Ingeniería de Servidores (2014-2015)
Grado en Ingeniería Informática
Universidad de Granada

Memoria Práctica 4

Jose Arcos Aneas

16 de diciembre de 2014

Índice

Índice de figuras

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

En ubuntu, podemos instalarlo simplemente usando (como root) el comando: *apt-get install phoronix-test-suite.*
Una vez instalado, tenemos que ejecutarlo, para que nos pregunte sí queremos aceptar la licencia y sí queremos permitir enviar a los desarrolladores estadísticas anónimas para mejorar el programa. Para ello escribimos: *phoronix-test-suite* Para listar los benchmarks disponibles, podemos utilizar los siguiente comando: *phoronix-test-suite list-available-tests* - Para listar los benchmarks *phoronix-test-suite list-available-suites* - Para listar las suites Para listar los benchmark que tenemos hemos de utilizer el commando: **phoronix-test-suite list-tests**

```
blunt@blunt:~$ phoronix-test-suite list-tests

Phoronix Test Suite v4.8.3
Available Tests

pts/aio-stress                  - AIO-Stress                          Disk
pts/apache                      - Apache Benchmark                    System
pts/apitest                     - APITest                             Graphics
pts/apitrace                    - APITrace                            Graphics
pts/battery-power-usage         - Battery Power Usage                 System
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-webkitfltk            - Timed WebKitFLTK Compilation        Processor
pts/bullet                      - Bullet Physics Engine               Processor
pts/byte                        - BYTE Unix Benchmark                 Processor
pts/c-ray                       - C-Ray                               Processor
pts/cachebench                  - CacheBench                          Processor
pts/cairo-demos                 - Cairo Performance Demos             Graphics
pts/cairo-perf-trace            - cairo-perf-trace                    Graphics
pts/clomp                       - CLOMP                               Processor
pts/compilebench                - Compile Bench                       Disk
pts/compress-7zip               - 7-Zip Compression                   Processor
pts/compress-gzip               - Gzip Compression                    Processor
pts/compress-lzma               - LZMA Compression                    Processor
pts/compress-pbzip2             - Parallel BZIP2 Compression          Processor
pts/corebreach                  - CoreBreach                          Graphics
pts/crafty                      - Crafty                              Processor
pts/csgo                        - Counter-Strike: Global Offensive    Graphics
```

Figura 1.1: Listado de bechmarks disponibles con phoronix-test-suite list-tests.

[1]

2. De los parámetros que le podemos pasar al comando ¿Qué significa -c 30 ? ¿y -n 1000?

La opción -c concurrencia indica el número máximo de peticiones que se podrán ejecutar simultáneamente. Con -c 30 lo ponemos a 30. La opción -n peticiones establece el número

[1] http://www.phoronix-test-suite.com/

de peticiones de página web que se harán al servidor. Con -n 1000 establecemos que se hagan mil peticiones.
[2]

3. Ejecute ab contra a las tres máquinas virtuales (desde el SO anfitrión a las máquina virtuales de la red local) una a una (arrancadas por separado) y muestre las estadísticas. ¿Cuál es la que proporciona mejores resultados? Fíjese en el número de bytes transferidos, ¿es igual para cada máquina?

En primer lugar *ab* se instala con el comando **apt-get install apache2-utils**. Una vez instalado podemos generar un benchmarking de la pagina de que deseemos. En lugar de copiar y pegar el resultado de cada una de las salidas de este comando en cada uno de los servidores se han generado tres graficas en relacion al numero de peticiones y el tiempo que se ha consumido para su realizacion. El ejercicio pide que nos que comprobemos si se han transferido el numero cuya respuesta es que no. Cada servidor transfiere un número distinto de bytes.

Windows

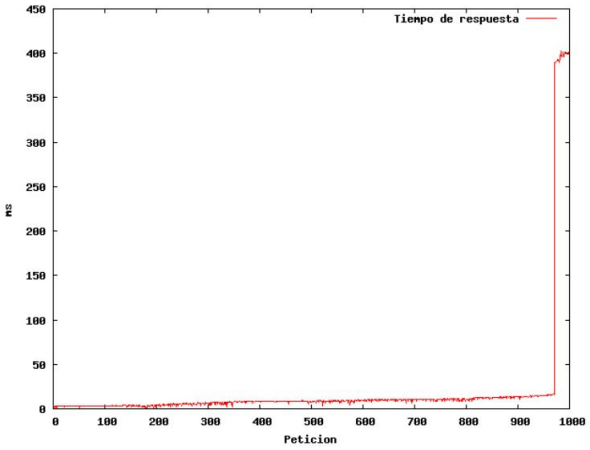

Figura 3.1: Grafica de Windows.

CentOs

[2]http://httpd.apache.org/docs/2.2/programs/ab.html

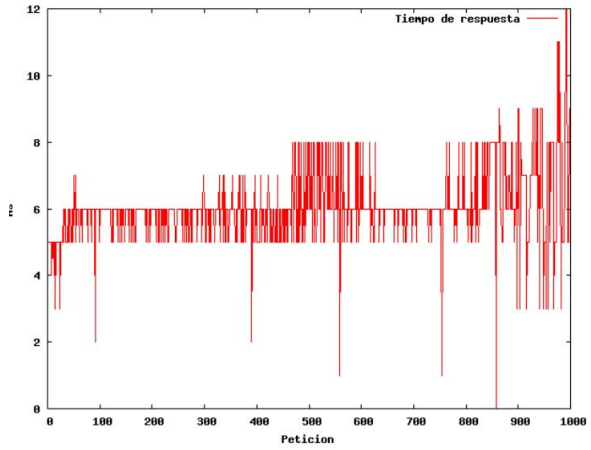

Figura 3.2: Grafica de CentOS.

Ubuntu

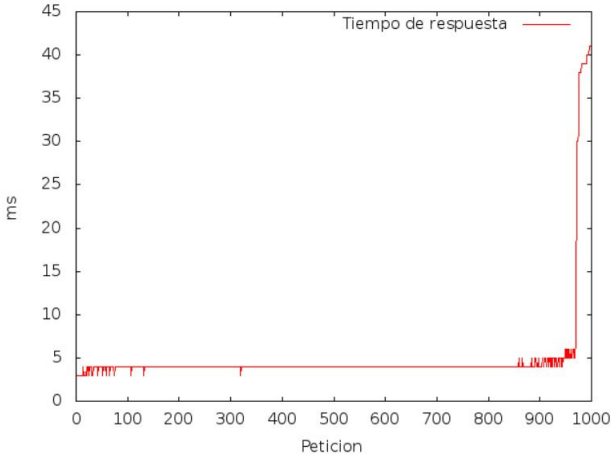

Figura 3.3: Grafica de Ubuntu.

Se ha realizado 3 gráficas una por cada servidor, con su tiempo de respuesta. Con el comando *"ab –c 30 –n 1000 http://<servidor>"*
Puede observarse que se han puesto las gráficas en orden, de mayor tiempo a menor tiempo. Se puede ver como el que peor tiempo de respuesta es el de Windows server. (Aunque no se distinga exactamente los valores que toma) Al final ser sobrecarga demasiado y tarda hasta 40-50 veces más de lo normal. Y después le sigue, con un tiempo un poco más constante la máquina CentOs que aunque también se le envíen 1000 peticiones no llega a sobrecargarse tanto como la de Windows. Pero de media tiene unos tiempo un

poco peores que los de Ubuntu. Y finalmente Ubuntu Server tiene unos tiempo constantes inferiores a 5 ms, pero llega un momento en el que se sobrecarga y comienza a tardar hasta 20 veces más lento.

4. Instale y siga el tutorial en http://jmeter.apache.org/usermanual/build-web-test-plan.html realizando capturas de pantalla y comentándolas. En vez de usar la web de jmeter, haga el experimento usando alguna de sus máquinas virtuales (Puede hacer una página sencilla, usar las páginas de phpmyadmin, instalar un CMS, ATC.).

Después de instalarlo, con los comando de siempre (apt-get install jmeter), lo ejecutamos, hacemos click derecho en "Plan de Pruebas" y seleccionamos Añadir - Grupo de Hilos. Lo modificamos y lo dejamos con estos parametros:

Figura 4.1: Seleccion del grupo de hilos.

Ahora creamos las opciones por defecto para las consultas HTTP. Para ello, click derecho, Añadir - Elementos de Configuración - Valores por defecto para Petición HTTP. Lo dejamos con estas opciones:

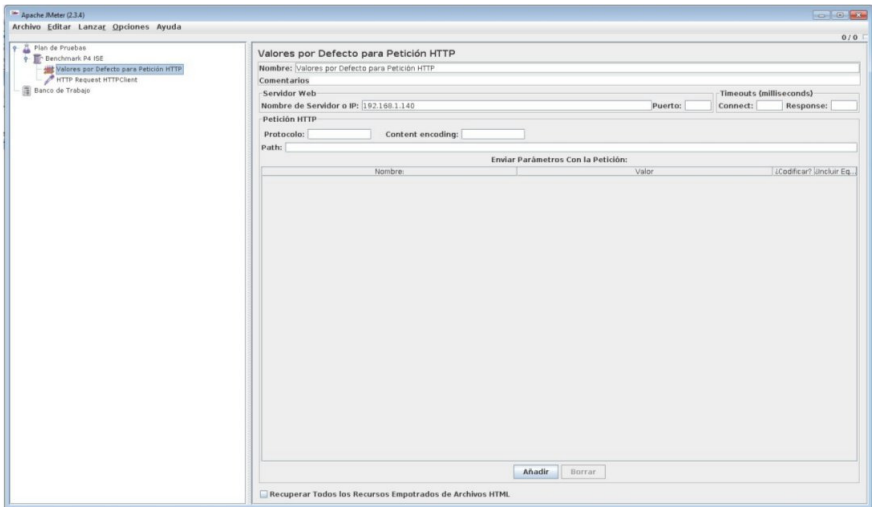

Figura 4.2: Creacion de opciones para consulta HTTP.

Después creamos dos consultas HTTP. Una nos llevará al directorio raíz, otra a una página con imágenes. Para añadir las consultas sería pulsar Añadir – Muestreador – HTTP Request HTTP client. Añadimos dos y dejamos estos parámetros:

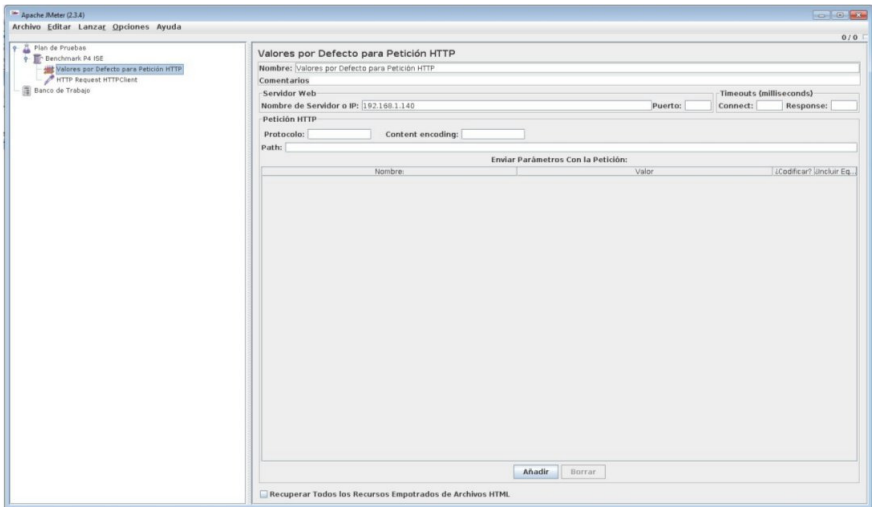

Figura 4.3: Creacion de consultas HTTP.

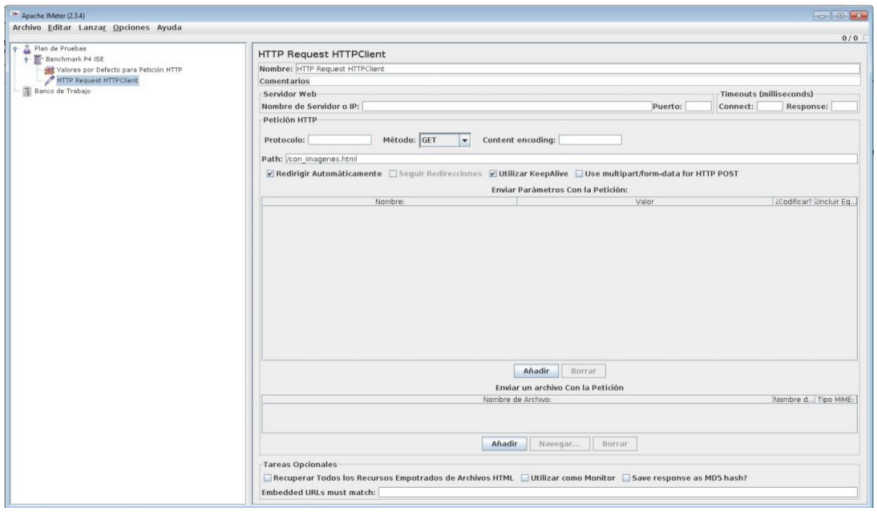

Figura 4.4: creacion Request HTTPClient.

Por último, solo nos queda añadir el gráfico. Añadir – Listener – Gráfico de Resultados. Una vez creado, lo guardamos en donde queramos.

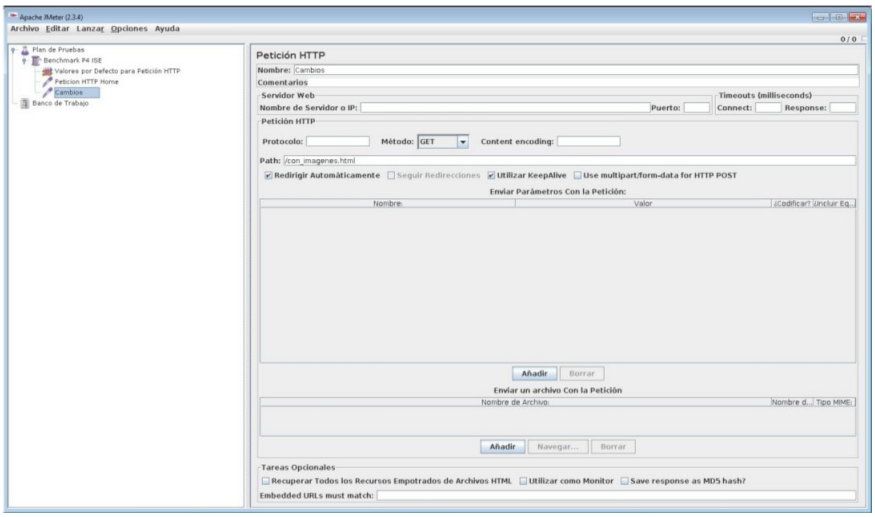

Figura 4.5: Añadir el grafico de resultados.

Por último, ejecutamos (Lanzar – Arrancar) y se generará un gráfico con los resultados:

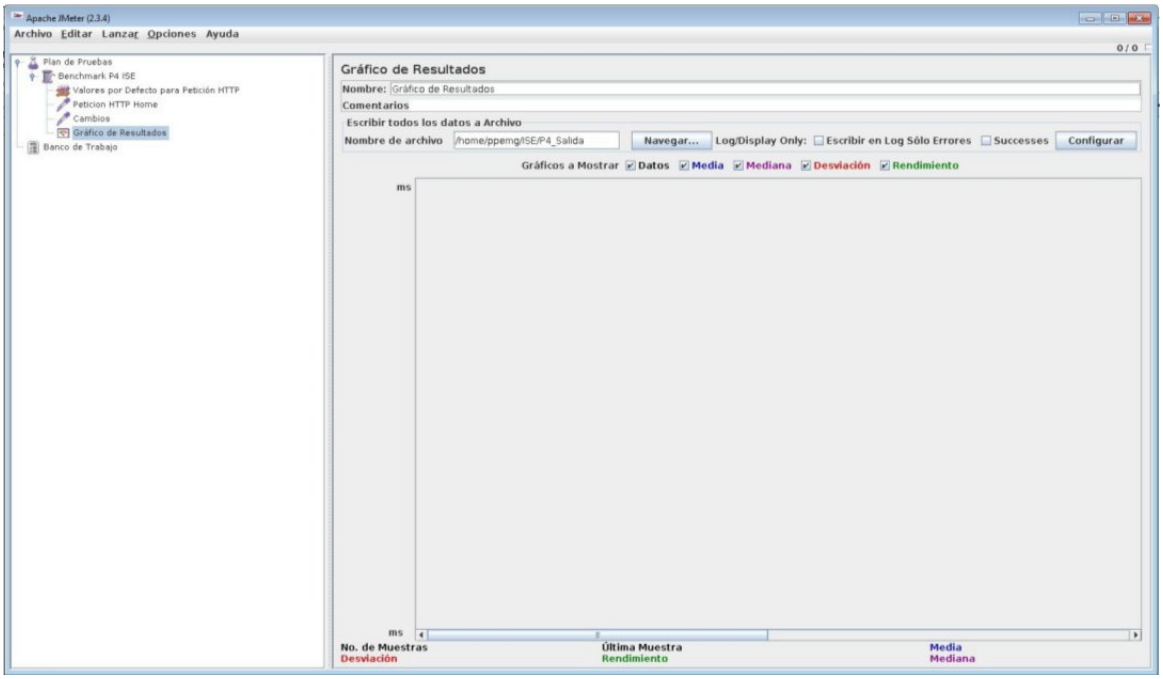

Figura 4.6: Generación del gráfico.

5. Programe un benchmark usando el lenguaje que desee. Elbenchmark debe incluir: 1) Objetivo del benchmark 2) Métricas (unidades, variables, puntuaciones, etc.) 3) Instrucciones para su uso 4) Ejemplo de uso analizando los resultados

He hecho un benchmark para un test de CPU y de memoria. He hecho un bucle que haga operaciones de suma, resta,division y multiplicacion. Respecto a la CPU he incluido unos calculos para punto flotante. Ademas, he intentado considerar el caso para multicpu con openmp. Para el test de memoria tenia pensado realizar una ordenación por el algoritmo de la burbuja (por ejemplo) pero en lugar de eso he creado un vector de memoria dinamica e 2.500.000 enteros que son alrededor de 8.000.000 de bytes. Lo suficiente para que no quepa en la cahe. Accedo aleatoriamente a las posiciones del vector.

```
1 /*
2  * Benchmark para la asignatura de ISE.
3  * Test de CPU y de Memoria.
4  *
5  */
6 #include <iostream>
7 #include <fstream>
```

```
8  #include <ctime>
9
10 using namespace std;
11
12 int main(){
13
14     clock_t tantes,tdespues;
15     double tiempo;
16     int puntuacion_cpu, puntuacion_mem;
17
18     cout<<'\n';
19     for(int i=0;i<30;i++) cout<<"*";
20     cout<<"\n** BENCHMARK 0.1  **\n";
21     for(int i=0;i<30;i++) cout<<"*";
22     cout<<endl;
23
24     //TEST DE CPU
25
26     cout<<"\n### TEST DE CPU ### \n";
27
28     int cantidad=0;
29     double cantidad2=1.5;
30
31     tantes = clock();
32
33     cout<<"\nPasando test de CPU..."<<endl;
34     #pragma omp parallel for
35     for(int i=0;i<50000000;i++){
36         cantidad = cantidad*2;
37         cantidad = cantidad/2;
38         cantidad++;
39         cantidad--;
40     }
41     for(int i=0;i<5000000;i++)
42         cantidad2=cantidad2*1.5;
43     for(int i=0;i<5000000;i++)
44         cantidad2=cantidad2/1.5;
45     #pragma omp end parallel for
46
47     tdespues = clock();
48
49     tiempo = static_cast<double>(tdespues - tantes)/CLOCKS_PER_SEC;
50     cout<<"\nTiempo empleado: "<<tiempo<<" segundos"<<endl;
51
52     puntuacion_cpu = static_cast<int>(10000/tiempo);
53     cout<<"\nPuntuacion obtenida: "<<puntuacion_cpu<<" puntos\n"<<endl;
54
55    //TEST DE MEMORIA
56
57    cout<<"\n\n### TEST DE MEMORIA ### \n";
58
59    int *vector;
60    int aux;
61    vector = new int[2500000];
```

```
62
63    tantes = clock();
64
65    cout<<"\nPasando test de Memoria..."<<endl;
66     #pragma omp parallel for
67     for(int i=0;i<200;i++){
68         //Escritura en vector
69         for(int j=0;j<500000;j++)
70             for(int k=0;k<2500000;k+=500000)
71                 vector[k+j]=j;
72         //Lectura del vector
73         for(int j=0;j<500000;j++)
74             for(int k=0;k<2500000;k+=500000)
75               aux = vector[j];
76     }
77     #pragma omp end parallel for
78
79    tdespues = clock();
80
81      tiempo = static_cast<double>(tdespues - tantes)/CLOCKS_PER_SEC;
82    cout<<"\nTiempo empleado: "<<tiempo<<" segundos"<<endl;
83
84      puntuacion_mem = 10000/tiempo;
85    cout<<"\nPuntuacion obtenida: "<<puntuacion_mem<<" puntos\n"<<endl;
86 }
```

Para obtener una valoración he dividido 10000 entre el tiempo que tarda en ejecutarse, lo que me genera una puntuación. Con lo que a menor tiempo mayor puntuación. Los resultados obtenidos son los siguiente:

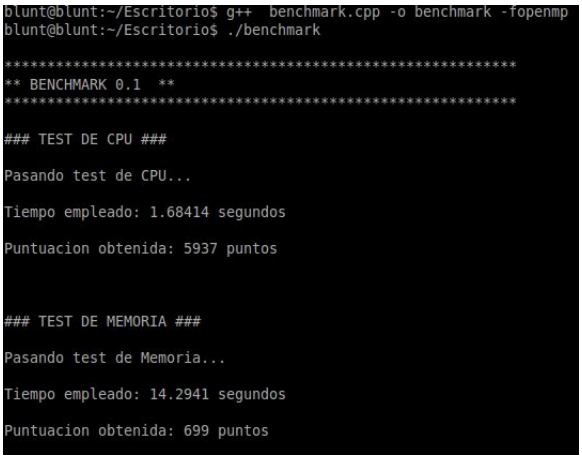

Figura 5.1: Resultado de mi benchmark.

En la imagen se muestra que el tiempo de ejecución del test de CPU es menor y por tanto se obtiene una mayor puntuación.

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

Para mas infomación sobre estos benchmark aconsejo la siguiente dirección [3] .
Una podría ser el de "blogbench": - phoronix-test-suite install blogbench En [4] viene una descripción del funcionamiento del benchmark blogbench, cuya traducción literal es la siguiente.
Blogbench es un punto de referencia del sistema de ficheros portátil que intenta reproducir la carga de un servidor de archivos ocupado del mundo real. Hace hincapié en el sistema de archivos con múltiples hilos de realizar lecturas al azar, escribe y reescribe con el fin de tener una idea realista de la escalabilidad y la concurrencia de un sistema puede manejar.
En mi sistema una ejecución de este programa es la siguiente :

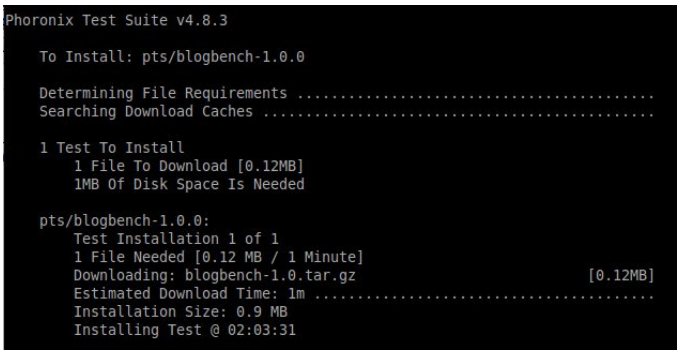

Figura 5.2: Ejecucion de blogbench.

Y el último que vamos a probar es otro para procesadores llamado "bork", que al igual que el anterior [5] en su página tiene una definición de la labor que desempeña este benchmark : Bork es un pequeño, multiplataforma herramienta de cifrado de archivos. Está escrito en Java y diseñado para ser incluido junto con los archivos que cifra para el almacenamiento a largo plazo. Esta prueba mide la cantidad de tiempo que se necesita para cifrar un archivo de ejemplo. Para ejecutarlo introducir el siguiente comando. - phoronix-test-suite benchmark bork

[3] http://openbenchmarking.org/tests/pts
[4] http://openbenchmarking.org/test/pts/blogbench
[5] http://openbenchmarking.org/test/pts/bork

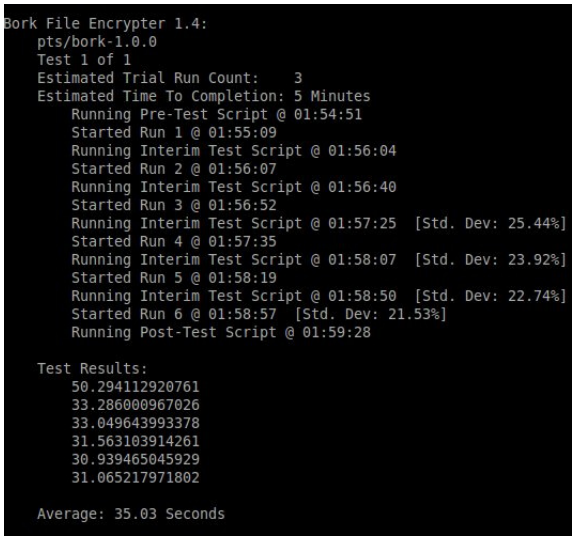

Figura 5.3: Ejecución del bork.

Cuestión opcional 2: ¿Qué es Scala? Instale Gatling y pruebe los escenarios por defecto.

En primer lugar "Scala" es un lenguaje de programacion de proposito general diseñado par aexpresar patrones comunes de programacion de foroma concisa y elegante. Se integran las caracterisiticas d ellos elnguajes orientados a obejetos y funcional. Scala no es una extencion de Jaca pero es toatamelete interoperable con él. Scala se traduce a bytecodes Java y la eficiencia de lso programas compilados por lo general es igual que Java. Bueno pasando un poco de la teoría, vamos a instalarlo cn Ubuntu con el siguiente comando: *"sudo apt-get install scala"*

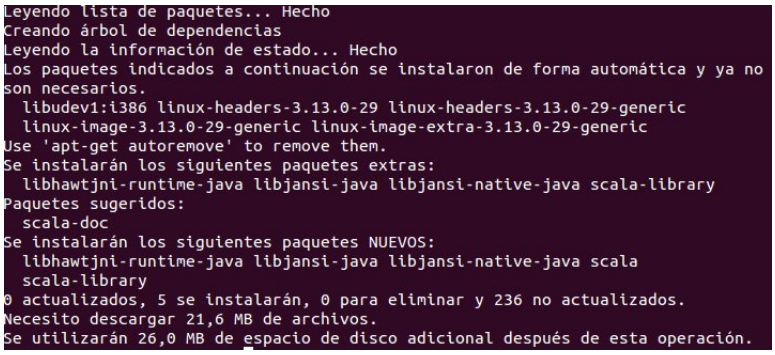

Figura 5.4: Instalación de Scala.

El código de hola mundo seria el siguiente:

```
object HolaMundo {
  def main(args: Array[String]) {
    println("Hola Mundo! " + args.toList)
  }
}
```

Figura 5.5: Código de hola mundo.

Guardaremos el codigo anterior para poder compiilarlo con el siguiete comando.

```
:~/Escritorio$ scalac holaMudo.scala
:~/Escritorio$
```

Figura 5.6: Ejecución programa en Scala.

Una vez compilado obtendremos dos nuevos archivos holaMundo.class u holaMundo $.class. Si hay errores durante la compilacion seran mostrados en el terminal.

```
holaMudo.scala:3: error: illegal character
            println(“Hola Mundo”)
                    ^
holaMudo.scala:3: error: illegal character
            println(“Hola Mundo”)
                                ^
holaMudo.scala:1: error: expected class or object definition
objetc HolaMundo {
^
three errors found
```

Figura 5.7: Errores en Scala.

El terminal nos muestra la salida del programa. List() nos indica los parametros que le pasamos a la funcion. Esto se muestran por pantalla por la linea:
println("Hola Mundo! - args.toList);
El resultado es el siguiente:
Hola Mundo! List(Parametro1, Parametro2)

```
blunt@blunt:~/Escritorio$ scala HolaMundo
Hola Mundo! List()
blunt@blunt:~/Escritorio$ scala HolaMundo Parametro1 Parametro2
Hola Mundo! List(Parametro1, Parametro2)
```

Figura 5.8: Ejecución programa con Scala.

Para instalar Gatling en Ubuntu podemos hacerlo desde los repositorios que vienen in-

cluidos con el siguiente comando *sudo apt-get install gatling.* [6]

Cuestión opcional 3: Lea el artículo y elabore un breve resumen.

El articulo hace una comparación entre dos benchmark conocidos; Gantling y Jmeter.
El articulo muestra su total apoyo a ambos proyectos y realiza comparaciones de los benchmark en distintas situaciones mostrandos sus diferencias en unas graficas de memoria y CPU, al final muestran sus diferencias y las conclusiones a las que llegan:

"Tanto JMeter y Gatling demostraron las características deseadas de relativamente planas tiempos de respuesta de las transacciones medidas durante RampUp y bajo carga, con poca variación. Tiempo medio de respuesta no debe ser utilizado como una medida del rendimiento de la herramienta a un lado de la observación previa en este sentido. Tanto JMeter y Gatling fueron capaces de mantener un rendimiento promedio en la región de 30.000 peticiones por minuto con ninguna desviación. Tanto JMeter y Gatling fueron RampUp capaz de 10.000 usuarios simultáneos en 10 minutos, que se considera normalmente un objetivo agresivo desde un solo generador de carga. Tanto JMeter y Gatling demostraron comportamiento de caché correcta, sobre todo cuando hace peticiones condicionales para recursos estáticos que responden con un HTTP 304. Gatling fueron capaces de proporcionar rápidamente con un parche para asegurar que este . Ambos planes de prueba JMeter y Gatling incluyen extracción de contenido a través de expresiones regulares desde el cuerpo de la respuesta, así como las afirmaciones de texto y códigos de respuesta HTTP contenidos sin menoscabo de su rendimiento."
En definitiva el articulo no recomienda uno u otro, termina diciendo que el uso de uno u otro en mas bien "subjetivo".

Cuestión opcional 4: Seleccione un benchmark entre SisoftSandra y Aida. Ejecútelo y muestre capturas de pantalla comentando los resultados.

[7] He elegido SisoftSandra.
He realizado dos tests, con dos configuraciones distintas:
1a 4 CPU – 1 GB de RAM
2a 2 CPU – 2 GB de RAM
Realizamos los dos test, y obtenemos dos gráficos:

[6] http://gatling.io/docs/1.5.6/user_documentation/tutorial/first_steps_with_gatling.html#first-steps-with-gatling
[7] http://www.sisoftware.co.uk/

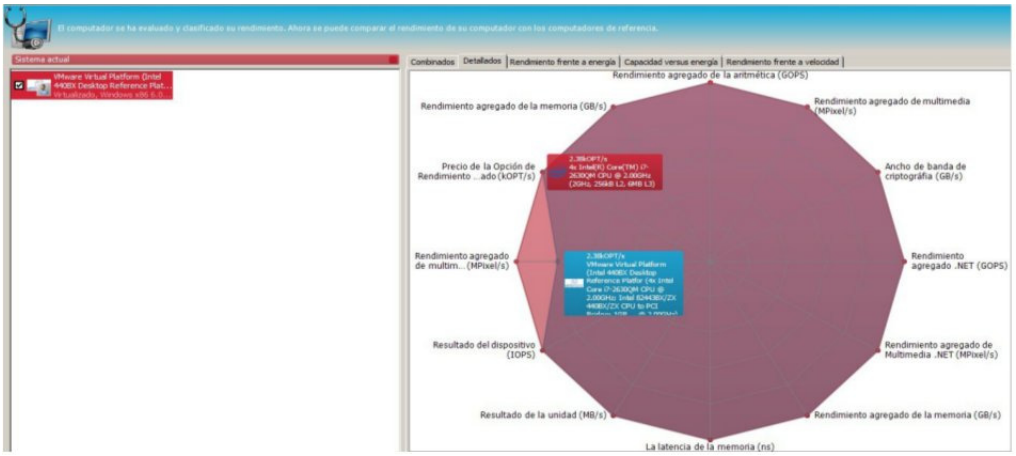

Figura 5.9: Grafico de la primera configuración.

Después, cambiamos las opciones de la máquina virtual para establecer la segunda configuración y volvemos a pasar el benchmark. Hecho esto, obtenemos:

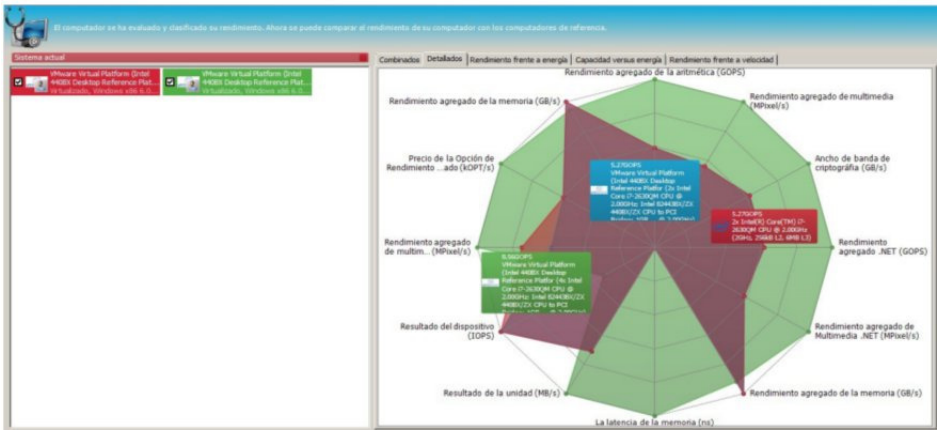

Figura 5.10: Segunda configuración.

Cómo vemos, la configuración de dos procesadores es inferior a la de 4 en rendimiento del procesador (lógicamente) pero hay algunas áreas en las que se nota el aumento en la memoria RAM.