss, Zeus Technology Ltd), and benchmarking progress for 100,000 requests. The right window, titled 'lifka@Kali: ~ 69x24', shows the output of the command 'ps -A | grep ab | wc -l', which returns the number '1', indicating that only one thread is active.

Figura 3.2: Número de hebras que genera el proceso `ab` para conexiones simulando distintos usuarios

Podemos ver como `ab` solo crea una hebra, a pesar de que hayamos indica que lo haga con cierta concurrencia. `ab` genera las conexiones concurrentes sin necesidad de crear más de una hebra.

4. 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) y muestre y comente las estadísticas. ¿Cuál es la que proporciona mejores resultados? Fíjese en el número de bytes transferidos, ¿es igual para cada máquina?

Para realizar este análisis se han cerrado todas las tareas posibles en la máquina anfitriona, para intetar que no afecte a los resultados. Procedemos a lanzar *ab* contra las tres máquinas virtual.

```
ab -c 5 -n 500 http://192.168.56.MAQUINA/
```

Resultado para Ubuntu Server:

```
ab -c 5 -n 500 http://192.168.56.3/
This is ApacheBench, Version 2.3 <$Revision: 1604373 $>

Benchmarking 192.168.56.3 (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Finished 500 requests


Server Software:      Apache/2.4.7
Server Hostname:      192.168.56.3
Server Port:          80

Document Path:        /
Document Length:       11510 bytes

Concurrency Level:     5
Time taken for tests:   0.335 seconds
Complete requests:     500
Failed requests:        0
Total transferred:     5891500 bytes
HTML transferred:      5755000 bytes
Requests per second:   1492.52 [#/sec] (mean)
Time per request:       3.350 [ms] (mean)
Time per request:       0.670 [ms] (mean, across all
```


concurrent requests)					
Transfer rate:	17174.13 [Kbytes/sec] received				
Connection Times (ms)					
	min	mean[+/-sd]	median	max	
Connect:	0	0 0.1	0	1	
Processing:	1	3 3.3	2	26	
Waiting:	0	2 1.8	2	23	
Total:	1	3 3.3	2	26	
Percentage of the requests served within a certain time (ms)					
50 %	2				
66 %	3				
75 %	4				
80 %	5				
90 %	6				
95 %	10				
98 %	15				
99 %	17				
100 %	26 (longest request)				

Para 500 peticiones con una concurrencia igual a 5, el resultado es que todas las peticiones se satisfacen correctamente en 0.335 segundos. El tiempo por cada petición es de 3.350 milisegundos, al dividir esa cifra entre las 5 que se están realizando a la vez, el resultado es que por cada petición se tarda de media 0.670 milisegundos.

Resultado para Windows Server 2008 R2:

ab -c 5 -n 500 http://192.168.56.3/	
This is ApacheBench, Version 2.3 <\$Revision: 1604373 \$>	
Benchmarking 192.168.56.3 (be patient)	
Completed 100 requests	
Completed 200 requests	
Completed 300 requests	
Completed 400 requests	
Completed 500 requests	
Finished 500 requests	
Server Software:	Microsoft-IIS/7.5
Server Hostname:	192.168.56.3
Server Port:	80

Document Path:	/
Document Length:	689 bytes
Concurrency Level:	5
Time taken for tests:	3.937 seconds
Complete requests:	500
Failed requests:	0
Total transferred:	466000 bytes
HTML transferred:	344500 bytes
Requests per second:	126.99 [# /sec] (mean)
Time per request:	39.374 [ms] (mean)
Time per request:	7.875 [ms] (mean, across all concurrent requests)
Transfer rate:	115.58 [Kbytes/sec] received
Connection Times (ms)	
	min mean[+/-sd] median max
Connect:	0 30 298.9 0 3001
Processing:	1 9 69.5 2 701
Waiting:	1 9 69.5 2 701
Total:	1 39 368.4 2 3703
Percentage of the requests served within a certain time (ms)	
50 %	2
66 %	2
75 %	2
80 %	3
90 %	3
95 %	4
98 %	18
99 %	3700
100 %	3703 (longest request)

Igual que en el caso anterior hemos realizado el test para 500 peticiones con una concurrencia de 5, el resultado es que todas las peticiones se han realizado correctamente para un tiempo de 3.937 segundos (mucho más alto que para Ubuntu Server). Se ha tardado 39.374 milisegundos para cada petición, una media de 7.875 si tenemos en cuenta la concurrencia. El tiempo que ha tardado es muy superior al de Ubuntu Server.

Resultado para CentOS:

```
ab -c 5 -n 500 http://192.168.56.4/
This is ApacheBench, Version 2.3 <$Revision: 1604373 $>
```

Benchmarking 192.168.56.4 (be patient)

Completed 100 requests

Completed 200 requests

Completed 300 requests

Completed 400 requests

Completed 500 requests

Finished 500 requests

Server Software: Apache/2.4.6

Server Hostname: 192.168.56.4

Server Port: 80

Document Path: /

Document Length: 4897 bytes

Concurrency Level: 5

Time taken for tests: 0.613 seconds

Complete requests: 500

Failed requests: 0

Non-2xx responses: 500

Total transferred: 2589500 bytes

HTML transferred: 2448500 bytes

Requests per second: 815.32 [#/sec] (mean)

Time per request: 6.133 [ms] (mean)

Time per request: 1.227 [ms] (mean, across all concurrent requests)

Transfer rate: 4123.56 [Kbytes/sec] received

Connection Times (ms)

	min	mean[+/-sd]	median	max
Connect:	0	0 0.5	0	6
Processing:	1	6 11.0	3	120
Waiting:	1	4 4.3	3	27
Total:	1	6 11.0	3	120

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

50 % 3

66 % 3

75 % 5

80 % 8

90 % 11

95 % 17

98 % 27

99 %	87
100 %	120 (longest request)

El resultado para 500 peticiones con una concurrencia de 5 es una respuesta exitosa para todas las peticiones con un tiempo total de 0.613 segundos. Es un poco más lento que Ubuntu Server, pero bastante más rápido que Windows Server. Para cada petición tarda 6.133 milisegundos (casi el doble que en Ubuntu Server), y teniendo en cuenta la concurrencia, una media de 1.227 milisegundos por petición.

A pesar de que unas máquinas, aparentemente, son más rápidas que otras, esto no es del todo así, ya que depende de muchas otras variables. Por ejemplo, cada una de ellas descarga una cantidad de bytes diferente, debido a que cada servidor web alberga una web de distinto tamaño. También depende de la red, no solo de la máquina (aunque en este caso la red sea la misma). Además, viendo el porcentaje de respuestas satisfechas en función del tiempo, vemos como, por ejemplo, CentOS ha sido capaz de resolver el 99 % de las peticiones en 87 ms, pero ha necesitado 33ms solo para un 1 %. Finjándonos en el resultado de Windows Server vemos como esto ha sido aún más notable, por ello el resultado de unas peticiones rápidas, se ha visto alterado por culpa de un pequeño porcentaje de peticiones lentas, por lo que no es una medida objetiva.

El resultado es que para los sistemas y las configuraciones actuales, si solo tenemos en cuenta el tiempo que ha tardado (a pesar de ser poco objetivo) Ubuntu Server es el servidor web más rápido, seguido por CentOS, con un resultado de casi 2 veces más tiempo que Ubuntu Server, y por último por Windows Server con un tiempo de varios ordenes por encima de los anteriores.

Para Ubuntu Server se han transferido 5.891.500 bytes, para Windows Server 466.000 bytes, y para CentOS 2.589.500 bytes. Como puede verse los bytes transferidos por el servidor son diferentes en cada máquina, esto es debido a que la web que envía cada máquina es diferente, por ejemplo, la web que alberga CentOS es la predeterminada que se instala con el servicio, y esta muestra unos mensajes con ayuda, sin embargo la de Windows Server es una imagen indicando que está funcionando IIS. [4]

Teniendo en cuenta que los bytes transferidos por cada uno son diferentes, podemos ver cuántos bytes por segundo es capaz de transferir cada máquina. Ubuntu Server sería la más veloz con 7174.13 Kbytes/sec, seguida de CentOS con 4123.56 Kbytes/sec, y por último Windows Server con 115.58 Kbytes/sec. Pero esto sigue sin ser objetivo, ya que el documento que solicitaba cada una era de un tamaño diferente, además de que alguna petición más lenta puede haber alterado el resultado.

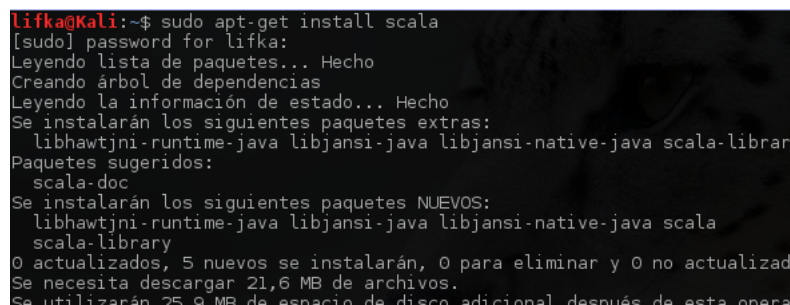
Es muy complicado determinar cuál es la que proporciona mejores resultados, ya que depende de muchas variables. De cara a seleccionar una la pregunta más correcta que deberíamos hacernos es: ¿qué resultados son mejores para satisfacer nuestro objetivo con mayor éxito?

5. ¿Qué es Scala? Instale Gatling y pruebe los escenarios por defecto.

5.1. ¿Qué es Scala?

Scala es un lenguaje de programación con dos paradigmas: es orientado a objetos y funcional. Se ejecuta en la máquina virtual java (JVM) y permite descubrir la mayoría de los errores durante la compilación, por lo que es útil para programar código delicado. Tiene algunas peculiaridades, como que las funciones en Scala son objetos en sí mismas. Tiene similitudes con Java. [5, 7]

Vamos a realizar un ejemplo con Scala. En primer lugar se ha realizado su instalación mediante el gestor de paquetes apt, el paquete ya se encontraba en los repositorios oficiales de *Kali Linux 2.0* (máquina utilizada).



```
lifka@kali:~$ sudo apt-get install scala
[sudo] password for lifka:
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 nuevos se instalarán, 0 para eliminar y 0 no actualizados
Se necesita descargar 21,6 MB de archivos.
Se utilizarán 25,9 MB de espacio de disco adicional después de esta operación.
```

Figura 5.1: Instalando Scala en Kali Linux 2.0 mediante el gestor de paquetes apt

Realizaremos la prueba utilizando el intérprete de Scala. [6]

```
object holaMundo {
  def main(args: Array[String]) {
    println("Hola ISE")
  }
}
```

```

lifka@kali:~$ scala
Welcome to Scala version 2.9.2 (OpenJDK 64-Bit Server VM, Java 1.7.0_91).
Type in expressions to have them evaluated.
Type :help for more information.

scala> object holaMundo {
|   def main(args: Array[String]) {
|       println("Hola ISE")
|   }
| }
defined module holaMundo

scala> holaMundo.main(Array.empty)
Hola ISE

scala>

```

Figura 5.2: Ejemplo sencillo del lenguaje Scala

5.2. Instale Gatling y pruebe los escenarios por defecto.

Gatling está basado en Scala e implementa sus escenarios en ese lenguaje, vamos a probar uno de sus escenarios por defecto. [3]

Siguiendo los manuales de Gatling [8], procedemos a descargar Gatling: <http://gatling.io/#/download>

Tras descomprimirlo nos encontramos con varios directorios, en *bin* se encuentran los ejecutables. Ejecutaremos */bin/gatling.sh*

```
./bin/gatling.sh
```

Esto provoca que Gatling compile los escenarios en Scala, y nos pregunte cuál de ellos deseamos utilizar.

```

root@kali:/home/lifka/Escritorio/gatling-charts-highcharts-bundle-2.1.7/bin# ./gatling.sh
GATLING_HOME is set to /home/lifka/Escritorio/gatling-charts-highcharts-bundle-2.1.7
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 5.3: Iniciando Gatling, este compila los escenarios y los pregunta cuál deseamos utilizar

A modo de ejemplo seleccionaremos el escenario por defecto 0. Tras seleccionarlo se ejecutará la prueba y los resultados se guardarán en el directorio */results*. Del siguiente modo:

```
/results/[ESCENARIO]-[ID]/index.html
```

```

=====
2015-12-12 00:27:18                                     23s elapsed
---- Scenario Name -----
[#####] 100%
waiting: 0 / active: 0 / done:1
----- Requests -----
v Global (OK=13 KO=0 )
v request_1 (OK=1 KO=0 )
v request_1 Redirect 1 (OK=1 KO=0 )
v request_2 (OK=1 KO=0 )
v request_3 (OK=1 KO=0 )
v request_4 (OK=1 KO=0 )
v request_4 Redirect 1 (OK=1 KO=0 )
v request_5 (OK=1 KO=0 )
v request_6 (OK=1 KO=0 )
v request_7 (OK=1 KO=0 )
v request_8 (OK=1 KO=0 )
v request_9 (OK=1 KO=0 )
v request_10 (OK=1 KO=0 )
v request_10 Redirect 1 (OK=1 KO=0 )
=====
Simulation finished
Parsing log file(s)...

```

Figura 5.4: Probando un escenario de Gatling

Si consultamos los resultados, nos encontraremos con una serie de gráficas indicando el rendimiento de la máquina registrado durante el test.

> Global Information

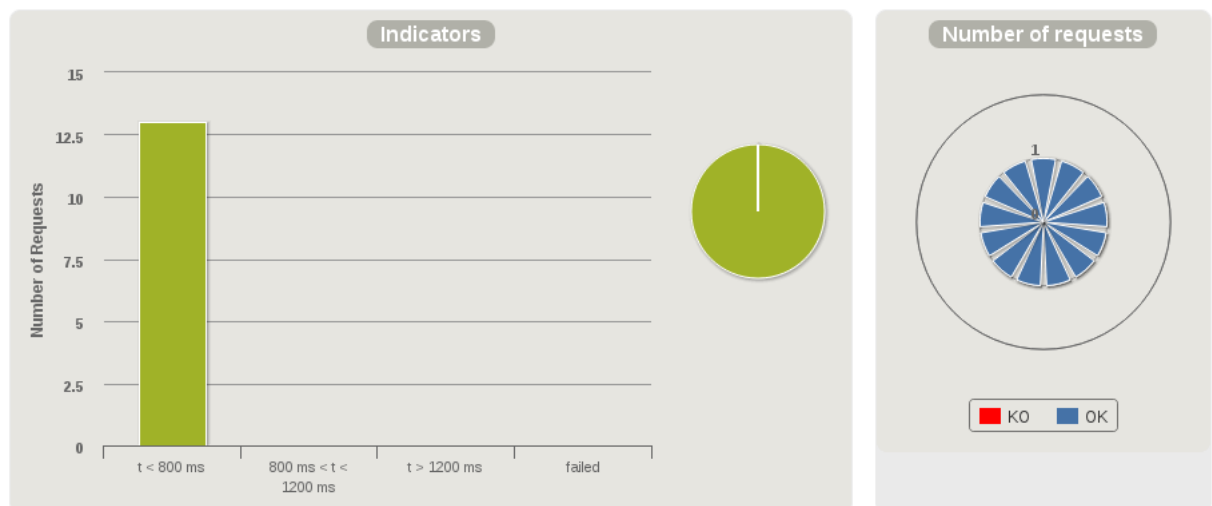


Figura 5.5: Resultados de una prueba realizada con Gatling

Como puede verse en la imagen, la máquina ha respondido positivamente a todas las peticiones en menos de 800 milisegundos.

6. Lea el artículo y elabore un breve resumen.

En el artículo se comenta la utilidad de estas herramientas diciendo que son dos de las más conocidas en el mundo de los benchmarks. Posteriormente se procede a simular un ejemplo práctico, para ello se utiliza un nginx como servidor web configurado de forma que simule una web normal. A pesar de ello hay que tener en cuenta que el resultado puede ser distinto en escenarios realistas.

Se realiza la prueba durante 20 minutos con Gatling y con dos versiones de JMeter, utilizando los mismos parámetros para los tres: 10.000 usuarios y 30.000 peticiones por minuto. Los resultados son bastante parecidos, pero es necesario realizar una comparación.

Gatling no muestra el tamaño de respuesta en bytes, por lo que en el artículo se realiza una aproximación que puede no ser exacta a la realidad. JMeter es más pesado en la máquina virtual java que Gatling, por ello se utiliza un recolector de basura externo, además JMeter es más pesado en la memoria y utiliza más la CPU. Ambos son capaces de mantener un buen rendimiento y de realizar las 10.000 peticiones de usuarios concurrentes sin problema. Además, ambos también tienen un comportamiento correcto en cuanto al uso del caché, y ambos permiten usar expresiones regulares.

La conclusión es que la elección entre ambas es subjetiva, y varía en función de la comodidad por las herramientas que nos proporciona cada herramienta.

7. 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 alguna de sus máquinas virtuales (Puede hacer una página sencilla, usar las páginas de phpmyadmin, instalar un CMS, etc.).

Se realizará la instalación en una máquina con Kali Linux 2.0 mediante el gestor de paquetes apt.

```
root@kali:~# apt-get install jmeter
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes extras:
  ant ant-optional aspectj fop icc-profiles-free jmeter-help jmeter-http
  libapache-pom-java libaspectj-java libavalon-framework-java libbatik-java
  libbcmail-java libbcprov-java libbsf-java
  libcommons-codec-java libcommons-collections3-java
  libcommons-httpclient-java libcommons-io-java libcommons-jexl-java
  libcommons-jexl2-java libcommons-lang3-java libcommons-logging-java
  libcommons-parent-java libdom4j-java libexcalibur-logger-java
```

Figura 7.1: Instalando JMetter mediante el gestor de paquetes apt en Kali Linux 2.0


```
sudo apt-get install jmeter
```

Una vez completada la instalación procedemos a llevar a cabo el tutorial que se indica.

Comenzamos abriendo la interfaz mediante el comando:

```
jmeter
```

Ahora crearemos un “Thread Group”, lo cual sirve para indicar el número de usuarios (conurrencia). Para ello seleccionamos el “ancho de pruebas” y vamos al menú “Editar”, desde ahí aparecerá una opción “Añadir” que nos permite añadir un nuevo grupo de hilos. Podremos configurar el número de usuarios (lo pondremos a 5), y el número de veces que se va a repetir la prueba (se pondrá en 130). 5 usuario y 130 peticiones, por lo tanto se tomarán 650 muestras.

Figura 7.2: Configurando grupo de hilos en JMeter

Ahora se configurará la petición HTTP que tiene por defecto JMeter. Para ello hay que seleccionar el grupo de hilos y con el botón derecho añadir “Valores por defecto para petición HTTP”, desde la pestaña de elementos de configuración.

Figura 7.3: Configurando petición HTTP que tiene por defecto JMeter

También debemos configurar las peticiones que se realizarán al servidor, añadiremos el login. Para ello hay que añadir desde la pestaña “Muestrador” una petición HTTP especificando el usuario y contraseña.

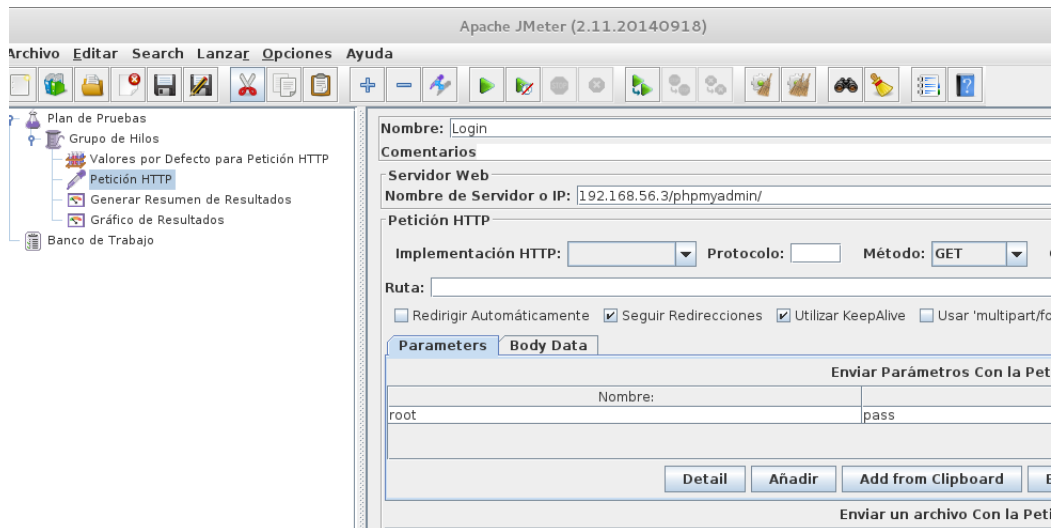


Figura 7.4: Configurando petición HTTP a JMeter

Si nuestro servidor utiliza cookies, es conveniente activar las cookies. Para ello seleccionamos añadir un gestor de cookies HTTP.

Para ver los resultados añadiremos un gráfico de resultados y un reporte del resumen. Con el primero podremos ver una gráfica del rendimiento, la desviación, la mediana, y los datos obtenidos, el segundo solo lo usaremos para detectar si hay errores, ya que nos permite ver el porcentaje de errores, además del mínimo, máximo, y valores del gráfico.

Lanzamos el test y este es el resultado:

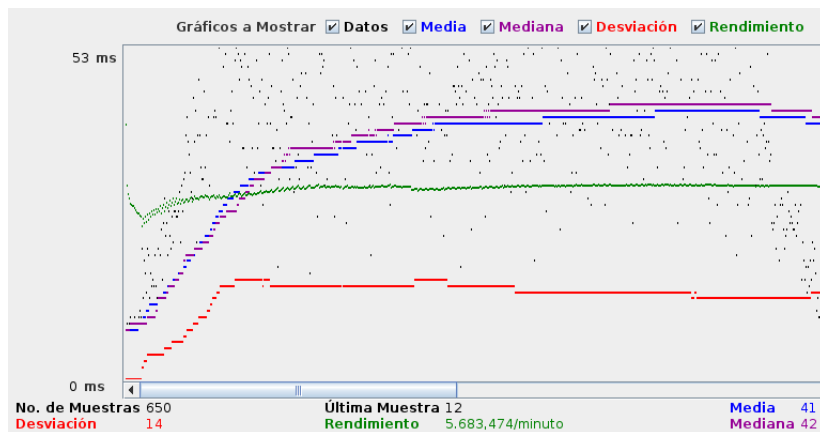


Figura 7.5: Gráfico generado por JMeter de una prueba realizada al login de phpmyadmin en Ubuntu Server

Etiqueta	# Muestras	Media	Mín	Máx	Desv. Están...	% Error	Rendimiento	Kb/sec	Media de Byt...
Login	650	41	8	100	14,27	0,00%	94,7/sec	882,83	9543,7
Total	650	41	8	100	14,27	0,00%	94,7/sec	882,83	9543,7

Figura 7.6: Reporte del resumen generado por JMeter de una prueba realizada al login de phpmyadmin en Ubuntu Server

Como puede apreciarse, la aplicación a resistido las 650 peticiones sin problema. El rendimiento en las primeras peticiones era ligeramente mayor, pero más o menos se ha mantenido constante. Los cálculos aproximan que el sistema podría procesar 5.683,474/minuto, si mantiene el mismo rendimiento para ese número de peticiones.

Vamos a intentar llevar al sistema al límite, para ello repetiremos el test para un número mayor de usuarios. Realizaremos solamente 1 petición, pero la realizarán 2000 usuarios a la vez.

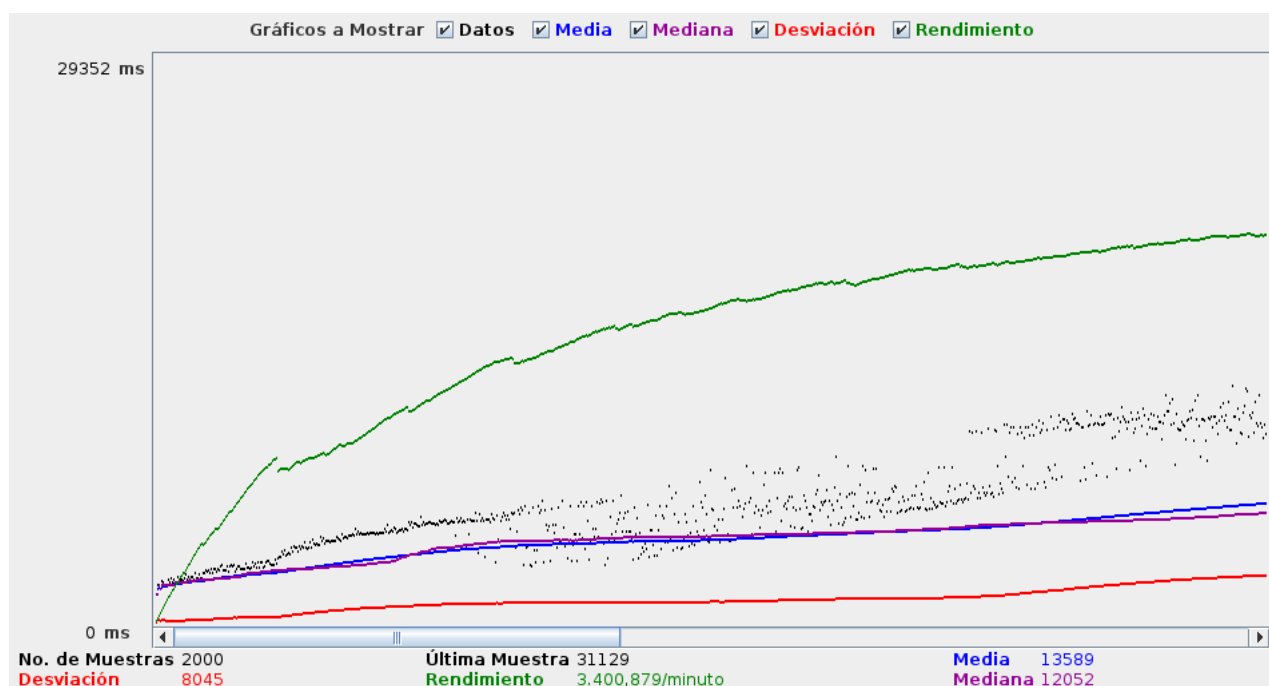


Figura 7.7: Gráfico generado por JMeter de una prueba realizada al login de phpmyadmin en Ubuntu Server

Etiqueta	# Muestras	Media	Mín	Máx	Desv. Están...	% Error	Rendimiento	Kb/sec	Media de Byt...
Login	2000	13589	1581	33591	8045,88	3,15%	56,7/sec	515,87	9319,7
Total	2000	13589	1581	33591	8045,88	3,15%	56,7/sec	515,87	9319,7

Figura 7.8: Reporte del resumen generado por JMeter de una prueba realizada al login de phpmyadmin en Ubuntu Server

Esta vez el rendimiento empeora notablemente a lo largo del tiempo. Además, el sistema no ha sido capaz de responder con éxito las últimas peticiones, se han respondido con éxito el 96.85 %, por lo que hemos descubierto que nuestro sistema actual no sería capaz de resistir el login de 2000 usuarios a la vez, esto podría ser un problema dependiendo del número de usuarios que se estimen. Además, las peticiones que ha resuelto con éxito no las ha resuelto con demasiada agilidad, habría que tratar de realizar pruebas con el número de usuarios aproximado que se espera en el sistema, de ese modo podríamos determinar si el rendimiento no es el adecuado.

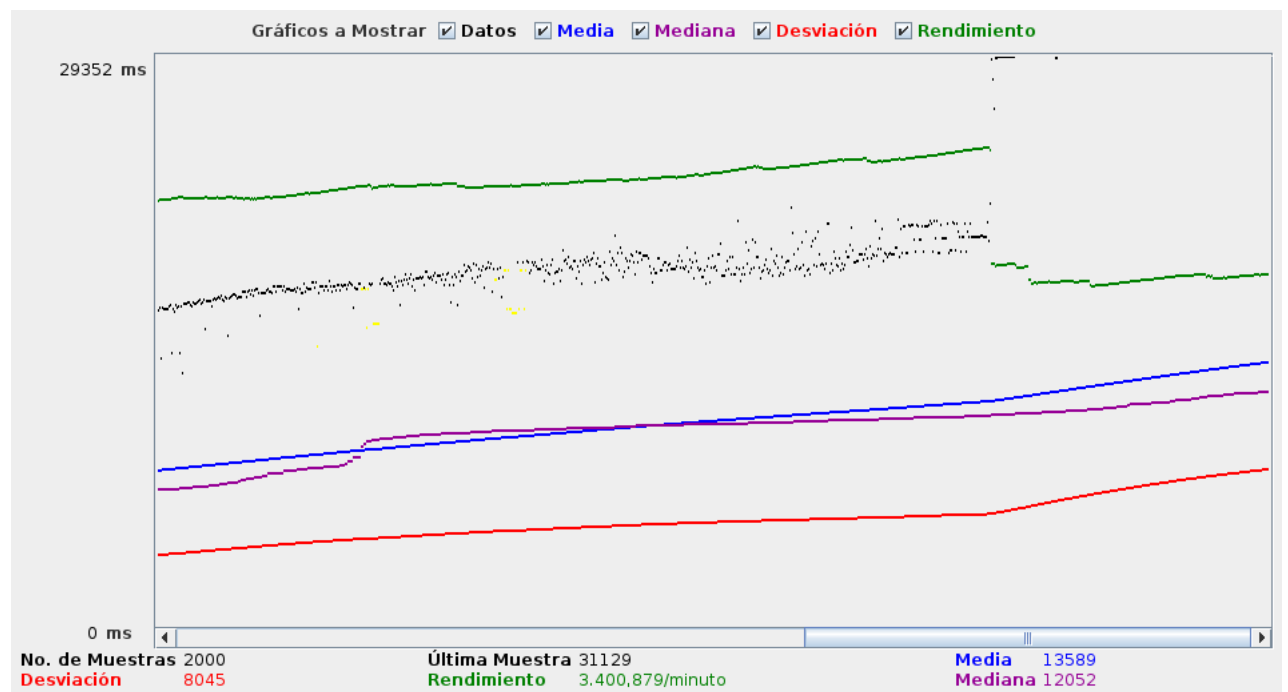


Figura 7.9: Errores mostrados en el gráfico generado por JMeter de una prueba realizada al login de phpmyadmin en Ubuntu Server

8. Seleccione un benchmark entre SisoftSandra y Aida. Ejecútelo y muestre capturas de pantalla comentando los resultados.

Se ha seleccionado Aida, procedemos a descargar la versión de prueba de 30 días desde su página oficial, la cual se indica en la práctica (<http://www.aida64.com/downloads>).

La instalación es sencilla, únicamente hay que indicar el idioma, aceptar los términos y condiciones de uso que establece la empresa, seleccionar el directorio donde se instalará y si se desea crear accesos directos.

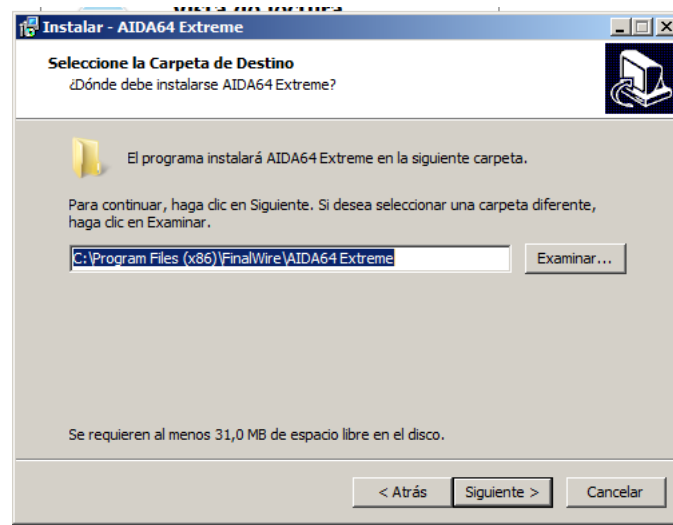


Figura 8.1: Seleccionando directorio en el que se instalará Aida64 en Windows Server 2008 R2



Figura 8.2: Instalación finalizada de Aida64 en Windows Server 2008 R2

Una vez instalado lo ejecutamos, y esto es lo que podemos ver. A la izquierda tenemos una serie de opciones para inspeccionar el hardware de la máquina y el rendimiento, y en la parte superior un menú de opciones:

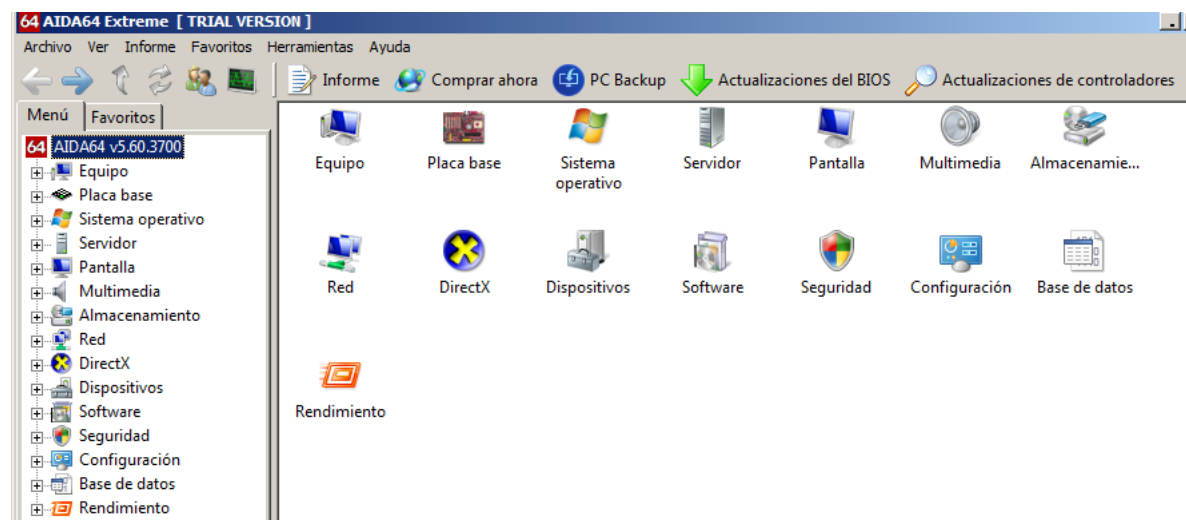


Figura 8.3: Instalación finalizada de Aida64 en Windows Server 2008 R2

Para utilizar un benchmark seleccionaremos alguna de las pruebas que aparecen en “Herramientas”, por ejemplo de disco. [11].

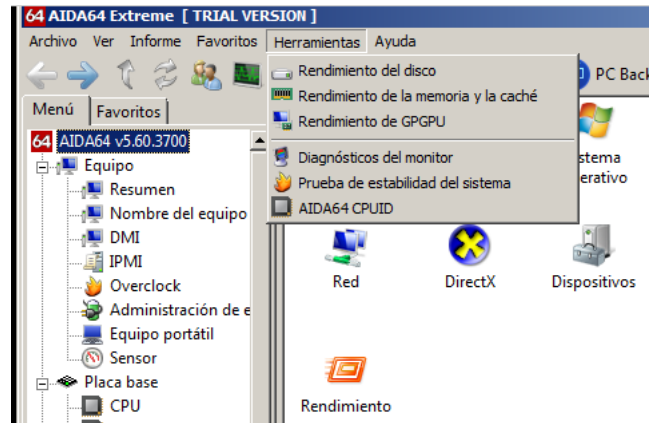


Figura 8.4: Tipos de benchmarks que ofrece Aida64

En la parte de abajo se muestran los benchmarks disponibles para realizar pruebas relacionadas con el disco:

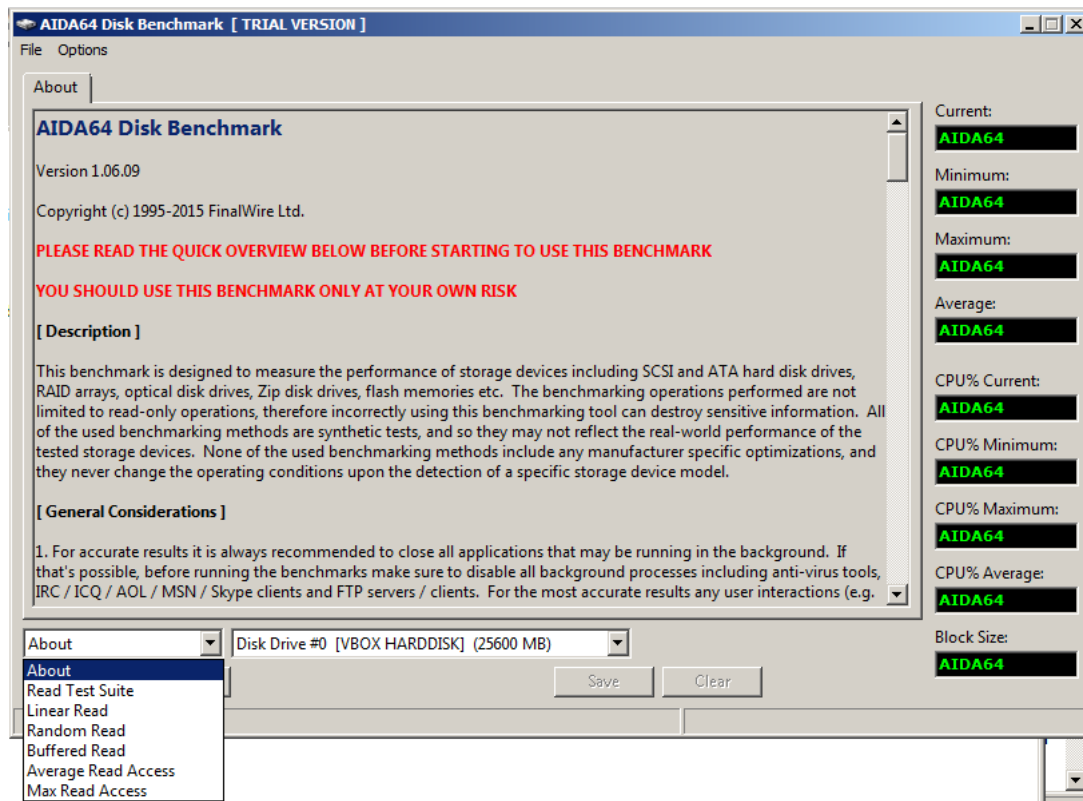


Figura 8.5: Benchmarks de disco que ofrece Aida64

En este caso vamos a seleccionar “Linear Read” para el disco 0, de este modo analizaremos

el rendimiento de las lecturas para este disco.

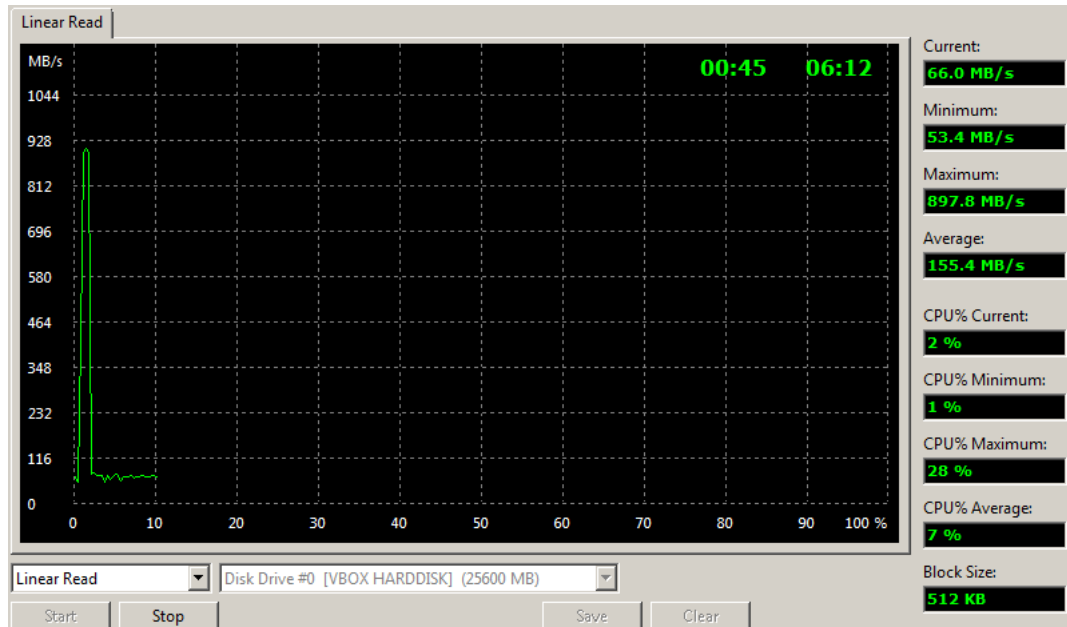


Figura 8.6: Realizando benchmark de lecturas en disco utilizando Aida64

Y este es el resultado generado:

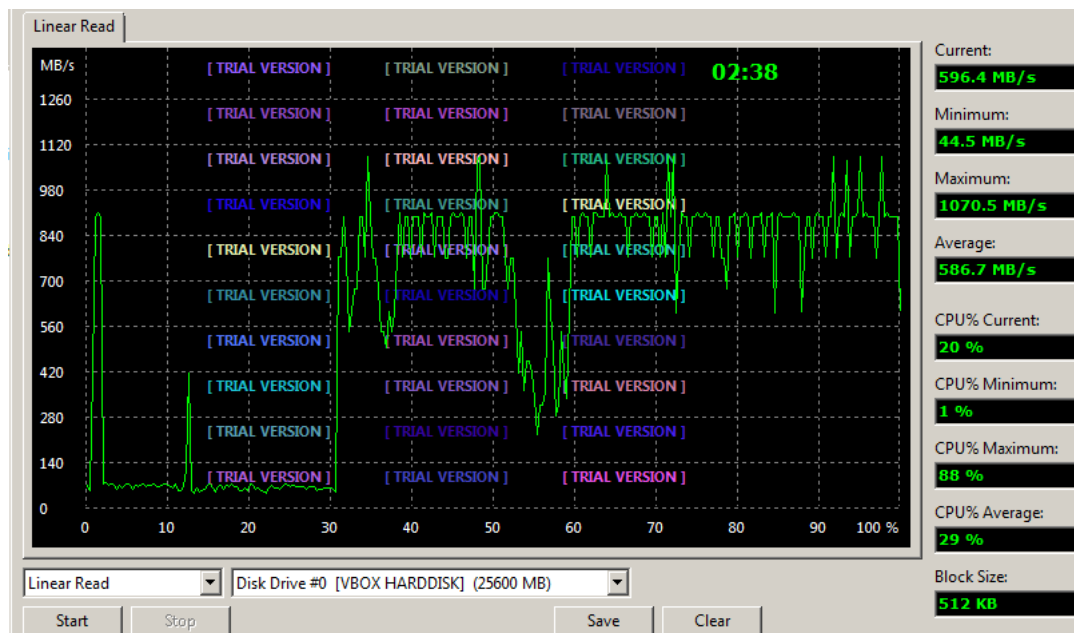


Figura 8.7: Resultado del benchmark de lecturas en disco realizao con Aida64

Como podemos apreciar en la imagen, el resultado varía entre un amplio rango, su mayor pico es de casi 1120 MB/s, el cual lo alcanza en varias ocasiones. Hay momentos en los que este rendimiento desciende notablemente, en un momento desciende por debajo de los 280 MB/s, justo después de alcanzar algunos de los mayores picos.[10, 9]

9. 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

9.1. Objetivo del benchmark

Se ha decidido realizar un benchmark para medir el rendimiento de la batería de un sistema Android. Este benchmark realizará un profiling de algunas de las características de la batería, para posteriormente sobrecargar el terminal, y a partir de la diferencia con los niveles iniciales asignarle una puntuación reflejando la resistencia de la batería al test.

9.2. Métricas

Para realizar las medidas se ha encontrado una dificultad, y es la afirmación de Google de que no es necesario controlar de forma exacta el nivel de batería [25], por lo que la API no nos ofrece esta facilidad. Lo que podemos es ver el nivel de la batería sin precisión decimal, por lo que para poder analizar el rendimiento de la batería es necesario llegar a drenar puntos de este nivel. No se pretende medir el rendimiento de la batería en sí misma, sino la resistencia que ofrece soportando un alto uso del procesador y de los sensores que disponga el terminal en cuestión.

La métrica se realiza guardando el nivel de la batería previamente a activar el sensor de bluetooth y Wi-Fi, y realizar bucles realizado vibraciones, una gran serie de conexiones HTTP, búsquedas de dispositivos a través de bluetooth, cálculos, y cambios en el brillo. Todo esto se realiza impidiendo al dispositivo atenuar la pantalla. Tras esto se vuelve a captar el nivel de batería y se asigna una puntuación.

La puntuación que se asigna castiga especialmente los primeros niveles de batería perdidos, ya que pueden llegar a ser los más significativos, e igualando un poco más la puntuación para pérdidas superiores. Se busca dar una puntuación positiva, acotada superiormente por 100. Para esto los cálculos que se realizan son los siguientes:

1º - Obtener la diferencia de antes y después del test.

2º - Desde 1 hasta la diferencia, de 1 en 1:

3º - Puntuación acumulada (inicialmente la máxima, 100) menos el índice de la iteración.

4º - Puntuación obtenida en el paso anterior dividida entre 10.

- 5º - La puntuación anterior se resta a la puntuación acumulada.
- 6º - Vuelta al paso 2º.

Por ejemplo:

```
Punt. inicial: 67
Punt. final: 65

Diferencia —> 67 - 65 = 2

100 - 1 = 99 —> 99 / 10 = 9.9 —> 100 - 9.9 = 90.1
90.1 - 2 = 88.1 —> 88.1 / 10 = 8.81 —> 90.1 - 8.81 = 81.29

Punt.: 81.29
```

9.3. Instrucciones para su uso

El primer paso sería instalar la aplicación en un terminal Android, para ello solo hay que copiar la aplicación a la raíz del dispositivo y desde este acceder a ella y seguir los pasos que nos indica nuestro sistema. Posteriormente hay que ejecutarla y pulsar el botón de inicio.



Figura 9.1: Pantalla inicial del benchmark mostrando advertencia

Si tratamos de utilizar el benchmark durante la carga del dispositivo, nos mostrará un advertencia para que lo desconectemos, ya que de lo contrario no funcionaría correctamente.



Figura 9.2: Pantalla inicial del benchmark

Pulsamos el botón de inicio y vemos como este cambiará de color, mientras que el color esté activo no se debe manipular el dispositivo. Durante el uso del benchmark se activarán varios sensores, estos volverán a su estado inicial al finalizar el proceso. El proceso puede tardar varios minutos. (Nota: estos minutos son necesarios, ya que se requiere intentar drenar algunos niveles enteros de la batería).



Figura 9.3: Realizando el benchmark al dispositivo

Una vez finalizado, se mostrará una pantalla con los resultados obtenidos, asignándose así una puntuación:



Figura 9.4: Resultados obtenidos por el benchmark

9.4. Ejemplo de uso analizando los resultados

Partiendo del ejemplo anterior, podemos ver como en la primera pantalla se realiza un profiling mostrando algunos datos de la batería: carga actual, estado de la batería, voltaje, y temperatura. (Figura 9.2)

En la figura 9.3, se aprecia la puntuación asignada en función de la métrica que ya hemos mencionado, junto con información acerca del tiempo que se ha requerido para realizar el test, y la diferencia en la carga del dispositivo.

En la misma figura, para realizar todas las operaciones que se incluyen el sistema ha necesitado 257 segundos, y en ese tiempo se han consumido hasta 3 puntos de la batería. A partir de esto la métrica le ha asignado una puntuación que puede servirnos para comparar con otros dispositivos, o con el mismo dispositivo en distintas situaciones.

9.5. Implementación

No se incluye en la memoria porque supone unas 15 páginas solo de código, por ello se ha optado por incluir el proyecto junto a la práctica. Solo se va a incluir a continuación el código de la función que realiza el benchmark.

```
void bechmarkStart(){
    boolean wifi_conectado;
    boolean blu_inicio;

    Calendar c = Calendar.getInstance();
    int seg = c.get(Calendar.SECOND);
```

```

int min = c.get(Calendar.MINUTE);

Vibrator v = (Vibrator)
this.getSystemService(Context.VIBRATOR_SERVICE);

BluetoothAdapter bluetooth =
BluetoothAdapter.getDefaultAdapter();

// START *****

double nivel_inicio = getNivelActualR();
double time_ini = System.nanoTime();

// EVITAR ATENUAR PANTALLA
PowerManager pm = (PowerManager)
getSystemService(Context.POWER_SERVICE);

PowerManager.WakeLock wl =
pm.newWakeLock(PowerManager.SCREEN_DIM_WAKE_LOCK,
"START");

wl.acquire();

// SENSORES ON

// wifi
WifiManager wifi = (WifiManager)
this.getSystemService(Context.WIFI_SERVICE);

wifi_conectado = wifi.isWifiEnabled();
if (!wifi_conectado)
    wifi.setWifiEnabled(true);

// Bluetooth
blu_inicio = bluetooth.isEnabled();
if (!blu_inicio) {
    bluetooth.enable();
}

// COMPUTOS
for (int i = seg; i < 10 + seg; i++){

```

```

for (int z = 0; z < 30; z++)
    v.vibrate(1000);

for(int j = min; j < 5000 + min; j++){

    httpRequest("192.168.215.78");
    bluetooth.startDiscovery();
    String locationProvider =
    LocationManager.NETWORK_PROVIDER;

    Thread hebra = new Thread(new Runnable() {
        @Override
        public void run() {

            double sum = 0;
            Calendar c = Calendar.getInstance();
            int seg = c.get(Calendar.SECOND);
            int min = c.get(Calendar.MINUTE);

            // Localizacion GPS
            String locationProvider =
            LocationManager.NETWORK_PROVIDER;

            // Computos
            for(int x = seg; x < 10000 + seg; x++) {
                seg = c.get(Calendar.SECOND);
                sum *= (seg / Math.pow(min,2));
            }
        }
    });
    hebra.start();

}

// Brillo

Settings.System.putInt(getApplicationContext().
    getContentResolver(), Settings.System.
    SCREEN_BRIGHTNESS_MODE, 0);

if (i % 2 == 0) {

```

```

        Settings.System.putInt (getApplicationContext().
            getContentResolver(), Settings.System.
            SCREEN_BRIGHTNESS, 255);

        WindowManager.LayoutParams wm = getWindow().
            getAttributes();

        wm.screenBrightness = 255;
        getWindow().setAttributes(wm);
    } else {
        Settings.System.putInt (getApplicationContext().
            getContentResolver(),
            Settings.System.SCREEN_BRIGHTNESS, 0);

        WindowManager.LayoutParams wm =
            getWindow().getAttributes();
        wm.screenBrightness = 0;
        getWindow().setAttributes(wm);
    }

}

wl.release();

// RESULTADOS *****
double nivel_final = getNivelActualR();
double time_fin = System.nanoTime();
setResults(nivel_inicio, nivel_final, time_ini,
    time_fin);

// Restaurar *****
if (!wifi_conectado)
    wifi.setWifiEnabled(false);

if (!blu_inicio)
    bluetooth.disable();

}

```

[25, 15, 16, 19, 24, 17, 14, 22, 21, 23, 26, 18, 20]

Referencias

- [1] <https://wiki.ubuntu.com/PhoronixTestSuite>, consultado el 09 de Diciembre de 2015.
- [2] <https://openbenchmarking.org/test/pts/openssl>, consultado el 10 de Diciembre de 2015.
- [3] <http://gatling.io/#/>, consultado el 11 de Diciembre de 2015.
- [4] <https://httpd.apache.org/docs/2.2/programs/ab.html>, consultado el 11 de Diciembre de 2015.
- [5] <http://www.scala-lang.org/>, consultado el 11 de Diciembre de 2015.
- [6] <http://www.scala-lang.org/documentation/getting-started.html>, consultado el 11 de Diciembre de 2015.
- [7] <http://www.scala-lang.org/what-is-scala.html>, consultado el 11 de Diciembre de 2015.
- [8] <http://gatling.io/docs/2.1.7/quickstart.html>, consultado el 12 de Diciembre de 2015.
- [9] http://archive.benchmarkreviews.com/index.php?option=com_content&task=view&id=635&Itemid=60&limitstart=9, consultado el 14 de Diciembre de 2015.
- [10] <http://koltsoff.com/pub/blockspeed/>, consultado el 14 de Diciembre de 2015.
- [11] <http://www.aida64.com/products/features/benchmarking>, consultado el 14 de Diciembre de 2015.
- [12] <http://linux.die.net/man/1/wc>, consultado el 16 de Diciembre de 2015.
- [13] <http://ss64.com/bash/ps.html>, consultado el 16 de Diciembre de 2015.
- [14] <http://developer.android.com/intl/es/guide/topics/connectivity/bluetooth.html>, consultado el 20 de Diciembre de 2015.
- [15] <http://developer.android.com/intl/es/guide/topics/manifest/manifest-intro.html>, consultado el 20 de Diciembre de 2015.
- [16] <http://developer.android.com/intl/es/guide/topics/manifest/manifest-intro.html>, consultado el 20 de Diciembre de 2015.
- [17] <http://developer.android.com/intl/es/guide/topics/ui/declaring-layout.html>, consultado el 20 de Diciembre de 2015.

- [18] <http://developer.android.com/intl/es/reference/android/content/Context.html>, consultado el 20 de Diciembre de 2015.
- [19] <http://developer.android.com/intl/es/reference/android/net/wifi/package-summary.html>, consultado el 20 de Diciembre de 2015.
- [20] <http://developer.android.com/intl/es/reference/android/os/PowerManager.html>, consultado el 20 de Diciembre de 2015.
- [21] <http://developer.android.com/intl/es/reference/android/os/Vibrator.html>, consultado el 20 de Diciembre de 2015.
- [22] <http://developer.android.com/intl/es/reference/android/provider/Settings.System.html>, consultado el 20 de Diciembre de 2015.
- [23] <http://developer.android.com/intl/es/training/basics/network-ops/connecting.html>, consultado el 20 de Diciembre de 2015.
- [24] <http://developer.android.com/intl/es/training/basics/network-ops/index.html>, consultado el 20 de Diciembre de 2015.
- [25] <http://developer.android.com/intl/es/training/monitoring-device-state/battery-monitoring.html>, consultado el 20 de Diciembre de 2015.
- [26] <http://developer.android.com/intl/es/training/volley/simple.html>, consultado el 20 de Diciembre de 2015.