

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

Memoria Práctica 4

Christian Andrades Molina

22 de diciembre de 2015

Índice

1. Instale la aplicación. ¿Qué comando permite listar los benchmarks disponibles?	4
2. Cuestión 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?	5
3. Cuestión 3: 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?	6
4. Cuestión 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.).	9
5. Cuestión 5: 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	13

Índice de figuras

1.1. Instalación de Phoronix Suite	4
1.2. available-suites	4
1.3. available-tests	4
1.4. available-virtual-suites	4
2.1. ab -n 1000 -c 30 http://localhost/	5
3.1. 'ab' sobre la máquina Ubuntu Server	6
3.2. 'ab' sobre la máquina CentOS	7
3.3. 'ab' sobre la máquina Windows Server	8
4.1. Instalacion de JMeter	9
4.2. Menú de JMeter	9
4.3. Thread Group	10
4.4. HTTP Request	10
4.5. HTTP Cookie Manager	11
4.6. HTTP Request 1	11
4.7. HTTP Request 2	12
4.8. Graph Results	12
5.1. Uso del Benchmark	13

5.2. Código del Benchmark 1	14
5.3. Código del Benchmark 2	14
5.4. Tiempos de ejecución del Benchmark en CentOS 1	15
5.5. Tiempos de ejecución del Benchmark en CentOS 2	15
5.6. Comparativa en gráfica.	16

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

Para instalar Phoronix Suite [4] [3] introducimos en Ubuntu Server la siguiente sentencia: `sudo apt-get install phoronix-test-suite`.

```
root@ubuntu:~# sudo apt-get install phoronix-test-suite
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes NUEVOS:
  phoronix-test-suite
0 actualizados, 1 se instalarán, 0 para eliminar y 50 no actualizados.
```

Figura 1.1: Instalación de Phoronix Suite

Una vez instalado, tenemos acceso a la lista de Benchmarks disponibles con estas 3 sentencias [2]:

1. Con esta opción tenemos todas las pruebas disponibles: `list-available-suites`

```
Available Suites
pts/audio-encoding      - Audio Encoding      System
pts/chess               - Chess Test Suite    Processor
pts/compilation         - Timed Code Compilation Processor
pts/compiler            - Compiler             Processor
pts/compression         - Timed File Compression Processor
```

Figura 1.2: available-suites

2. Con esta opción tenemos una lista de los perfiles de las pruebas disponibles: `list-available-tests`

```
pts/system-decompress-xz - System XZ Decompression Processor
pts/system-decompress-zlib - System ZLIB Decompression Processor
pts/system-libjpeg      - System JPEG Library Decode Processor
pts/system-libxml2      - System Libxml2 Parsing Processor
pts/system-boot-kernel  - System Kernel Boot Time Processor
pts/system-boot-total   - System Total Boot Time Processor
```

Figura 1.3: available-tests

3. Con esta opción tenemos una lista de las pruebas virtuales disponibles que pueden ser creadas dinámicamente basándose en los benchmarks ya disponibles: `list-available-virtual-suites`

```
root@ubuntu:~# phoronix-test-suite list-available-virtual-suites
Phoronix Test Suite v4.8.3
Available Virtual Suites
```

Figura 1.4: available-virtual-suites

2. Cuestión 2: 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?

Atendiendo al manual de ab [1], obtenemos información sobre ambos parámetros del comando. Por un lado, -c es el número de solicitudes enviadas con el fin de realizarse de forma concurrente, a la vez. Por tanto -c 5 serían 5 peticiones concurrentes. En cambio, -n nos indica el número de solicitudes a realizar durante el benchmark. En este caso, -n 100 daría lugar a 100 solicitudes para nuestro test.

Ejecutamos ab -n 1000 -c 30 http://localhost/ sobre nuestra máquina de CentOS y obtenemos:

```
Concurrency Level:      30
Time taken for tests:   1.064 seconds
Complete requests:     100
Failed requests:        0
Write errors:           0
Non-2xx responses:     104
Total transferred:     536536 bytes
HTML transferred:      515944 bytes
Requests per second:    94.01 [#/sec] (mean)
Time per request:       319.121 [ms] (mean)
Time per request:       10.637 [ms] (mean, across all concurrent requests)
Transfer rate:          492.57 [Kbytes/sec] received

Connection Times (ms)
  min   mean[+/-sd] median   max
Connect:    0    29  27.1    20   117
Processing:  2   154 140.0    82   500
Waiting:    1   150 137.1    79   492
Total:      20   183 154.7    91   565

Percentage of the requests served within a certain time (ms)
 50%    91
 66%   235
 75%   280
 80%   327
 90%   440
 95%   480
 98%   553
 99%   565
100%   565 (longest request)
[christian@localhost ~]$
```

Figura 2.1: ab -n 1000 -c 30 http://localhost/

3. Cuestión 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?

En este caso utilizaremos Windows 8 como SO anfitrión hacia las tres máquinas virtuales (Windows Server, Ubuntu Server y CentOS):

```
# ab -n 10000 -c 30 http://192.168.3.128/
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.3.128 (be patient)
Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
Completed 10000 requests
Finished 10000 requests

Server Software:      Apache/2.4.7
Server Hostname:      192.168.3.128
Server Port:          80

Document Path:        /
Document Length:      11510 bytes

Concurrency Level:     30
Time taken for tests:   48.106 seconds
Complete requests:     10000
Failed requests:        0
Total transferred:     117830000 bytes
HTML transferred:      115100000 bytes
Requests per second:   207.88 [#/sec] (mean)
Time per request:      144.317 [ms] (mean)
Time per request:      4.811 [ms] (mean, across all concurrent requests)
Transfer rate:         2391.99 [Kbytes/sec] received

Connection Times (ms)
  min      mean[+/-sd] median    max
Connect:    0       1   2.3      0     84
Processing:  9      143  93.9    124    690
Waiting:    3      130  93.5    109    688
Total:       9      144  93.9    124    690

Percentage of the requests served within a certain time (ms)
 50%    124
 66%    163
 75%    190
 80%    210
 90%    271
 95%    327
 98%    401
 99%    456
100%    690 (longest request)
```

Figura 3.1: 'ab' sobre la máquina Ubuntu Server

```

This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.3.129 (be patient)
Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
Completed 10000 requests
Finished 10000 requests


Server Software:      Apache/2.2.15
Server Hostname:      192.168.3.129
Server Port:          80

Document Path:        /
Document Length:      4961 bytes

Concurrency Level:    30
Time taken for tests:  46.793 seconds
Complete requests:     10000
Failed requests:       0
Non-2xx responses:     10000
Total transferred:     51590000 bytes
HTML transferred:      49610000 bytes
Requests per second:   213.71 [#/sec] (mean)
Time per request:      140.379 [ms] (mean)
Time per request:      4.679 [ms] (mean, across all concurrent requests)
Transfer rate:         1076.68 [Kbytes/sec] received


Connection Times (ms)
      min      mean[+/-sd] median   max
Connect:    0       1   2.1      1      55
Processing: 10     139 102.7    120   1459
Waiting:    4      127 101.0    109   1458
Total:      10     140 102.7    121   1460


Percentage of the requests served within a certain time (ms)
 50%    121
 66%    154
 75%    178
 80%    195
 90%    245
 95%    291
 98%    361
 99%    448
100%   1460 (longest request)

```

Figura 3.2: 'ab' sobre la máquina CentOS

```
# ab -n 10000 -c 30 http://192.168.3.130/
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.3.130 <be patient>
Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
Completed 10000 requests
Finished 10000 requests


Server Software:      Microsoft-IIS/8.0
Server Hostname:      192.168.3.130
Server Port:          80

Document Path:        /
Document Length:      1398 bytes

Concurrency Level:     30
Time taken for tests:   41.966 seconds
Complete requests:      10000
Failed requests:         0
Total transferred:      16420000 bytes
HTML transferred:       13980000 bytes
Requests per second:    238.29 [#/sec] (mean)
Time per request:       125.898 [ms] (mean)
Time per request:       4.197 [ms] (mean, across all concurrent requests)
Transfer rate:          382.10 [Kbytes/sec] received


Connection Times (ms)
      min      mean[+/-sd] median   max
Connect:    0       1    3.8      0   109
Processing:  0    125   75.4    109   625
Waiting:    0    123   75.1    109   578
Total:       0    126   75.4    109   625


Percentage of the requests served within a certain time (ms)
 50%    109
 66%    141
 75%    172
 80%    188
 90%    219
 95%    266
 98%    313
 99%    359
100%    625 (longest request)
```

Figura 3.3: 'ab' sobre la máquina Windows Server

En términos puramente numéricos, tenemos que:

- Windows Server: 41.966 segundos, 125.898 ms.
- CentOS: 46.793 segundos, 140.379 ms.
- Ubuntu Server: 48.106 segundos 144.317 ns.

Windows Server es 5 segundos y 7 (aproximadamente) más rápido a la hora de realizar la misma operación, tomando menos tiempo en cada petición recibida. El número de bytes transferidos difiere en cada máquina, siendo proporcional al tiempo requerido (Windows Server el más bajo y Ubuntu Server el más alto).

4. **Cuestión 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.).**

En primer lugar llevaremos a cabo la instalación de Jmeter. Para ello descargamos el archivo proporcionado por su página web, lo descomprimos y accedemos a él:

```
[christian@localhost Desktop]$ cd apache-jmeter-2.13
[christian@localhost apache-jmeter-2.13]$ cd bin
[christian@localhost bin]$ sh jmeter
Dec 16, 2015 8:23:36 PM java.util.prefs.FileSystemPreferences$1 run
INFO: Created user preferences directory.
```

Figura 4.1: Instalacion de JMeter

Una vez instalado e iniciado, se abrirá el siguiente menú donde llevaremos a cabo el test:

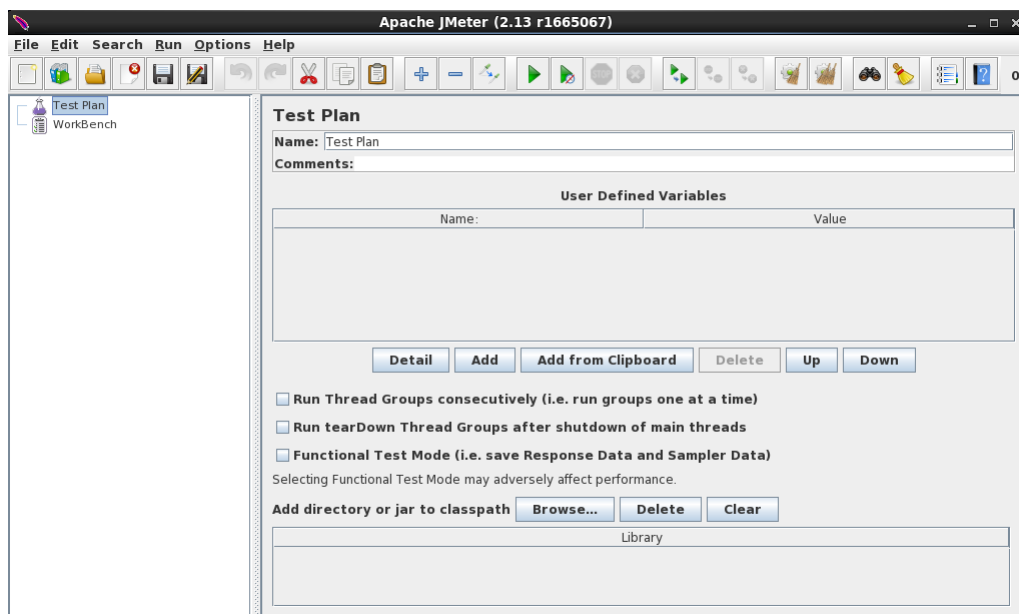


Figura 4.2: Menú de JMeter

El primer pasó será crear un grupo de hebra que determinará el número de usuarios a simular en nuestra prueba. Para ello, click secundario sobre Test Plan y añadimos un

ThreadGroup. Incrementamos el número de usuarios a 5.

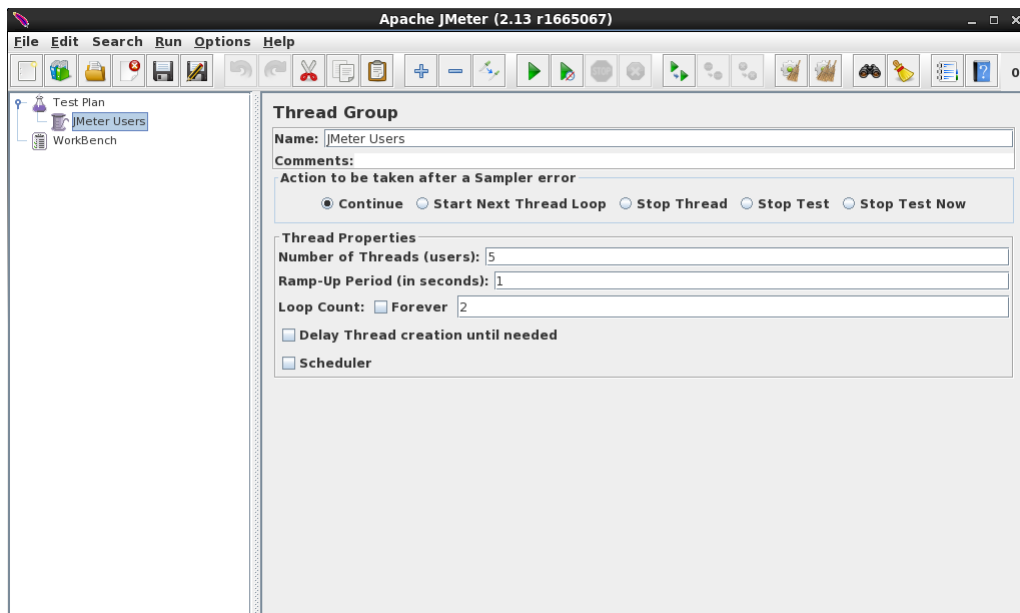


Figura 4.3: Thread Group

Crearemos un HTTP Request donde indicaremos la dirección web o IP sobre la que trabajar. En este caso será localhost:

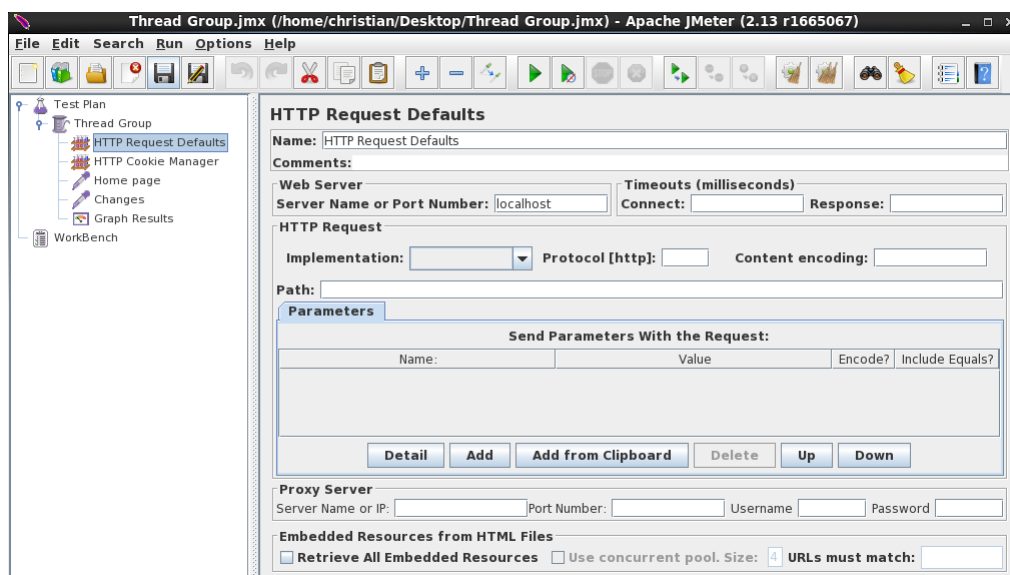


Figura 4.4: HTTP Request

Añadimos un soporte de Cookie's. Click secundario sobre Thread Group, add y HTTP Cookie Manager:

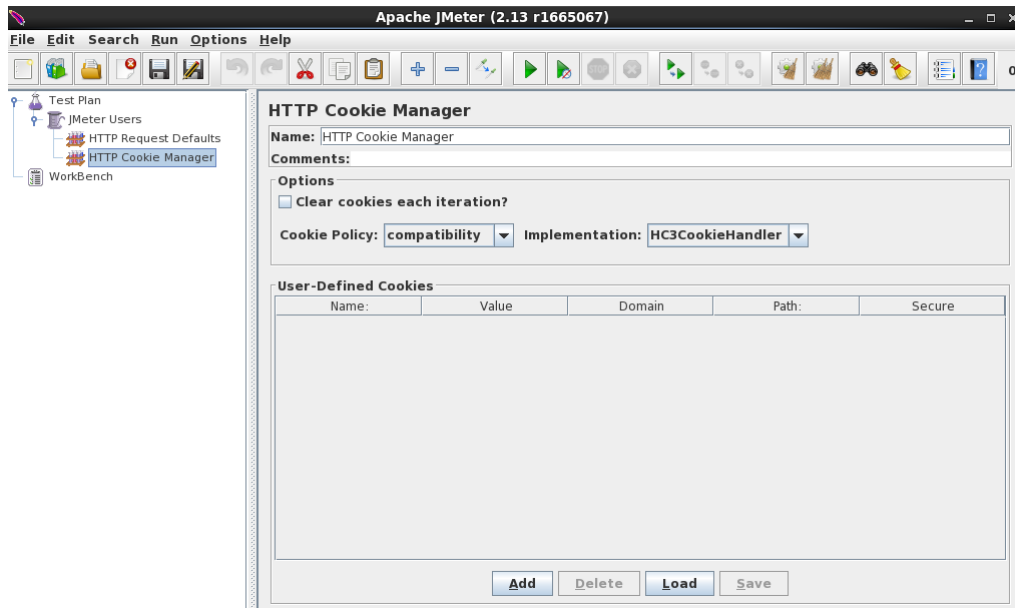


Figura 4.5: HTTP Cookie Manager

Dos HTTP Request al directorio Home Page y a un segundo para errores:

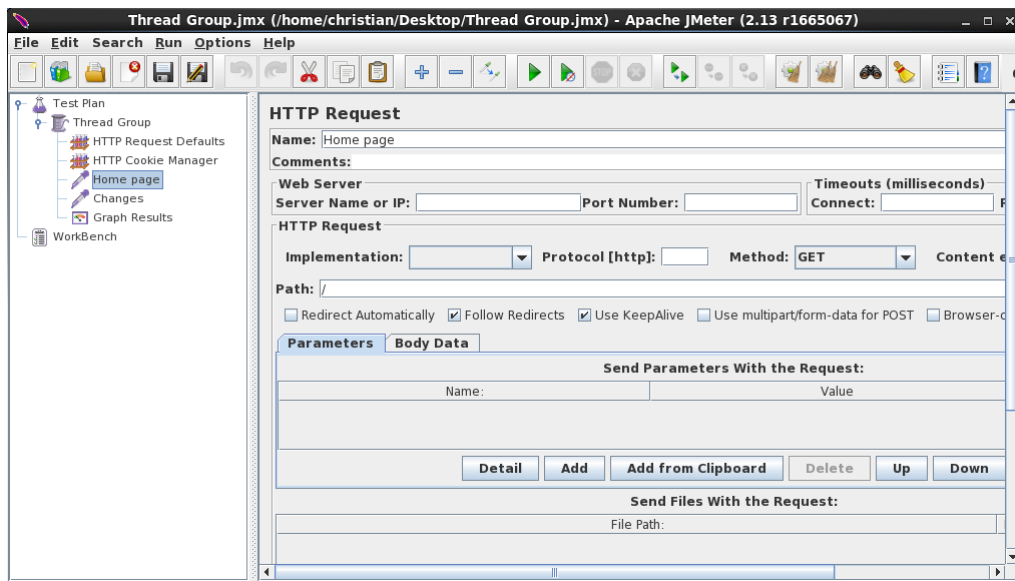


Figura 4.6: HTTP Request 1

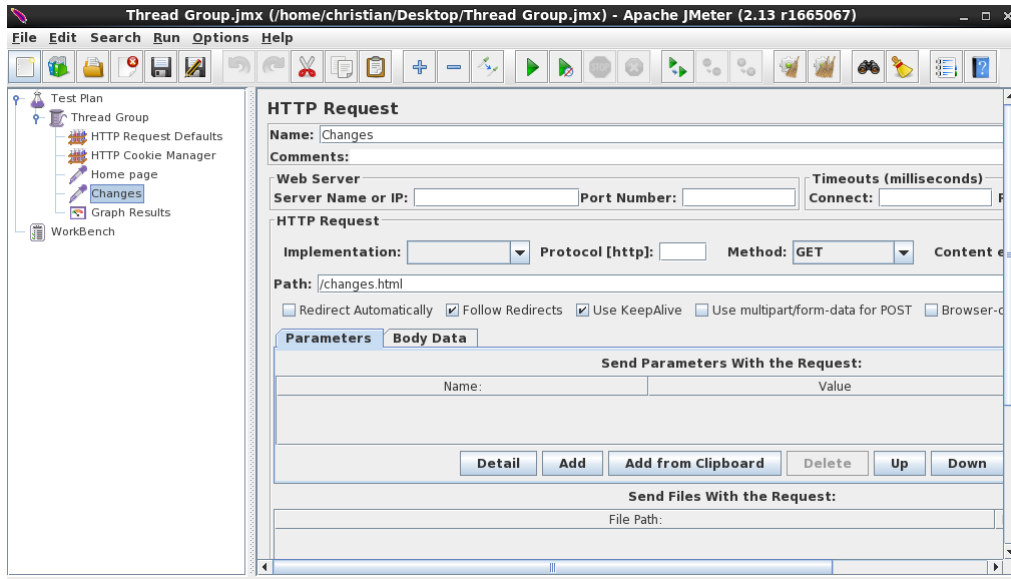


Figura 4.7: HTTP Request 2

Este es el resultado en modo gráfico:

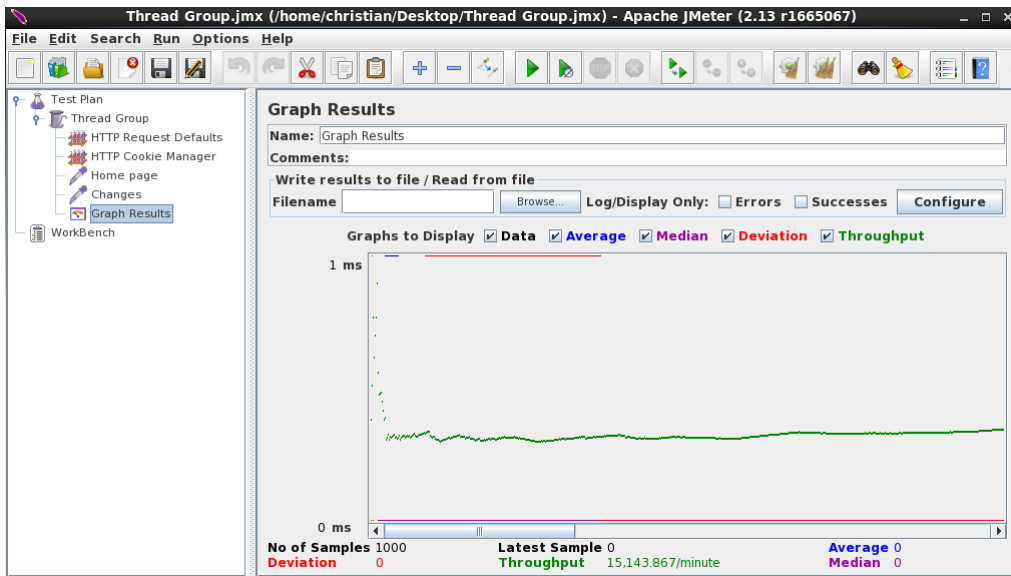


Figura 4.8: Graph Results

Las diferencias con respecto a la prueba hecha en el tutorial con jmeter.apache.org son evidentes, teniendo en cuenta en ese caso más parámetros como la conexión a internet del usuario. Nuestra prueba nos muestra una función constante mientras que la realizada en el tutorial varía en función de más factores.

5. Cuestión 5: 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

El objetivo de este test se sustenta en la creación de un benchmark clásico que mide el potencial de la CPU y memoria mediante operaciones de matrices y un algoritmo de ordenación. El desarrollo del benchmark se basa en código C++ y el uso de OpenMP, una interfaz de programación de aplicaciones (API) para la programación multiproceso de memoria compartida.

Para su implementación se han usado dos variables 'start' y 'end' para medir los tiempos de procesamiento de ambas operaciones, otras dos para almacenar los tiempos finales, uso de vectores y variables constantes.

El modo de uso es simple. Mediante terminal introducimos lo siguiente:

```
[christian@localhost Desktop]$ g++ benchmark.cpp
[christian@localhost Desktop]$ ./a.out
- Benchmark CPU: 5.66 segundos
- Benchmark MEMORIA: 4.17 segundos

TIEMPO FINAL: 9.83 segundos
```

Figura 5.1: Uso del Benchmark

El código implementado en el Benchmark es el siguiente:

```

#include <iostream>
#include <fstream>
#include <ctime>
#include <cstdlib>

using namespace std;

int main(){

    double tiempo1, tiempo2; // tiempos
    clock_t start,end; // variables de tiempo

    //////////////////////////////////// BENCHMARK CPU
    ////////////////////////////////////
    ////////////////////////////////////

    int A[1000][1000], B[1000][1000], C[1000][1000]; // matrices
    int k=1000, m=1000, n=1000; // tamaños

    cout<<"\n- Benchmark CPU: ";

    start = clock();

    ////////////////////////////////////
    #pragma omp parallel

    // Rellenamos las matrices.
    for(int i=0; i<k; ++i)
        for(int j=0; j<m; ++j)
        {
            A[i][j] = rand() % 1000;
        }

    for(int i=0; i<m; ++i)
        for(int j=0; j<n; ++j)
        {
            B[i][j] = rand() % 1000;
        }

    }

    // Inicializamos la matriz C.
    for(int i=0; i<k; ++i)
        for(int j=0; j<n; ++j)
            C[i][j] = 0;

    // Generamos la matriz C.
    for(int i=0; i<k; ++i)
        for(int j=0; j<n; ++j)
            for(int z=0; z<m; ++z)
                C[i][j] += A[i][z] * B[z][j];

    #pragma omp end parallel

    end = clock();

    tiempo1 = static_cast<double>(end-start)/CLOCKS_PER_SEC;
    cout<<"<<tiempo1<<" segundos"<<endl;

    //////////////////////////////////// BENCHMARK MEMORIA
    ////////////////////////////////////
    ////////////////////////////////////

    cout<<"- Benchmark MEMORIA: ";

    int min,aux;
    int *vector = new int[1000000000];
    int i, j;

    start = clock();

    ////////////////////////////////////
    #pragma omp parallel for

    for(i=0;i<1000000000;i++) //Rellenamos el vector

```

Figura 5.2: Código del Benchmark 1

```

vector[i]=1000000000-i;

for(i=0;i<1000000000;i++){ //Algoritmo de ordenacion por seleccion
    min=i;
    if (i != min){
        min=j;
    }

    aux=vector[i];
    vector[i]=vector[min];
    vector[min]=aux;
}

#pragma omp end parallel for
////////////////////////////////////

end = clock();

tiempo2 = static_cast<double>(end-start)/CLOCKS_PER_SEC;
cout<<"<<tiempo2<<" segundos"<<endl;
for(int i=0;i<80;i++) cout<<"_";

cout<<"\n\n          TIEMPO FINAL: "<<tiempo1 + tiempo2<<" segundos\n"<<endl;

system (0);

}

```

Figura 5.3: Código del Benchmark 2

A partir de varias ejecuciones, podemos conseguir un tiempo medio de ejecución en distintos sistemas operativos y comparar sus prestaciones frente a este Benchmark. En primer lugar la máquina virtual CentOS con 1 core y 1 GB RAM:

```

[christian@localhost Desktop]$ ./a.out
- Benchmark CPU: 8.39 segundos
- Benchmark MEMORIA: 7.05 segundos

TIEMPO FINAL: 15.44 segundos

[christian@localhost Desktop]$ ./a.out
- Benchmark CPU: 8.66 segundos
- Benchmark MEMORIA: 1.6 segundos

TIEMPO FINAL: 10.26 segundos

[christian@localhost Desktop]$ ./a.out
- Benchmark CPU: 9.06 segundos
- Benchmark MEMORIA: 1.68 segundos

TIEMPO FINAL: 10.74 segundos

```

Figura 5.4: Tiempos de ejecución del Benchmark en CentOS 1

Obteniendo una media de:

- Tiempo de CPU: 8.97 segundos
- Tiempo de Memoria: 3.44 segundos
- Tiempo total: 12.14 segundos

Luego volvemos a probar en la máquina virtual de CentOS con 3 GB de RAM y 4 cores:

```

[christian@localhost Desktop]$ ./a.out
- Benchmark CPU: 6.59 segundos
- Benchmark MEMORIA: 0.96 segundos

TIEMPO FINAL: 7.55 segundos

[christian@localhost Desktop]$ ./a.out
- Benchmark CPU: 5.95 segundos
- Benchmark MEMORIA: 0.81 segundos

TIEMPO FINAL: 6.76 segundos

[christian@localhost Desktop]$ ./a.out
- Benchmark CPU: 6.53 segundos
- Benchmark MEMORIA: 0.79 segundos

TIEMPO FINAL: 7.32 segundos

```

Figura 5.5: Tiempos de ejecución del Benchmark en CentOS 2

Obteniendo una media de:

- Tiempo de CPU: 6.356 segundos
- Tiempo de Memoria: 0.853 segundos
- Tiempo total: 7.21 segundos

Si realizamos una gráfica comparativa, tenemos:

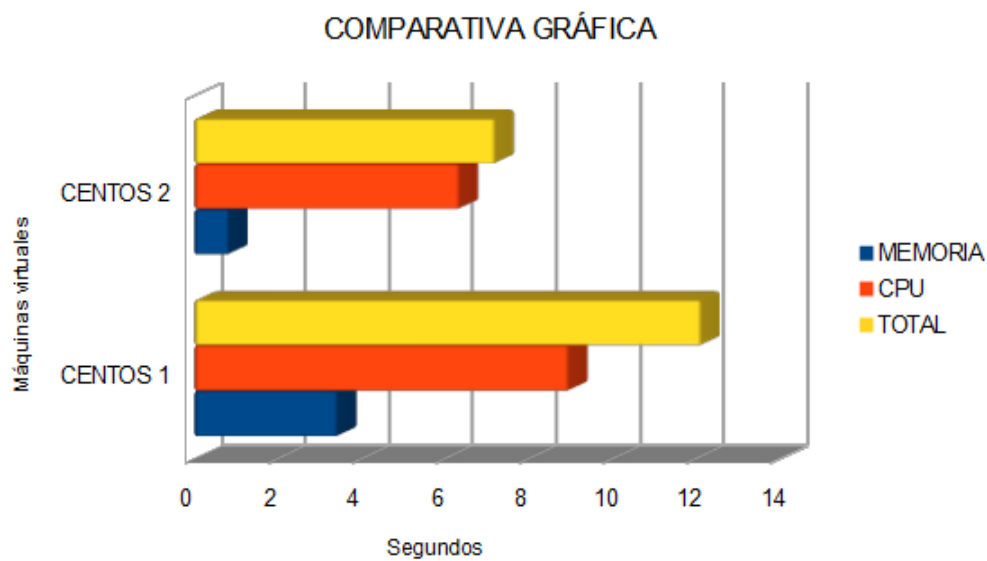


Figura 5.6: Comparativa en gráfica.

Podemos comprobar que a mayor número de recursos, el tiempo necesario para ejecutar el benchmark es inferior evidentemente, siendo el test de memoria el más beneficiado con una gran mejora.

Referencias

- [1] <http://linux.die.net/man/1/ab>.
- [2] <http://linux.die.net/man/1/phoronix-test-suite>.
- [3] <https://prasadlinuxblog.wordpress.com/2012/08/15/step-by-step-guide-installation-of-phoronix-benchmarking-tool/>.
- [4] <http://www.phoronix-test-suite.com/?k=downloads>.