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

Memoria Práctica 4

Pedro Torres Barrilado

22 de diciembre de 2015

Índice

Índice de figuras

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

Tras instalar phoronix-test-suite de su página oficial, para poder listar los benchamrks disponibles, se realiza con el comando: [1]

phoronix-test-suite list-available-tests.

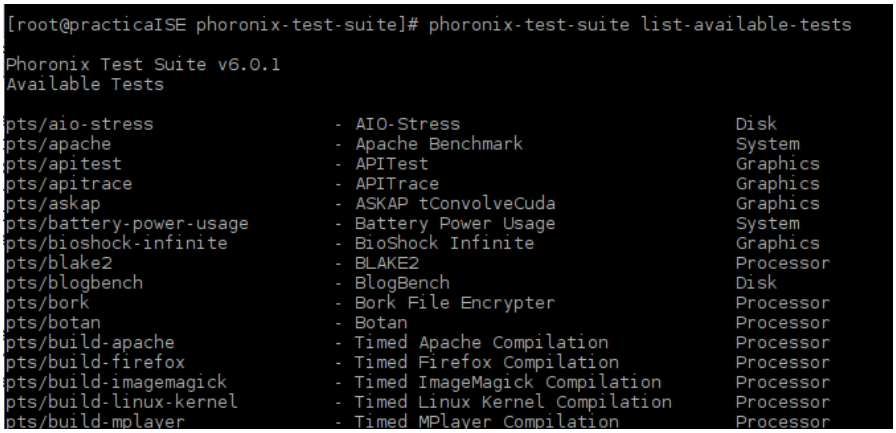


Figura 1.1: Test disponibles en phoronix test suite.

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

-c: sirve para indicar el número de peticiones múltiples que se van a realizar en el tiempo, en este caso 5.[2]

-n: número total de peticiones que se van a ejecutar en la sesión de benchmarking, en este caso 10. [3]

Primeramente en la máquina 1 tenemos ejecutándose un servidor apache, el cual se puede acceder desde la máquina 2, donde vamos a hacer el benchmark, por lo tanto en la máquina 2 introducimos .ªb -n 1000 -c 100 http://10.0.2.15/.el cual se encarga de realizar el benchmark al apache de la máquina 1, que nos da como resultado:

[1] `http://www.phoronix-test-suite.com/documentation/phoronix-test-suite.pdf`
[2] Página manual ab.
[3] Página manual ab.

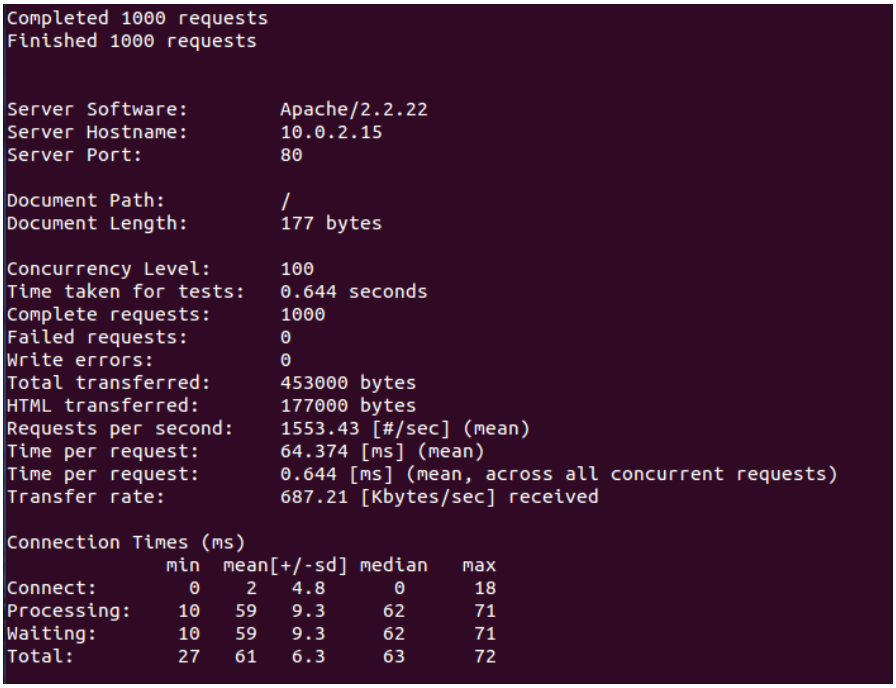


Figura 2.1: Resultado ejecución ab.

El número de hebras que se crean, en este caso son 100, debido a que estamos indicando con la orden -c, el número de hebras que van a realizar una petición a ese servidor a la vez[4].

[4]

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

Desde mac os x, vamos a realizar un apache benchmark a ubuntu server, centos, y windows server.[5]

Hacia Ubuntu Server:

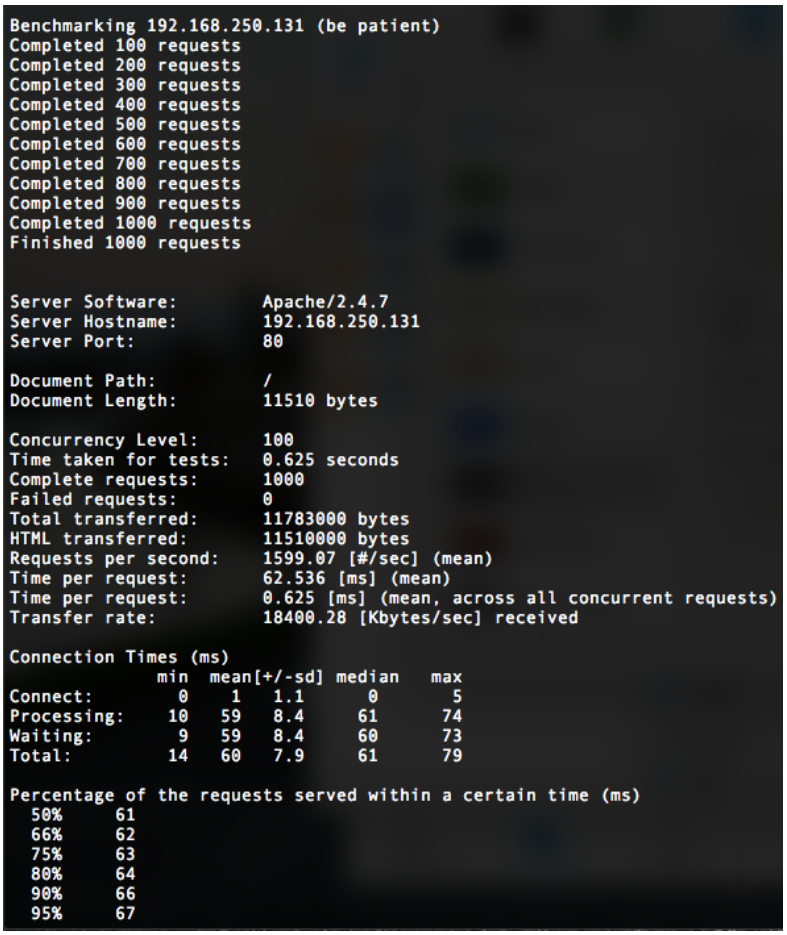


Figura 3.1: Resultado ejecución ab contra Ubuntu Server.

[5]Página de manual ab

Hacia CentOs:

```
Benchmarking 192.168.250.132 (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:        192.168.250.132
Server Port:            80

Document Path:          /
Document Length:        4897 bytes

Concurrency Level:      100
Time taken for tests:   0.999 seconds
Complete requests:      1000
Failed requests:        0
Non-2xx responses:      1000
Total transferred:      5179000 bytes
HTML transferred:       4897000 bytes
Requests per second:    1001.26 [#/sec] (mean)
Time per request:       99.874 [ms] (mean)
Time per request:       0.999 [ms] (mean, across all concurrent requests)
Transfer rate:          5063.99 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   1.1      0       5
Processing:    48   96  78.8     71     360
Waiting:       48   96  78.8     71     360
Total:         48   97  79.9     72     366

Percentage of the requests served within a certain time (ms)
  50%     72
  66%     74
  75%     76
  80%     76
  90%    299
```

Figura 3.2: Resultado ejecución ab contra CentOs.

Hacia Windows Server:

```
Benchmarking 192.168.250.129 (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/7.5
Server Hostname:        192.168.250.129
Server Port:            80

Document Path:          /
Document Length:        689 bytes

Concurrency Level:      100
Time taken for tests:   0.399 seconds
Complete requests:      1000
Failed requests:        0
Total transferred:      954000 bytes
HTML transferred:       689000 bytes
Requests per second:    2507.42 [#/sec] (mean)
Time per request:       39.882 [ms] (mean)
Time per request:       0.399 [ms] (mean, across all concurrent requests)
Transfer rate:          2336.02 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    1   1.3      0       7
Processing:     9   37  13.2     32      75
Waiting:        9   37  13.2     32      75
Total:         13   38  12.9     33      75

Percentage of the requests served within a certain time (ms)
  50%     33
  66%     37
  75%     43
  80%     48
  90%     62
  95%     66
  98%     69
```

Figura 3.3: Resultado ejecución ab contra Windows Server.

Una vez realizados los benchmarks a las respectivas máquinas, podemos comprobar que los mejores resultados proporciona, es windows server, debido a que por ejemplo en comparación con ubuntu server, el tiempo tomado para el test es de 0.625 segundos y en windows server es de 0.399 segundos.

Aparte de ésto se puede ver también que las peticiones por segundo que realiza windows server son mucho mayores que en el resto de sistemas, en windows el número de peticiones por segundo es de 2507.42, en ubuntu server es de 1599.07 y por último centOS es de 1001.26. Una vez visto que windows server es el claro ganador, si nos detenemos en

analizar ubuntu server y centOS, se puede ver que ubuntu server es el ganador, debido a que como dicho anteriormente el tiempo tomado para los test es menor además de el número de peticiones que es mayor. Esto pienso que aparte de el sistema operativo se puede deber a que la versión de apache que usa ubuntu server es mayor que la versión de apache que usa centOS.

El número de bytes transferidos es distinto en cada sistema, debido a que la tasa de transferencia que se produce entre cada sistema es distinta, teniendo a ubuntu server con la mayor tasa de transferencia.

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, etc.).

Primeramente descargamos Jmeter de su página oficial [6], donde se nos descargarg un archivo tgz que descomprimiremos con la orden tar -xf .ªrchivojmeter.tgz". (En este caso si no tenemos java, es necesario instalar su versión 6 o posterior)

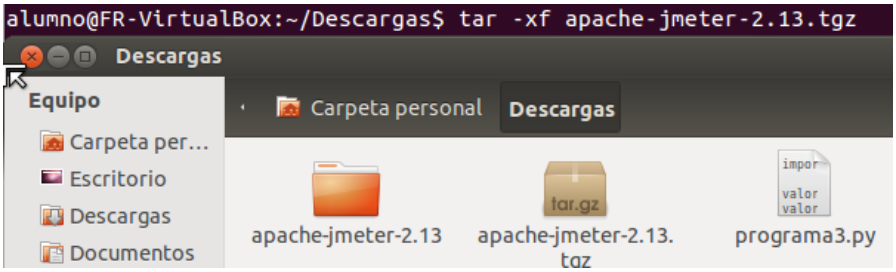


Figura 4.1: Descompresión jmeter.

[6] http://jmeter.apache.org/download_jmeter.cgi

Tras tenerlo descomprimido, nos dirigimos a la carpeta descomprimida de jmeter y dén-
tro de ésta a "bin":

Figura 4.2: Carpeta bin jmeter.

Una vez localizados en esa carpeta, para arrancar introducimos la orden "./jmeter", la cual comenzaría a arrancar jmeter.

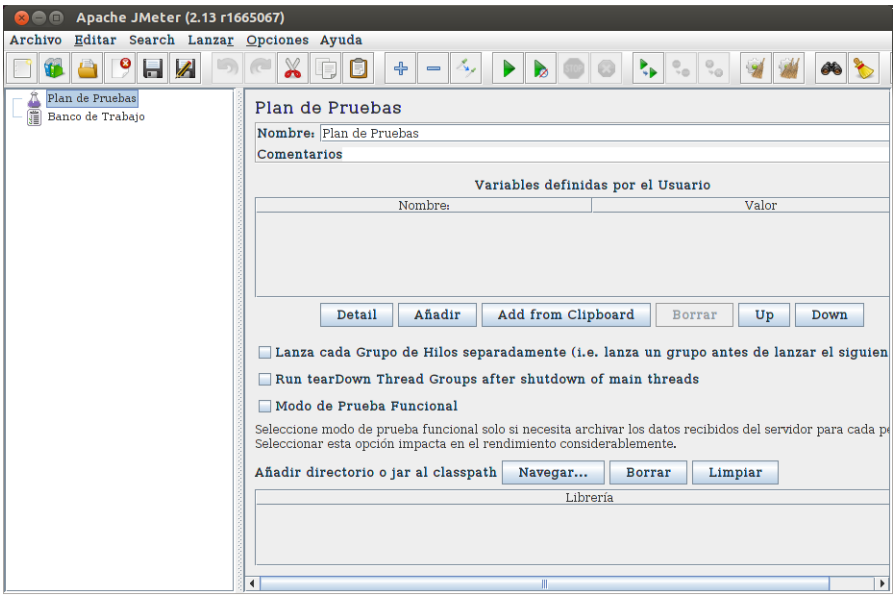


Figura 4.3: Ejecución jmeter.

Vamos a probar un test a una página web, la cual está siendo realizada con un servidor de apache, la cual es su página por defecto. Para realizar un test, primeramente añadimos un grupo de hilos[7]

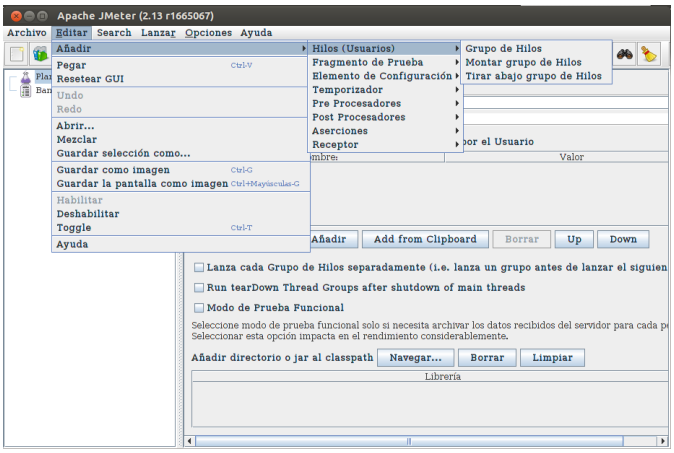


Figura 4.4: Añadir grupo de hilos jmeter.

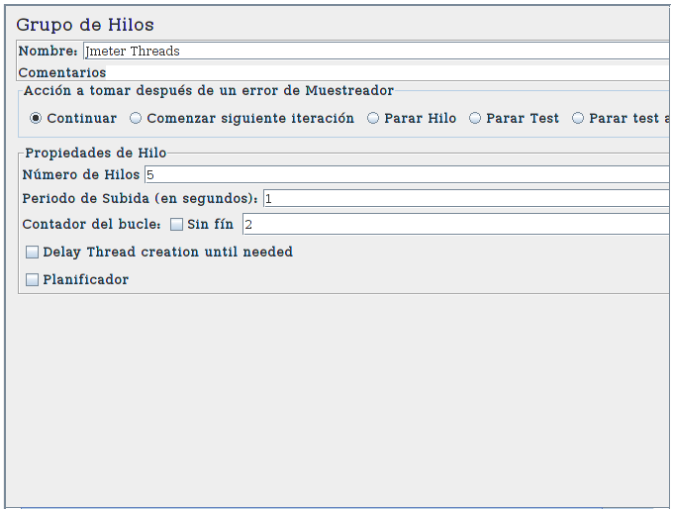


Figura 4.5: Información grupo de hilos jmeter.

Los parámetros que hemos modificado son el número de hilos que le damos un valor 5, el tiempo de periodo de subida (que es el tiempo en el que jmeter va a ir empezando cada hilo) le damos un valor en tiempo de 1 segundo y contador de bucle, que es el número de veces que se va a repetir el test.

[7] `http://jmeter.apache.org/usermanual/build-web-test-plan.html`

Una vez creados el grupo de hilos, es necesario indicar que van a realizar, en este caso peticiones http , con lo cual vamos a añadir una configuración de http para indicar posteriormente a donde queremos realizar esas peticiones: (click derecho en hilos, añadir, elemento de configuración, valores por defecto de http).

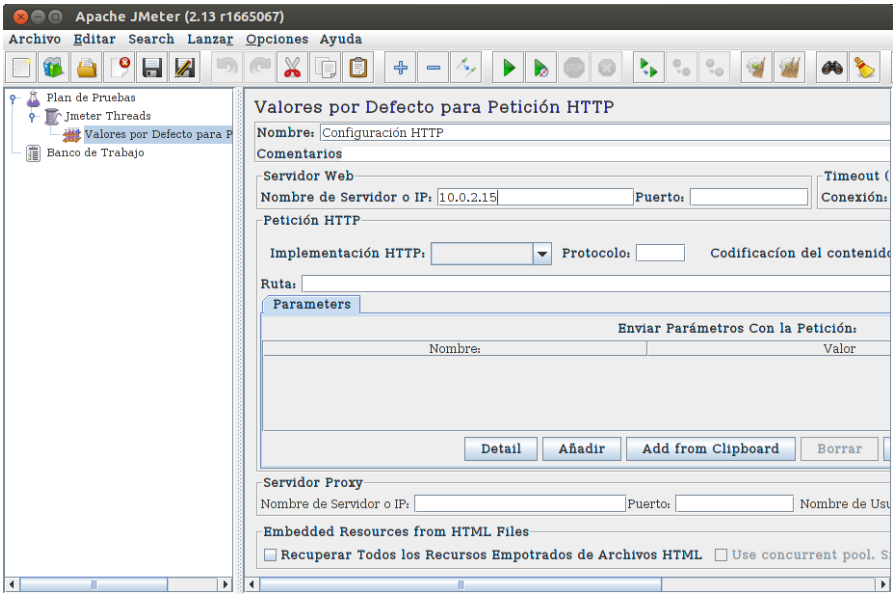


Figura 4.6: Valores por defecto de petición http jmeter.

Los parámetros que hemos modificado son el nombre que le hemos dado a esa configuración y la dirección donde se encuentra nuestro servidor.

Una vez realizada la configuración, es necesario añadir las peticiones (click derecho en hilos, añadir, muestreador, petición http). Realizamos dos peticiones a las rutas, una que va a ser a “/” y otra a /changes".

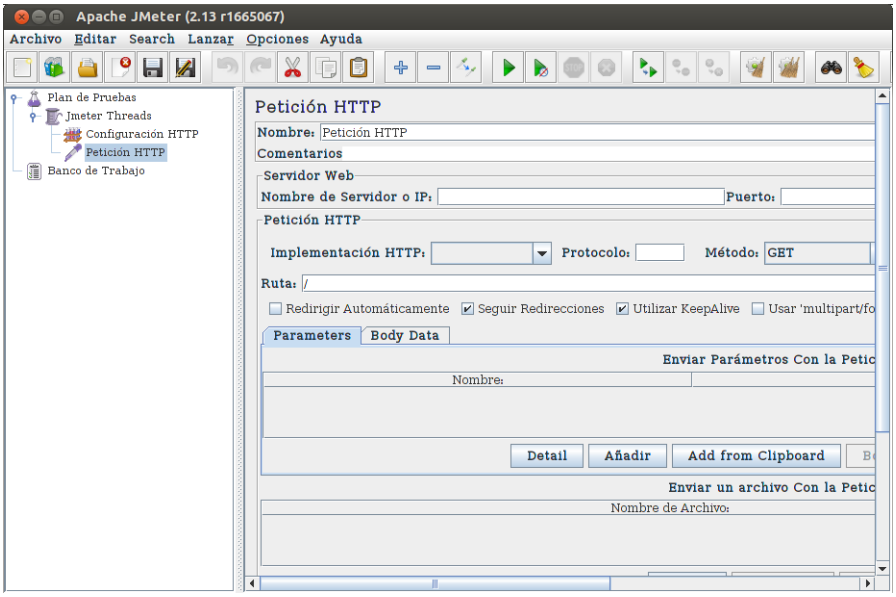


Figura 4.7: Petición http jmeter.

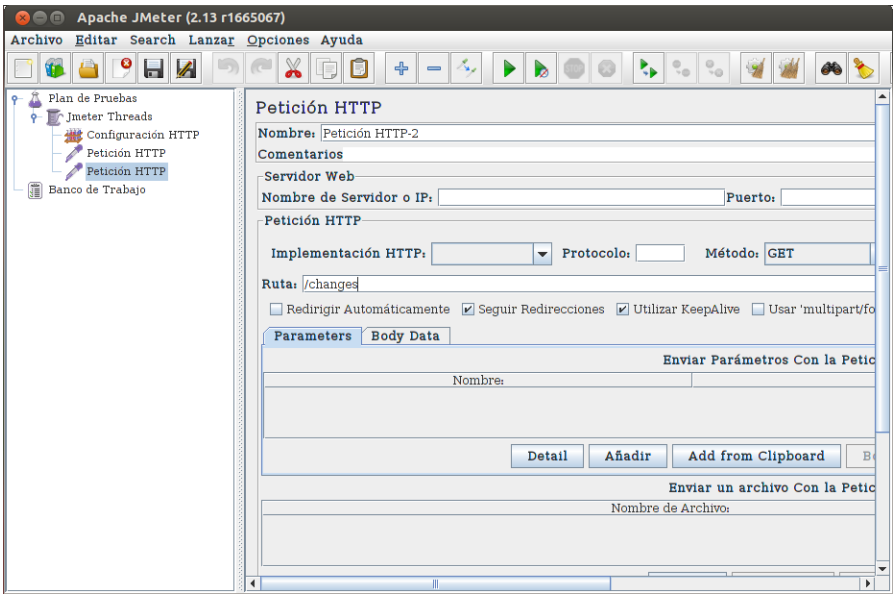


Figura 4.8: Petición http jmeter.

Una vez realizadas todas las peticiones, ya nos disponemos a mostrar los datos añadiendo un "Listener", el cual se encarga de mostrarnos los datos gráficamente. (click derecho en hilos, añadir, receptor, gráfico de resultados). Una vez añadido, le damos a empezar con la tecla superior "play"de color verde y ya nos mostraría los resultados:

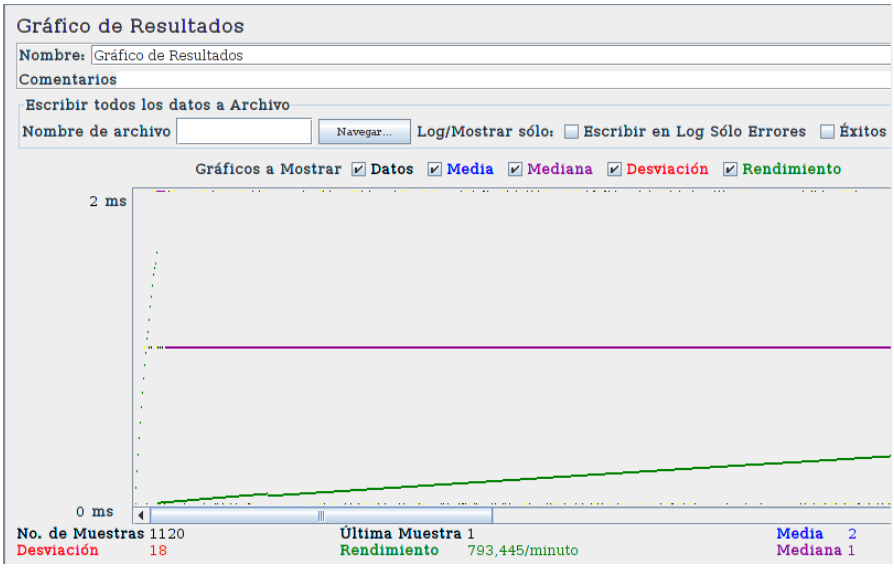


Figura 4.9: Gráfico resultados jmeter.

Nos muestra aparte de el número de muestras, la información respecto a la media, la desviación, mediana y rendimiento, mostrado por colores que se indican en la gráfica.

5. Programe un benchmark usando el lenguaje que desee.

5.1. 1)Objetivo del benchmark

Lo que realiza este benchmark es testear tanto la cpu como la memoria. Para la prueba de la cpu, la realiza usando operaciones en coma flotante, las cuales incluyen tanto multiplicación como suma.[8]
Para la memoria lo que hace es ordenar un vector usando el algoritmo de selección, el cual es el más indicado ya que se realizan de los máximos accesos a memoria, esto es necesario para intentar estresar lo máximo posible la memoria.

[8] `https://es.wikipedia.org/wiki/Unidad_de_coma_flotante`

5.2. 2)Métricas

Lo que se utiliza en este benchmark para la cpu son 3 vectores, los cuales dos de ellos se suman y se multiplican por un escalar[9][10] y después se multiplican guardando los valores de cada posición en otro vector. El tamaño de los vectores son de 50000 elementos y las veces que se realiza este procedimiento indicado en el código de abajo son 5000 veces.

```
for(k=0; k<=p; k++){

                    for(i=0; i<=n; i++){
                            a=(rand()%10);
                            b=(rand()%10);
                            d=(rand()%10);
                            x[i]=a;
                            y[i]=b;
                            z[i]=d;
                    }

                    clock_gettime(CLOCK_REALTIME,&cgt1);
                    //Multiplicacion vector por escalar
                    for(j=0; j<=n; j++){
                            y[j]=c*x[j]+y[j];
                    }

                    //Multiplicacion de vectores
                    for(i=0; i<=n; i++){
                            mul=x[i]*y[i];
                            z[i] = mul;
                    }

                    clock_gettime(CLOCK_REALTIME,&cgt2);
                    ncgt=(double) (cgt2.tv_sec-cgt1.tv_sec)+(double) ((cgt2.
                        tv_nsec-cgt1.tv_nsec)/(1.e+9));
                    ttotal += ncgt;
```

Para la memoria el vector resultante de la multiplicación por el escalar se ordena.

```
clock_gettime(CLOCK_REALTIME,&cgt1);
ordenacion(y,n);
clock_gettime(CLOCK_REALTIME,&cgt2);
```

El tiempo se mide gracias a la biblioteca time, que nos da información del tiempo en segundos.

[9] http://top500.org/project/linpack/
[10] https://es.wikipedia.org/wiki/Linpack

La puntuación total se da sumando la puntuación dada en el benchmark de la cpu junto la puntuación dada en el benchmark de la memoria. La puntuación de cada benchmark se da dividiendo 1 entre el tiempo que tarda esa ejecución. Cuanto mayor sea la puntuación mejor calificación se le da a la máquina.

```
titer = ttotal/p;
        int aux =  1/titer;
        int aux2 = 1/tmem;
        aux +=aux2;
        printf("************************************\n");
        printf("Tiempo por iteracion: %f\n", titer);
        printf("Tiempo total CPU: %f\n", ttotal);
        printf("Tiempo en ordenar el vector: %f\n",tmem);
        printf("-> PUNTUACION: %d\n",aux);
        printf("************************************\n");
```

5.3. 3)Instrucciones de uso

Para compilarlo es necesario añadir -lrt para que funcione la librería time:

"gcc benchej5.c -o benchej5 -lrt"

Para ejecutarlo directamente con "./benchej5" iniciaría.

5.4. 4)Resultados

```
alumno@FR-VirtualBox:~/Escritorio/Benchmarks$ ./ej5
********************************
Sistema Operativo = Linux.
Arquitectura = 32 bits
********************************

Iniciando benchmark...

******Benchmark para la CPU: ******
Progreso: 25%...
Progreso: 50%...
Progreso: 75%...

******Benchmark para la memoria: ******
Progreso: 25%...
Progreso: 50%...
Progreso: 75%...

************************************
Tiempo por iteración: 0.000358
Tiempo total CPU: 1.790615
Tiempo en ordenar el vector: 3.809316
-> PUNTUACION: 2792
************************************
```

Figura 5.1: Resultado ejecución benchmark Ubuntu.

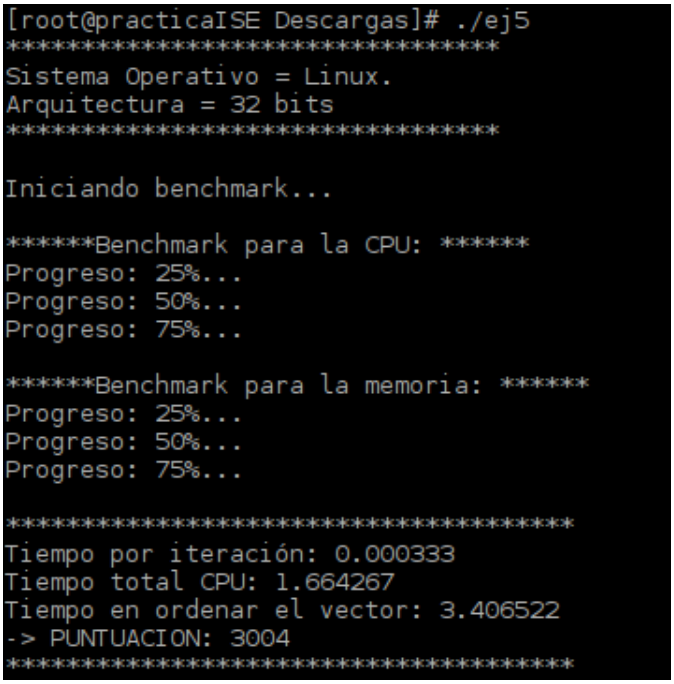


Figura 5.2: Resultado ejecución benchmark CentOs.

Este sería un resultado de la ejecución de el benchmark. Nos indica el sistema operativo que se está usando tanto como su arquitectura. Tras eso empieza a realizarse el benchmark, donde indica paso a paso cómo se va desarrollando la ejecución. Una vez que ha acabado, muestra el tiempo que ha tardado una iteración en realizar una operación con vectores la cpu, el tiempo total que ha tardado en realizar todas las iteraciones con la operación de vectores, el tiempo que ha tardado en realizar la ordenación del vector y por último la puntuación dada por el benchmark.

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

Instalamos por ejemplo, tras comprobar que está disponible, un benchmark que usa la cpu, en este caso el de "sample-programçon el comando "phoronix-test-suite install pts/sample-program"

Una vez instalado lo ejecutamos con el comando "phoronix-test-suite run sample-program". Lo que realiza este test que calcula 87.654.321 dígitos de el número Pi usando la fórmula Leibniz. [11]

[11] `http://openbenchmarking.org/test/pts/sample-program-1.1.0`

Una vez terminado el test, nos muestra en un html los resultados, incluyendo el las características de nuestra máquina:

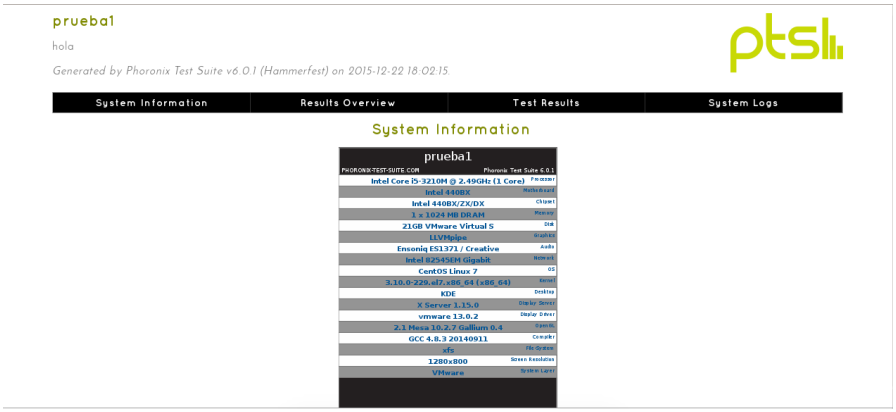


Figura 6.1: Resultado test phoronix test suite.

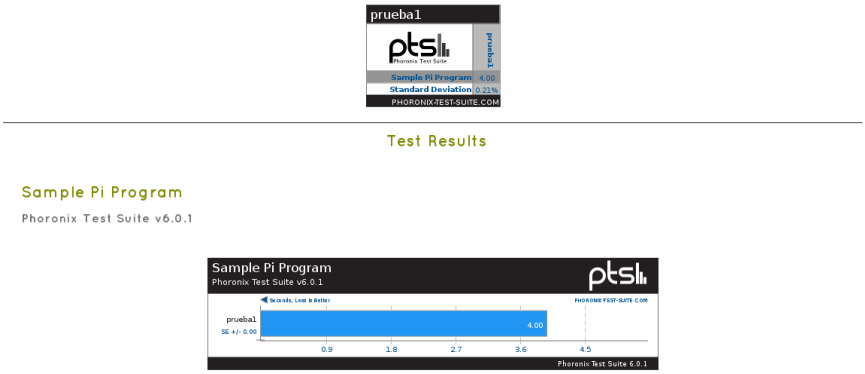


Figura 6.2: Resultado test phoronix test suite.

Esta última gráfica nos muestra el tiempo que tarda en realizarlo en segundos, en este caso nos da un valor 4. Cuanto menos segundos significa que nuestra máquina es mucho mejor. Si quieremos comparar los resultados con otras máquinas con distinto hardware y sistema operativo, en la página `http://openbenchmarking.org/test/pts/sample-program-1.1.0`, tenemos opción de ver distintos resultados de ese mismo test.

7. [OPCIONAL 2] ¿Qué es Scala? Instale Gatling y pruebe los escenarios por defecto.

SCALA: [12] es un lenguaje de programación multi-paradigma orientado a objetos el cual nos ayuda a expresar patrones de programación de forma sencilla y ordenada. Funciona bajo una máquina virtual de java, por lo que es uso entre scala y java es muy dinámico y no supone ningún problema.

Tras descargarlo, lo ejecutamos y nos muestra un menú donde vamos a seleccionar el stress que vayamos a realizar:

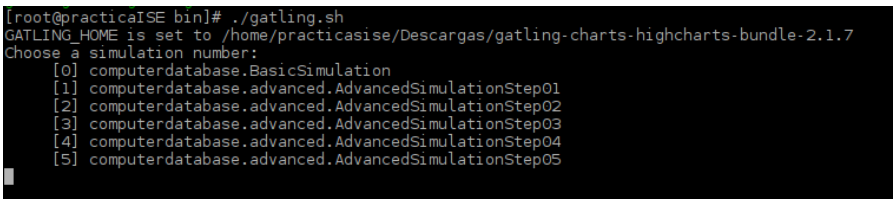


Figura 7.1: Ejecución gatling.

Pulsamos "0" y empezará a realizarse el basicSimulation, tras indicar una id y una descripción:

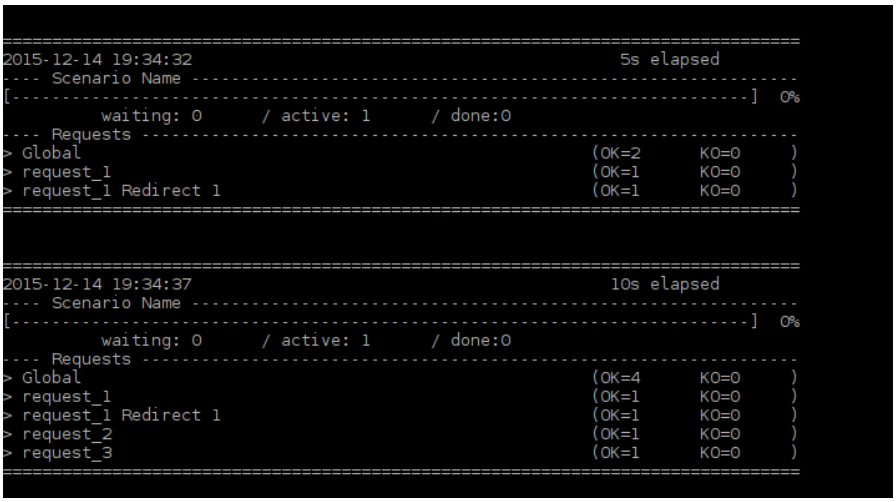


Figura 7.2: Ejecución gatling.

Irán saliendo el proceso por la consola, y al acabar, nos mostrará los resultados generales del test realizado.

[12] https://es.wikipedia.org/wiki/Scala_(lenguaje_de_programaci%C3%B3n)

```
================================================================================
---- Global Information --------------------------------------------------------
> request count                                         13 (OK=13     KO=0     )
> min response time                                     73 (OK=73     KO=-     )
> max response time                                    149 (OK=149    KO=-     )
> mean response time                                    92 (OK=92     KO=-     )
> std deviation                                         23 (OK=23     KO=-     )
> response time 50th percentile                         81 (OK=81     KO=-     )
> response time 75th percentile                        103 (OK=103    KO=-     )
> mean requests/sec                                 0.556 (OK=0.556  KO=-     )
---- Response Time Distribution ------------------------------------------------
> t < 800 ms                                            13 (100%)
> 800 ms < t < 1200 ms                                   0 (  0%)
> t > 1200 ms                                            0 (  0%)
> failed                                                 0 (  0%)
================================================================================
```

Figura 7.3: Resultados gatling.

Si queremos ver los resultados en un html, nos dirigimos a la carpeta results donde tengamos instalado gatling que nos indica la consola.

Una vez dentro del archivo html nos muestra muchos datos respecto al test, como la información global, estadísticas, número de peticiones por segundo, etc.

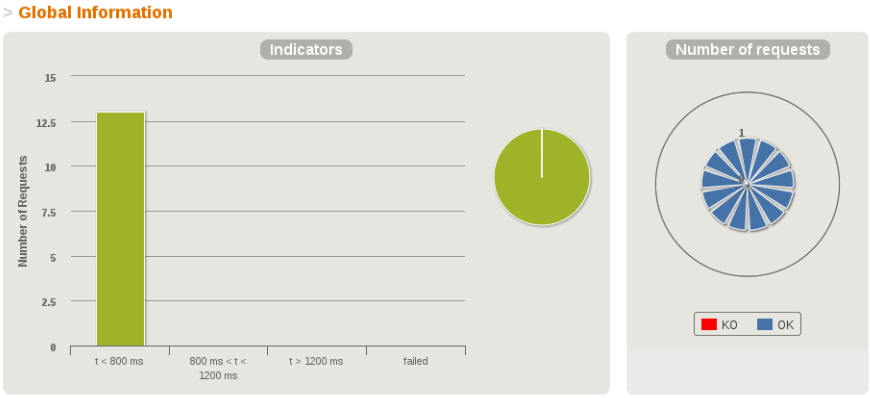
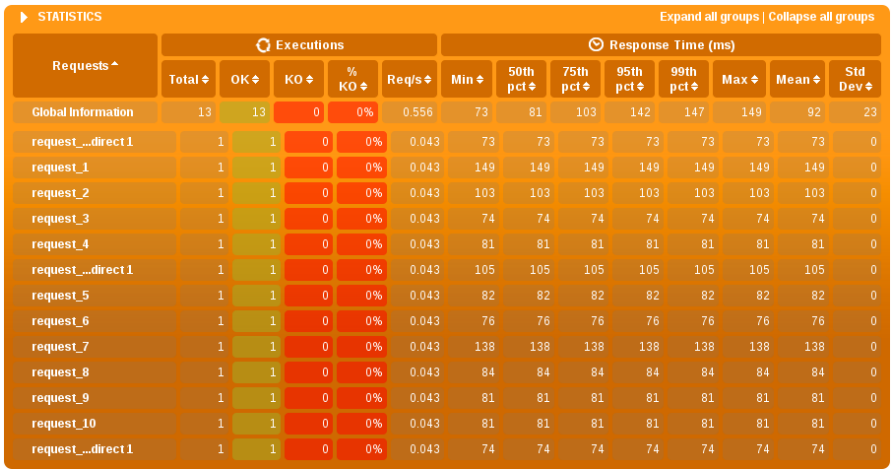


Figura 7.4: Página html gatling.



STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Req/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	13	13	0	0%	0.556	73	81	103	142	147	149	92	23
request_...direct 1	1	1	0	0%	0.043	73	73	73	73	73	73	73	0
request_1	1	1	0	0%	0.043	149	149	149	149	149	149	149	0
request_2	1	1	0	0%	0.043	103	103	103	103	103	103	103	0
request_3	1	1	0	0%	0.043	74	74	74	74	74	74	74	0
request_4	1	1	0	0%	0.043	81	81	81	81	81	81	81	0
request_...direct 1	1	1	0	0%	0.043	105	105	105	105	105	105	105	0
request_5	1	1	0	0%	0.043	82	82	82	82	82	82	82	0
request_6	1	1	0	0%	0.043	76	76	76	76	76	76	76	0
request_7	1	1	0	0%	0.043	138	138	138	138	138	138	138	0
request_8	1	1	0	0%	0.043	84	84	84	84	84	84	84	0
request_9	1	1	0	0%	0.043	81	81	81	81	81	81	81	0
request_10	1	1	0	0%	0.043	81	81	81	81	81	81	81	0
request_...direct 1	1	1	0	0%	0.043	74	74	74	74	74	74	74	0

Figura 7.5: Página html gatling.

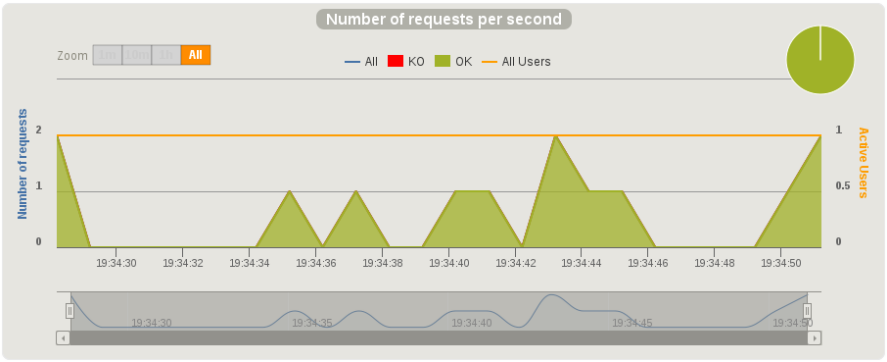


Figura 7.6: Página html gatling.

Ahora procedemos a ejecutar un nuevo benchmark, por ejemplo el Advanced Simulation Step 01, el cual nos da de resultado:

```
================================================================================
---- Global Information --------------------------------------------------------
> request count                                         13 (OK=13     KO=0     )
> min response time                                     73 (OK=73     KO=-     )
> max response time                                    179 (OK=179    KO=-     )
> mean response time                                    88 (OK=88     KO=-     )
> std deviation                                         27 (OK=27     KO=-     )
> response time 50th percentile                         81 (OK=81     KO=-     )
> response time 75th percentile                         85 (OK=85     KO=-     )
> mean requests/sec                                   0.91 (OK=0.91   KO=-     )
---- Response Time Distribution ------------------------------------------------
> t < 800 ms                                            13 (100%)
> 800 ms < t < 1200 ms                                   0 (  0%)
> t > 1200 ms                                            0 (  0%)
> failed                                                 0 (  0%)
================================================================================
```

Figura 7.7: Prueba benchmark gatling.

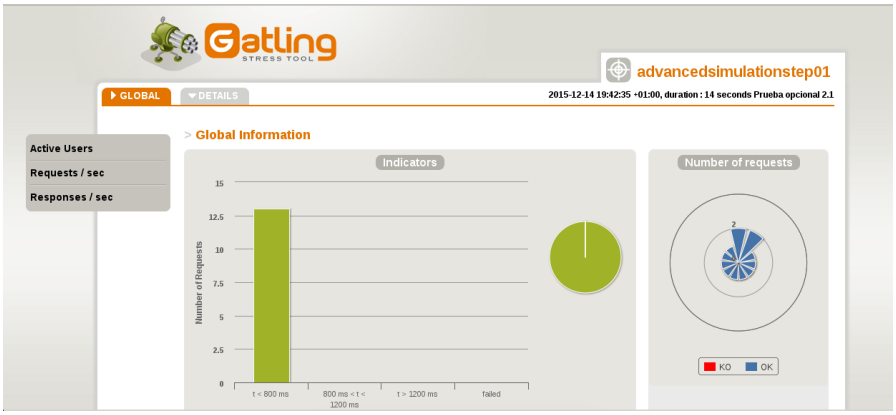


Figura 7.8: Página benchmark gatling.

8. [OPCIONAL 4]Seleccione un benchmark entre SisoftSandra y Aida. Ejecútelo y muestre capturas de pantalla comentando los resultados.

El benchmark que he seleccionado es Aida.[13]

Una vez instalado, nos encontramos con la página principal, donde para realizar un análisis pinchamos en un icono parecido a una "terminal", llamado System Stability Test.

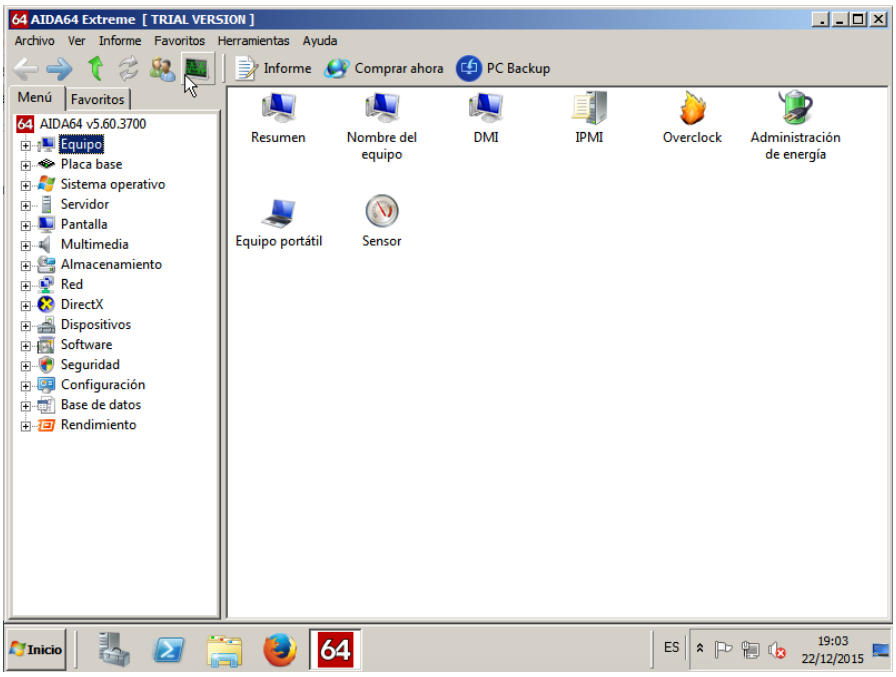


Figura 8.1: Página inicio Aida64.

[13] http://www.aida64.com/downloads

Se nos abrirá una ventana donde seleccionaremos a que queremos realizar el test y pulsamos abajo "Start", donde empezará el test. Cuando queramos pararlo pulsamos "stop".

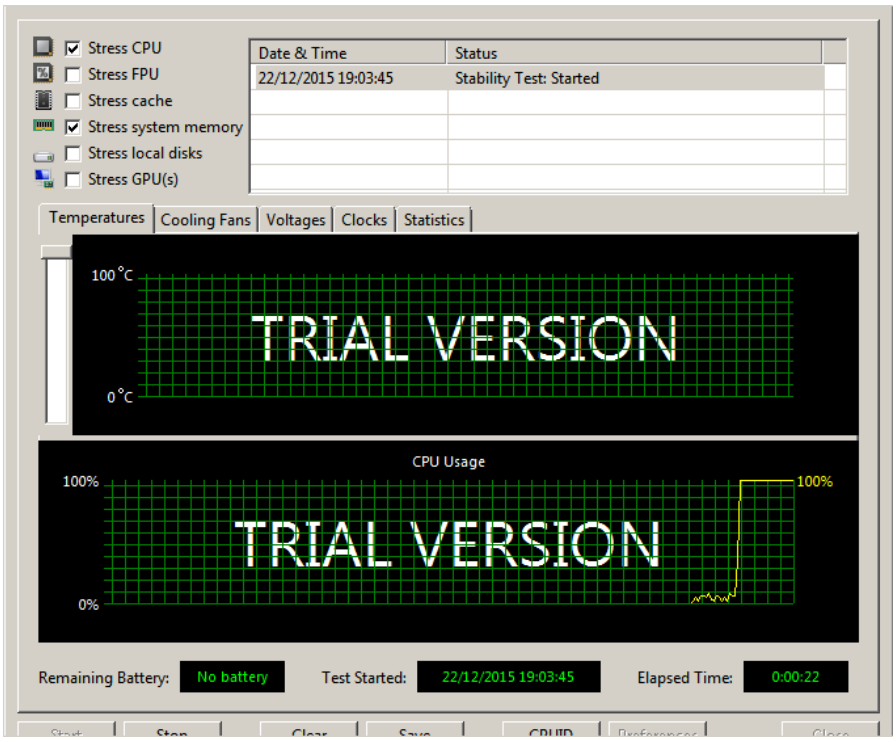


Figura 8.2: Prueba Aida.

Dentro de esta ventana nos da la información en las pestañas tanto de la temperatura, la velocidad del ventilador, el voltaje y la velocidad del reloj. A la vez que en la parte superior hay un log que indica los tests realizados y el tiempo que han transcurrido cada uno de ellos.