Ingeniería de Servidores (2016-2017)
Grado en Ingeniería Informática
Universidad de Granada

Práctica 4. Benchmarks.

Manuel Jiménez Molina

23 de diciembre de 2016

Índice

Índice de figuras

Índice de tablas

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

Descargamos Phoronix de su página oficial[4].

Instalamos el paquete que descargado:

sudo dpkg -i phoronix-test-suite_6.8.0_all.deb

A continuación ejecutamos phoronix con el comando:

phoronix-test-suite

Nos muestra una lista de opciones a realizar, escogemos 'Run a test'. Veremos como nos muestra una lista de benchmarks a realizar:

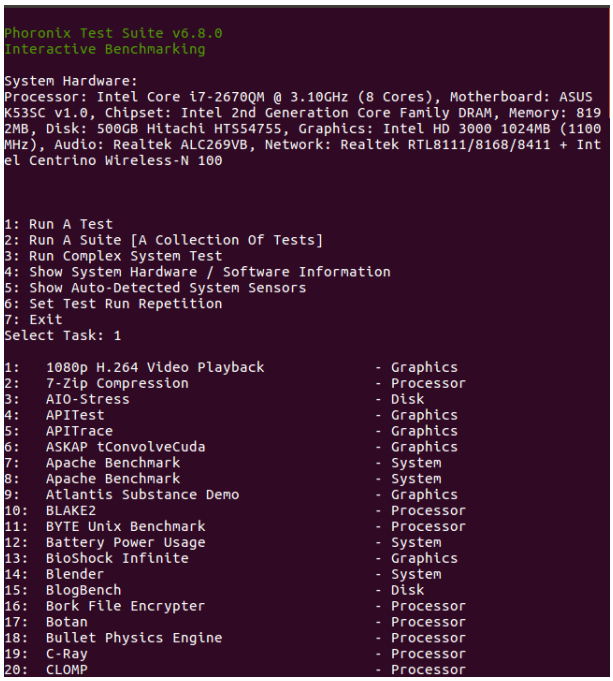

Figura 1.1: Hacer un test con Phoronix

Realizamos en benchmark número 65, llamado GpuTest. Vemos como se nos abre una pantalla en la que nos muestra con un entorno gráfico en 3D usando plot3d y OpenGL 3.0 nos muestra los fps (frames per seconds) y ms (delay o ping).Aquí vemos varias imágenes:

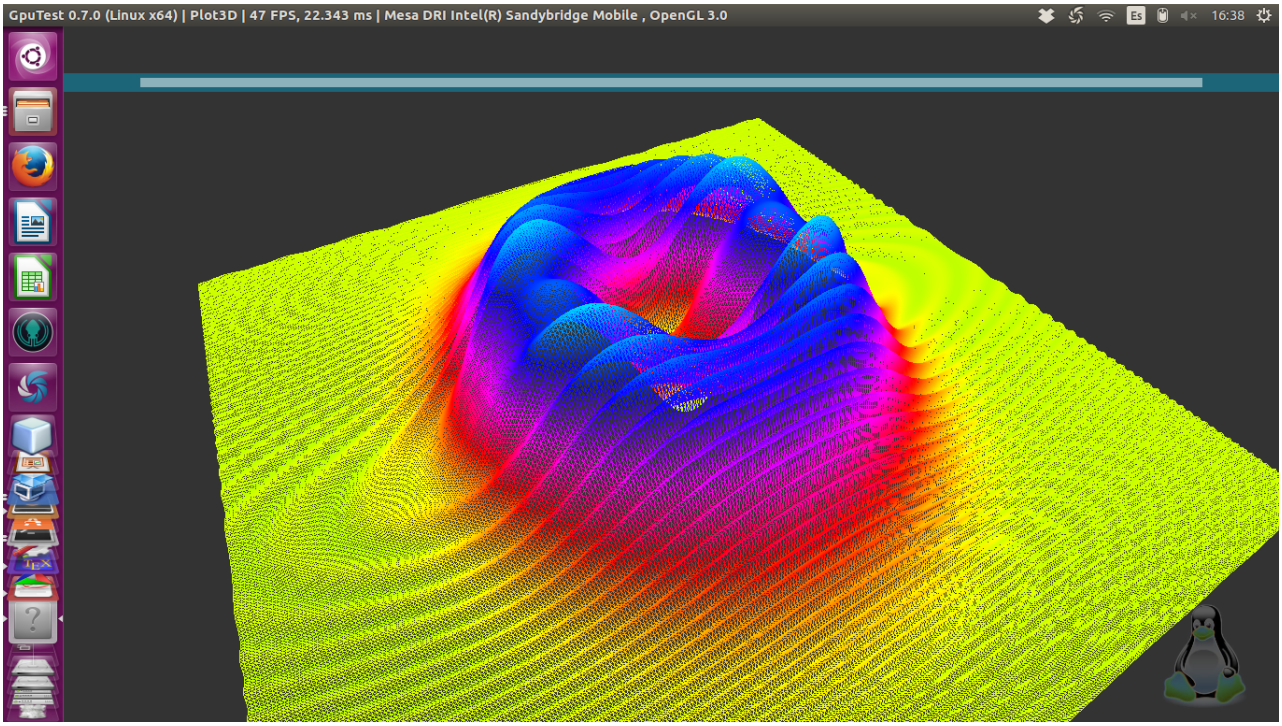

Figura 1.2: Imagen 1: test de gpu en Phoronix

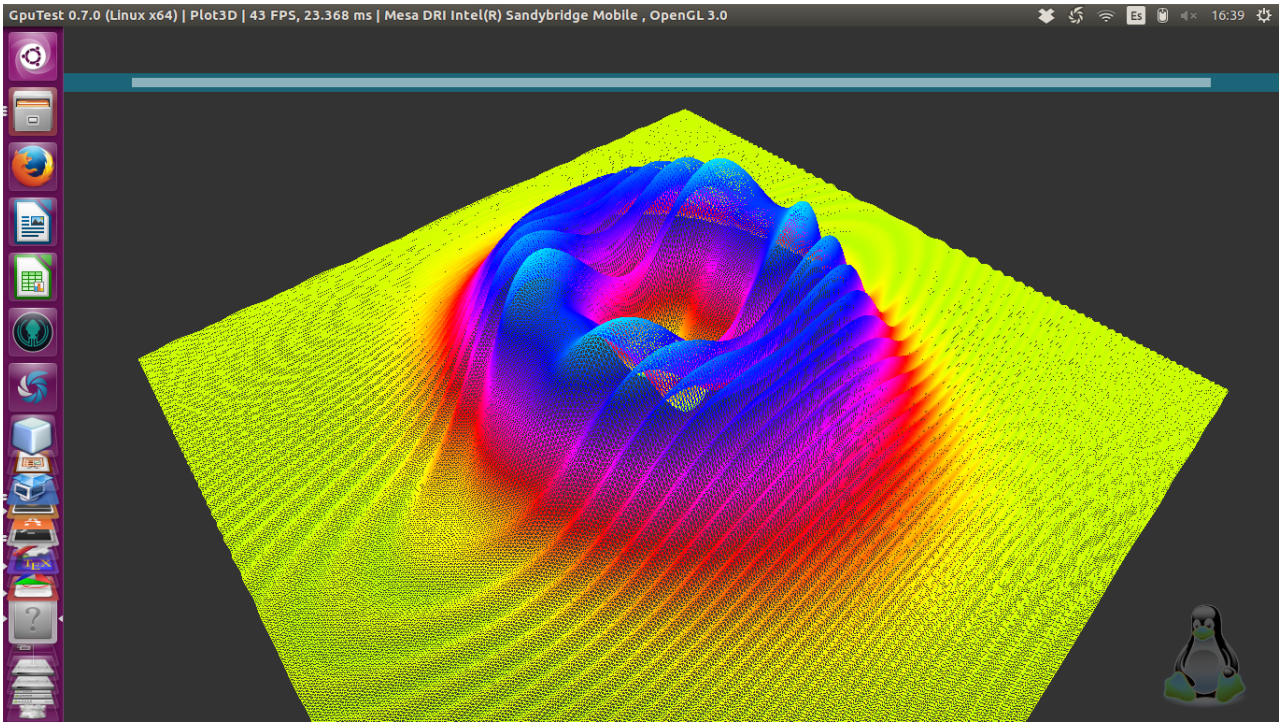

Figura 1.3: Imagen 2: test de gpu en Phoronix

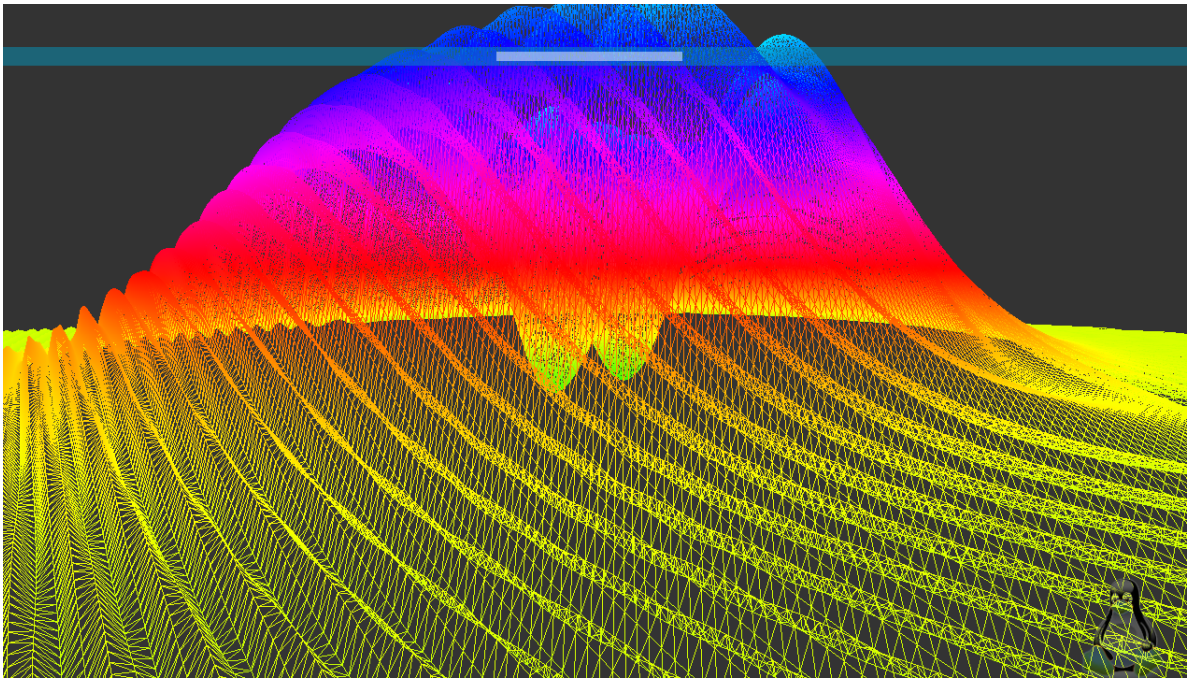

Figura 1.4: Imagen 3: test de gpu en Phoronix

Podemos ver como nuestra gpu tiene entre 40-60 fps y su delay o ping son entorno a 20-30 segundos. Este test consiste en realizar 3 pruebas en modo ventana y otros 3 en modo pantalla completa, mostrando durante un tiempo determinado una gráfica para cada uno de ellos.
Finalmente, nos muestra los resultados de los diferentes test, resaltando una media.

```
GpuTest 0.7.0:
    pts/gputest-1.3.1 [Test: Plot3D - Resolution: 1366 x 768 - Mode: Windowed]
    Test 1 of 2
    Estimated Trial Run Count:    3
    Estimated Test Run-Time:      4 Minutes
    Estimated Time To Completion: 8 Minutes (16:49 CET)
        Started Run 1 @ 16:42:27
        Started Run 2 @ 16:43:34
        Started Run 3 @ 16:44:40  [Std. Dev: 1.06%]

    Test Results:
        3099
        3053
        3116

    Average: 3089 Points


GpuTest 0.7.0:
    pts/gputest-1.3.1 [Test: Plot3D - Resolution: 1366 x 768 - Mode: Fullscreen]
    Test 2 of 2
    Estimated Trial Run Count:    3
    Estimated Time To Completion: 4 Minutes (16:49 CET)
        Started Run 1 @ 16:45:53
        Started Run 2 @ 16:46:59
        Started Run 3 @ 16:48:05  [Std. Dev: 0.03%]

    Test Results:
        3186
        3188
        3187

    Average: 3187 Points
```

Figura 1.5: Resultado test de gpu en Phoronix

Vemos como nos da una media para cada uno de los 3 test, con una puntuación de 3089 y 3187. Mirando las puntuaciones en esta página web[2] podemos ver por donde se encuentra nuestra gpu:

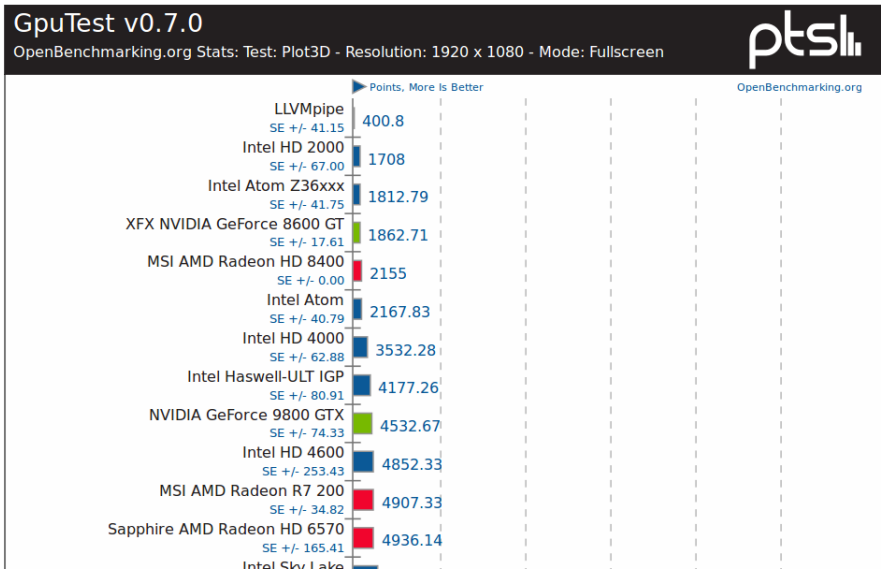

Figura 1.6: Comparativas de benchmarks de gpu

Usando el comando lspci[1] podemos ver la lista de dispositivos PCI en nuestro equipo. De este modo vemos que tenemos una gráfica de Intel de segunda generación.

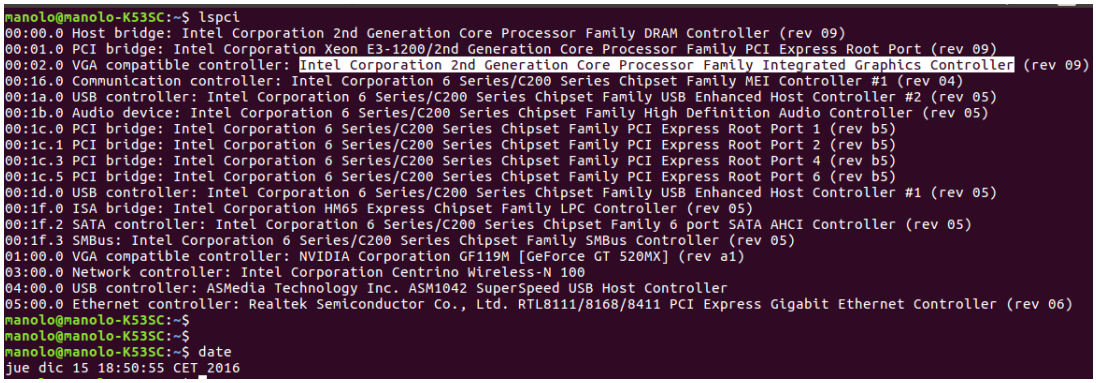

Figura 1.7: Obtener gpu de nuestra máquina

Nuestra gráfica corresponde a un Intel HD Graphics 3000 o 2000[3]:

Figura 1.8: Graficas 2ª generación de Intel

Viendo la comparativa de la imagen anterior y teniendo en cuenta la comparativa que nos ha hecho Phoronix dándonos una puntuación de entre 3089 y 3187, podemos decir que nuestra gráfica debe ser superior a Intel HD 2000 (puntuación de 1708) y menor que Intel HD 4000 (puntuación de 3532.28), por tanto seguramente nuestra gráfica sea una Intel HD 3000.

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) ¿cuántas "tareas" crea ab en el cliente?

Según el manual de apache benchmark (ab)[6] podemos saber que:

- -c 5: Opción para establecer el nivel de concurrencia, es decir, el número de solicitudes múltiples que se podrán realizar a la vez. Al ponerle 5 estaríamos realizando 5 peticiones a la vez.
- -n 100: Número de peticiones a realizar durante una sesión de benchmarking.

Vamos a realizar una monitorización para Apache que tenemos en Ubuntu Server con dirección 192.168.56.102 con concurrencia 5 y 100000 peticiones.A la vez que se ejecuta, monitorizamos los procesos que se crean en el cliente usando ps[7] y wc[8]:

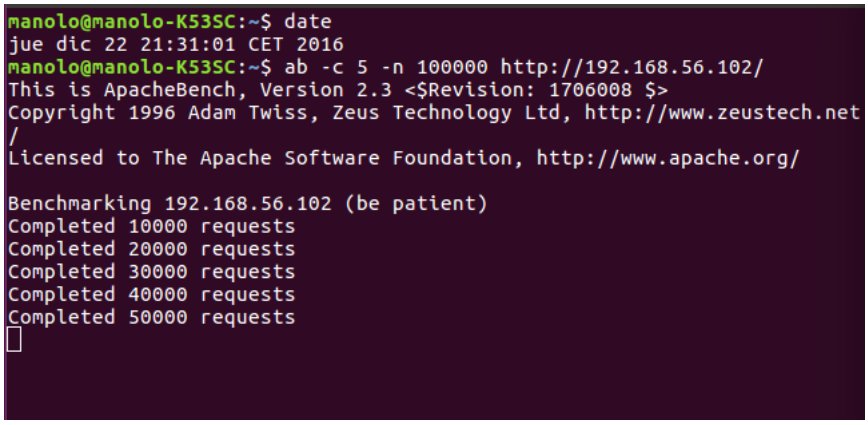

Figura 2.1: Ejecutando ab en Apache Server

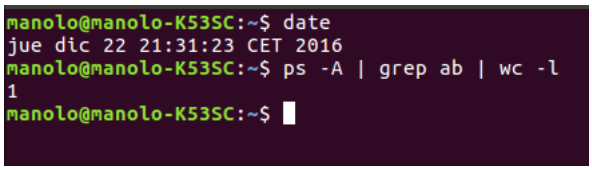

Figura 2.2: Monitorizando ab en Apache Server con ps

Vemos como solo se crea un proceso en el cliente a pesar de haber indicado a apache benchmark concurrencia de 5.
Aquí podemos ver como termina la ejecución de ab y observamos que indica que ha realizado la concurrencia que le indicamos.

```
manolo@manolo-K53SC:~$ date
jue dic 22 21:31:01 CET 2016
manolo@manolo-K53SC:~$ ab -c 5 -n 100000 http://192.168.56.102/
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.56.102 (be patient)
Completed 10000 requests
Completed 20000 requests
Completed 30000 requests
Completed 40000 requests
Completed 50000 requests
Completed 60000 requests
Completed 70000 requests
Completed 80000 requests
Completed 90000 requests
Completed 100000 requests
Finished 100000 requests


Server Software:        Apache/2.4.7
Server Hostname:        192.168.56.102
Server Port:            80

Document Path:          /
Document Length:        11510 bytes

Concurrency Level:      5
Time taken for tests:   36.316 seconds
Complete requests:      100000
Failed requests:        0
Total transferred:      1178300000 bytes
HTML transferred:       1151000000 bytes
Requests per second:    2753.58 [#/sec] (mean)
Time per request:       1.816 [ms] (mean)
Time per request:       0.363 [ms] (mean, across all concurrent requests)
Transfer rate:          31685.02 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.2      0      14
```

Figura 2.3: Final de ejecución 1 de ab en Apache Server

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.2      0      14
Processing:     0    2   5.1      1     419
Waiting:        0    1   0.4      1      22
Total:          0    2   5.1      1     419

Percentage of the requests served within a certain time (ms)
  50%      1
  66%      1
  75%      1
  80%      1
  90%      2
  95%      8
  98%     13
  99%     16
 100%    419 (longest request)
```

Figura 2.4: Fin de ejecución 2 de ab en Apache Server

Esto quiere decir que ab crea concurrencia sin necesidad de crear más hebras.

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).¿Cuál es la que proporciona mejores resultados? Muestre y coméntelos. (Use como máquina de referencia Ubuntu Server para la comparativa)

Vamos a ejecutar ab desde la máquina anfitriona con concurrencia 5 y 1000 peticiones a las demás máquinas. Los resultados obtenidos son los siguientes:

- **Ubuntu Server**:

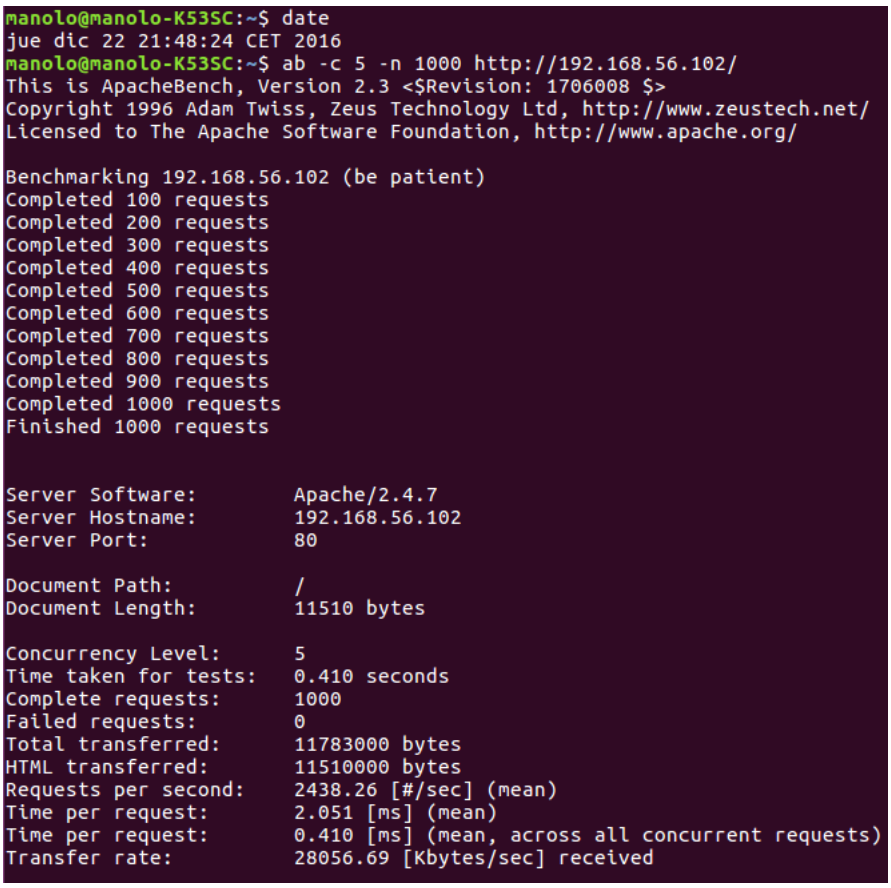

Figura 3.1: Imagen 1: Resultados de ab para Ubuntu Server

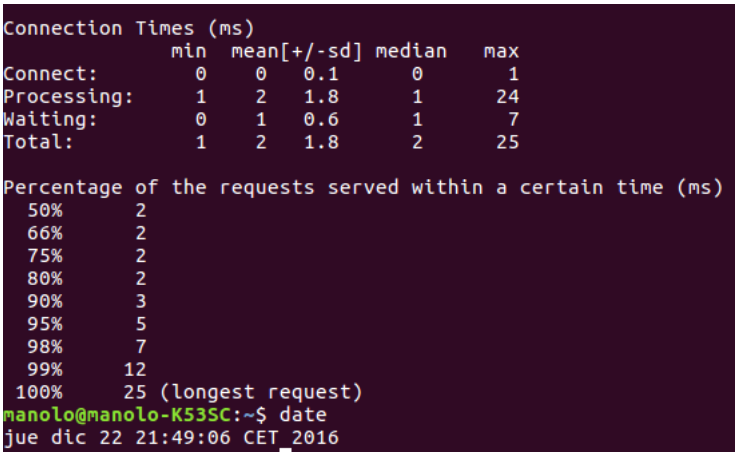

Figura 3.2: Imagen 2: Resultados de ab para Ubuntu Server

Para Ubuntu Server podemos ver como el tiempo total del test es de 0.41 segundos. El tiempo media para cada petición es de 2.051 milisegundos. Dado que tenemos concurrencia 5, el tiempo medio de petición en concurrencia es de 2.051 ms / 5 = 0.410 ms.
El total de bytes transferidos para Ubuntu Server es de 11783000 bytes.

- **Windows Server**:

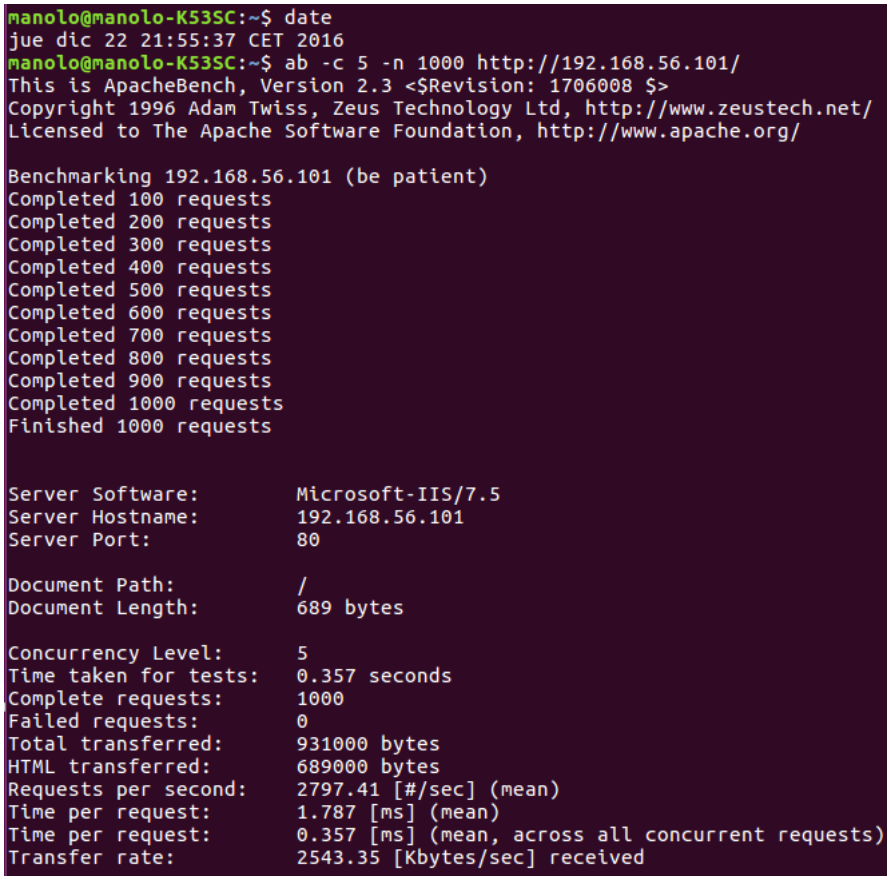

Figura 3.3: Imagen 1: Resultados de ab para Windows Server

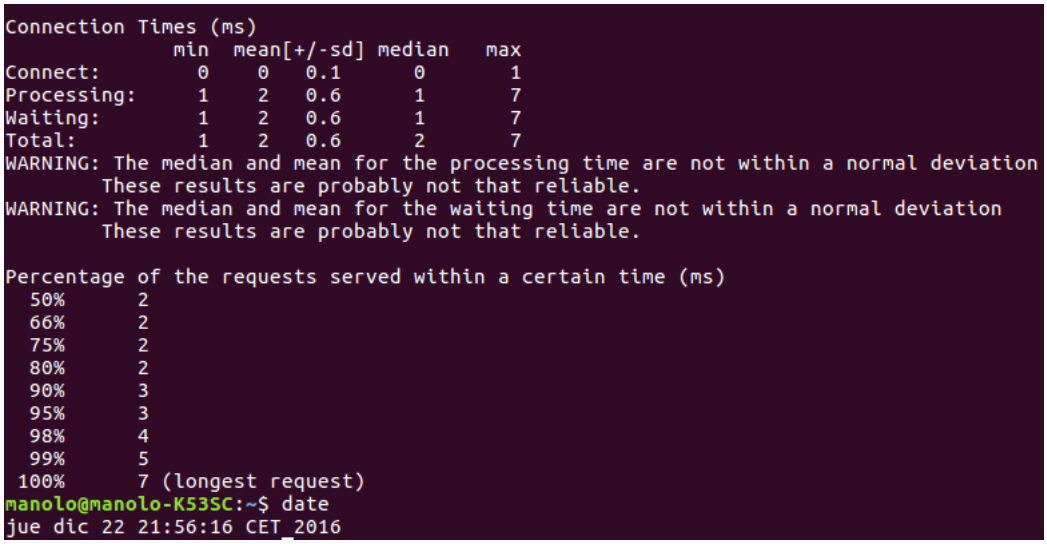

Figura 3.4: Imagen 2: Resultados de ab para Windows Server

Para Windows Server podemos ver como el tiempo total del test es de 0.357 segundos. El tiempo media para cada petición es de 1.707 milisegundos. Dado que tenemos concurrencia 5, el tiempo medio de petición en concurrencia es de 1.707 ms / 5 = 0.357 ms.
El total de bytes transferidos para Windows Server es de 931000 bytes.

- **CentOS**:

```
manolo@manolo-K53SC:~$ date
jue dic 22 22:18:33 CET 2016
manolo@manolo-K53SC:~$ ab -c 5 -n 1000 http://192.168.56.103/
This is ApacheBench, Version 2.3 <$Revision: 1706008 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 192.168.56.103 (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests


Server Software:        Apache/2.4.6
Server Hostname:        192.168.56.103
Server Port:            80

Document Path:          /
Document Length:        4897 bytes

Concurrency Level:      5
Time taken for tests:   0.631 seconds
Complete requests:      1000
Failed requests:        0
Non-2xx responses:      1000
Total transferred:      5179000 bytes
HTML transferred:       4897000 bytes
Requests per second:    1584.38 [#/sec] (mean)
Time per request:       3.156 [ms] (mean)
Time per request:       0.631 [ms] (mean, across all concurrent requests)
Transfer rate:          8013.20 [Kbytes/sec] received
```

Figura 3.5: Imagen 2: Resultados de ab para CentOS

```
Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.2      0       4
Processing:     1    3   5.4      2      92
Waiting:        1    2   1.1      2      14
Total:          2    3   5.4      3      92

Percentage of the requests served within a certain time (ms)
  50%      3
  66%      3
  75%      3
  80%      3
  90%      3
  95%      4
  98%      7
  99%     11
 100%     92 (longest request)
manolo@manolo-K53SC:~$ date
jue dic 22 22:18:56 CET 2016
```

Figura 3.6: Imagen 2: Resultados de ab para CentOS

Para CentOS podemos ver como el tiempo total del test es de 0.631 segundos. El tiempo media para cada petición es de 3.156 milisegundos. Dado que tenemos concurrencia 5, el tiempo medio de petición en concurrencia es de 3.156 ms / 5 = 0.631 ms.
El total de bytes transferidos para Windows Server es de 5179000 bytes.

Los diferentes resultados muestran a simple vista como unas máquinas realizan en test en menor tiempo que otras, pero influyen diferentes variables para que eso ocurra. Por ejemplo la cantidad de bytes que tiene cada servidor es diferente y ha de transmitirse. También influyen otros aspectos como la red e incluso el tiempo de llamadas al sistema realizadas para llevar a cabo la ejecución de Apache Benchmark. Vemos como por ejemplo Ubuntu Server tardó en atender el 100 % de las peticiones 25 ms, mientras que Windows Server tardó 7 ms en atender el 100 % y mucho más CentOS, con 92 ms para atender el 100 % (de 99 % a 100 % CentOS tardó 92ms-11ms=81ms, una brutalidad).

A primera vista podemos ver como Windows Server ha tardado 0.357 segundos en realizar el test, seguido de cerca por Ubuntu Server con 0.41 segundos y finalmente CentOS con 0.631 segundos.
Mirando un poco más, vemos que la cantidad de bytes transferidos varían para cada uno. Tenemos a Ubuntu Server con 11783000 bytes, Windows Server con 931000 bytes y CentOS con 5179000 bytes. Eso se debe a que cada servidor tiene una página web predeterminada distinta y por tanto los bytes varían.
Comparando la productividad de cada uno (Kbytes recibidos por segundo):

- Ubuntu Server: 28056.69 Kbytes / s
- Windows Server: 2543.35 Kbytes / s
- CentOS: 8013.2 bytes / ms

Según la productividad, el mejor es Ubuntu Server, seguido de CentOS y finalmente Windows Server .

Hemos visto como la productividad de Ubuntu Server es la mejor, sin embargo Windows Server realiza el test en un tiempo menor. Esto se debe a que la cantidad de bytes que recibe Ubuntu Server es mucho mayor. Basándonos en estas características la mejor opción podría ser Ubuntu Server con Apache.

Como conclusión final, es necesario decir que existen numerosas características a analizar de un servidor (muchas más de las analizadas) para determinar y concluir que sistema sería el mejor o adecuado para el uso que queramos darle.

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 sus máquinas virtuales ¿coincide con los resultados de ab?

Vamos a ir por pasos para usar Jmeter, siguiendo como guía su página oficial[5]:

- Paso 1: Instalación de Jmeter por el gestor de paquetes:

```
manolo@manolo-K53SC:~$ sudo apt-get install jmeter
[sudo] password for manolo:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
```

Figura 4.1: Instalación de Jmeter

- Paso 2: Ejecutamos el programa poniendo su nombre en el terminal y se nos abrirá:

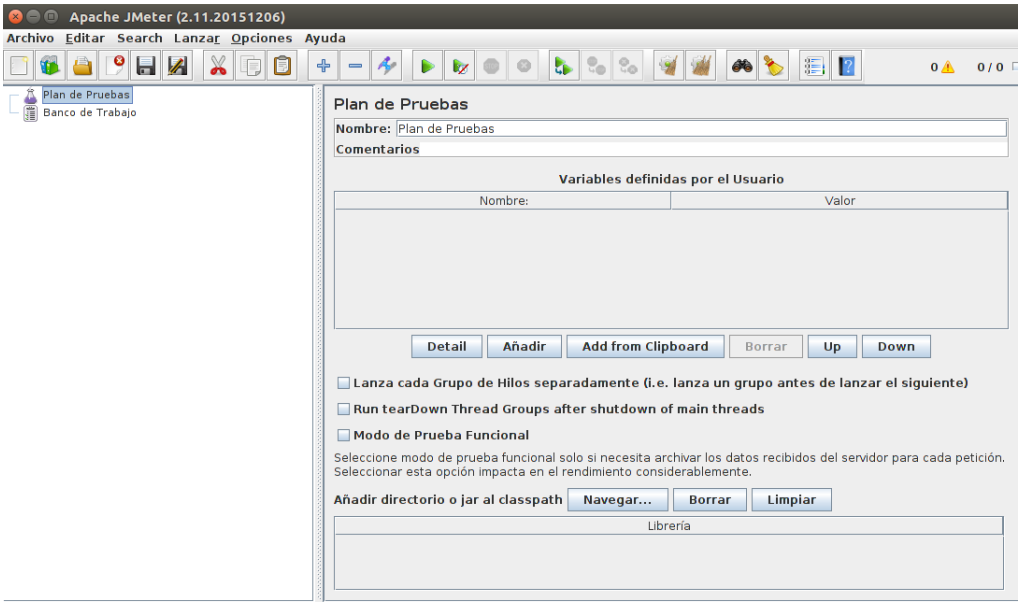

Figura 4.2: Abrir Jmeter

- Paso 3: Añadimos un Thread Group, que sirve para decirle a Jmeter el número de usuarios a simular, la frecuencia con la que los usuarios envían peticiones y cuantas peticiones deberían enviar. Para añadir, nos vamos al menú, pulsamos en .Editar",

seleccionamos .Añadir", escogemos "Hilos (Usuarios)z finalmente "Grupo de hilos". Nos deberá aparecer lo siguiente:

Figura 4.3: Crear Grupo de hilos en Jmeter

- Paso 4: Configuramos el grupo de hilos estableciendo "números de hilos.ª 5 (serían los usuarios) y el número de veces que se va a repetir la prueba poniendo 100 en Çontador del bucle".

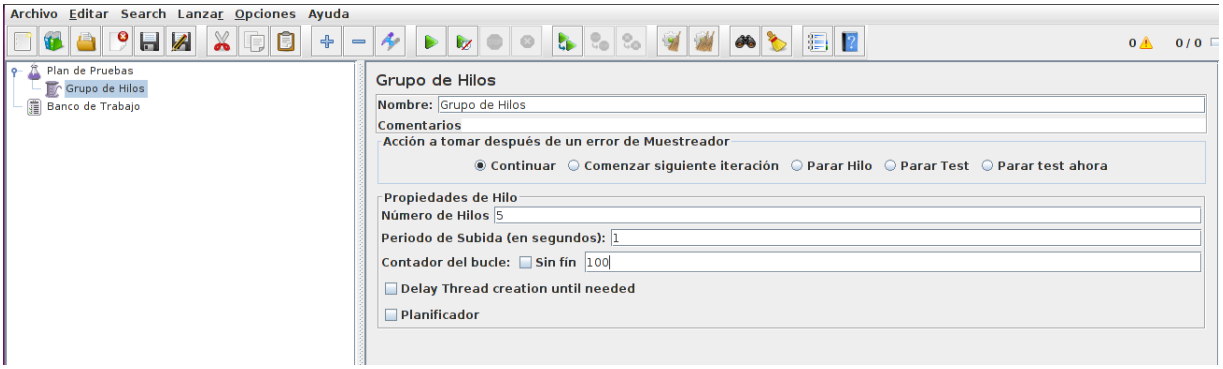

Figura 4.4: Configurar Grupo de hilos en Jmeter

- Paso 5: Añadir las propiedades por defecto para peticiones HTTP. Una vez que tenemos los usuarios, se definen las tareas que realizarán. Vamos a configurar las peticiones HTTP por defecto.
  Para añadir peticiones HTTP por defecto hacemos click derecho sobre "Grupo de hilosz hacemos .Añadir» .Elemento de configuración» "Valores por defecto para petición HTTP". Nos deberá aparecer lo siguiente:

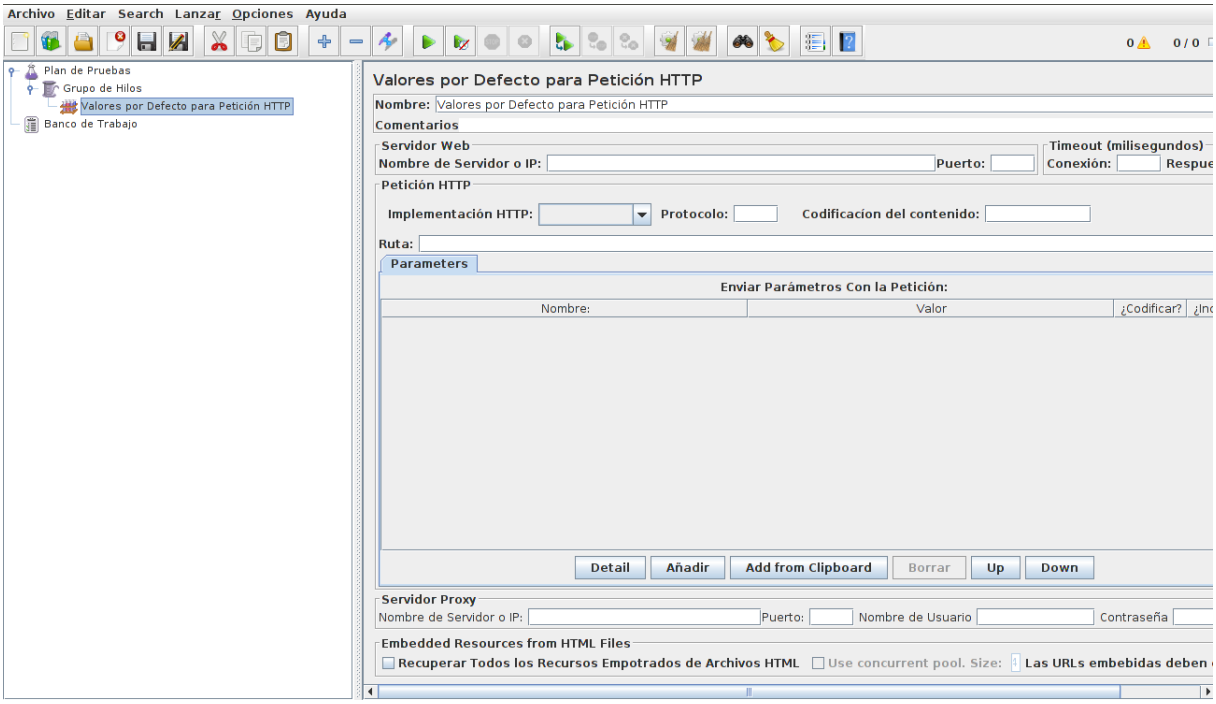

Figura 4.5: Añadir valores por defecto para petición HTTP al Grupo de hilos

- Paso 6: Establecemos el nombre del servidor o IP (será la de las máquinas virtuales) en el campo "Nombre del Servidor o IP"dentro de los Valores por Defecto para Petición HTTP. Se modificará más adelante conforme se prueben las diferentes máquinas.
- Paso 7: Añadir soporte a cookies. Vamos a activarlas, para ello click derecho sobre "Grupo de hilosz hacemos .Añadir» .Elemento de configuración» "Gestor de cookies HTTP". Nos deberá quedar así:

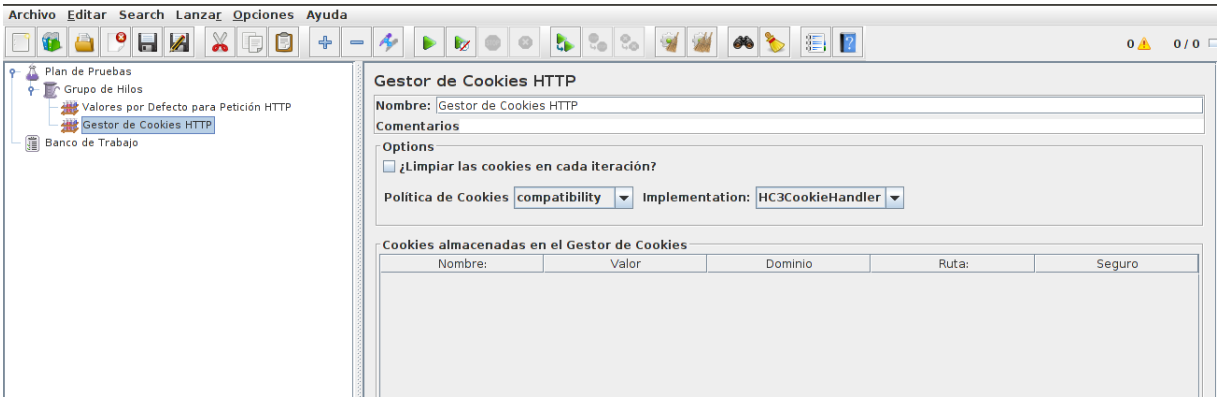

Figura 4.6: Añadir gestor de cookies HTTP al Grupo de hilos

- Paso 8: Añadir petición HTTP. Para ello click derecho sobre "Grupo de hilosz .ªñadir» "Muestreador» "Petición HTTP". Nos deberá quedar de esta forma:

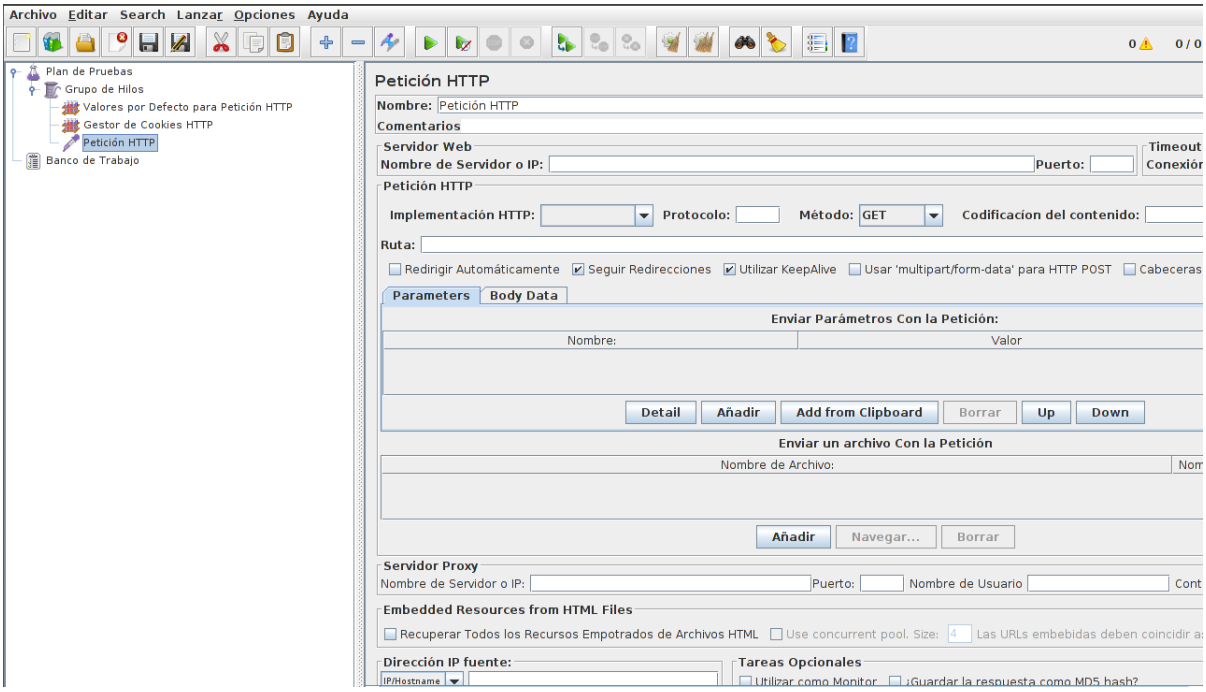

Figura 4.7: Añadir petición HTTP al Grupo de hilos

- Paso 9: Añadir un oyente para que almacene los datos de la prueba. Se encarga de almacenar los resultados de las peticiones HTTP en un archivo y los presenta

de forma gráfica. Para añadirlo, click derecho sobre "Grupo de hilosz .ªñadir» - eceptor» "Gráfico de Resultados". Veremos lo siguiente:

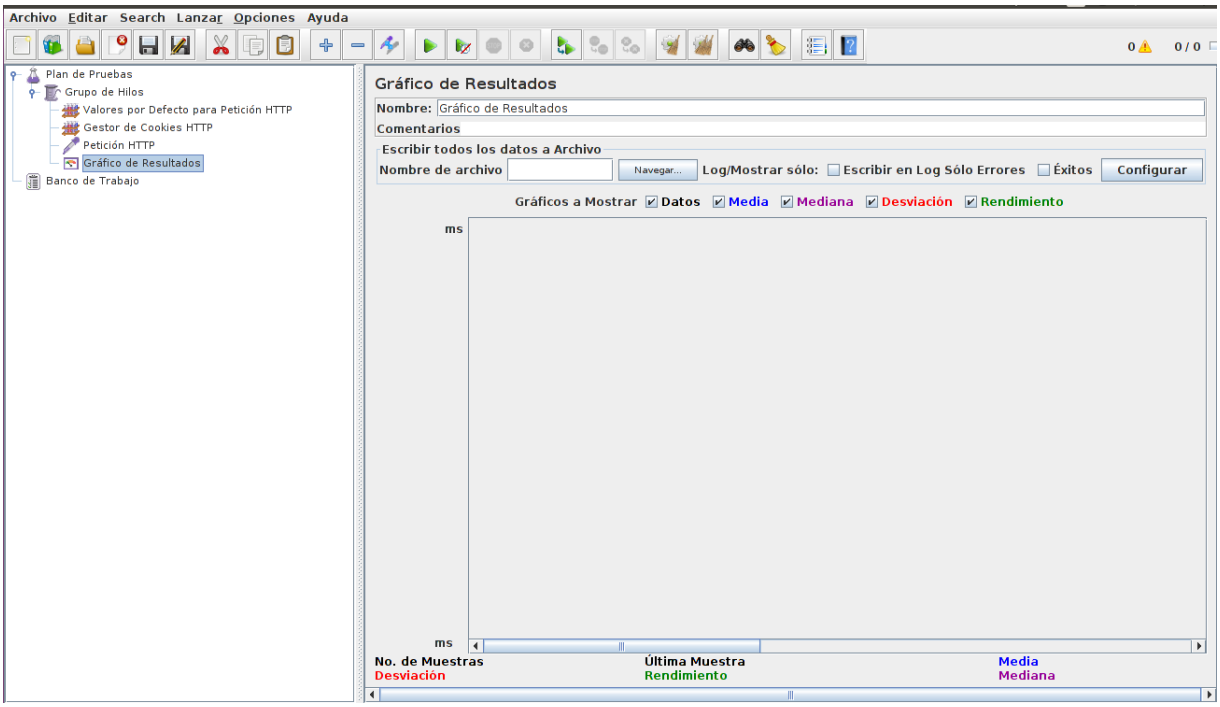

Figura 4.8: Añadir gráfico de resultados al Grupo de hilos

Probamos con phyadmin desde la máquina de Ubuntu Server, para ello tenemos que hemos configurado lo siguiente:

- Ponemos a 1000 el bucle y con 5 usuarios (esto serán 5000 muestras).

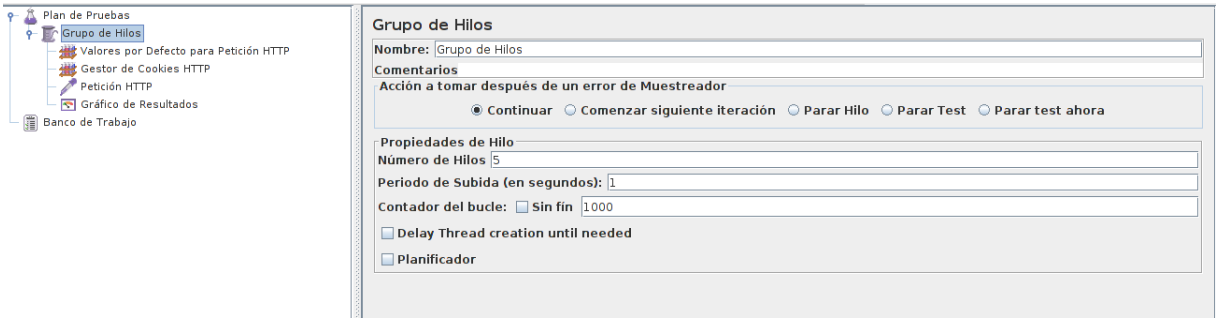

Figura 4.9: Jmeter con 5000 muestras

- Ponemos la url de phpmyadmin en "Valores por defecto para petición HTTP":

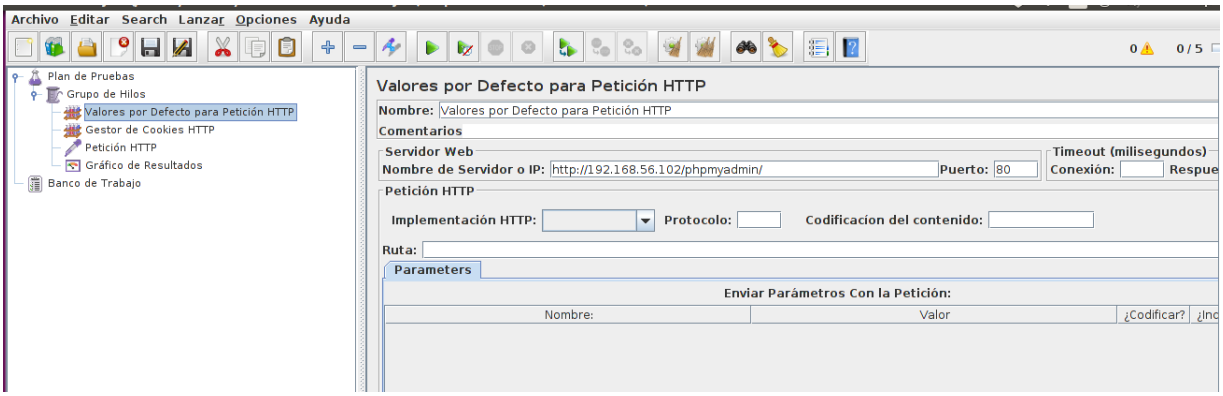

Figura 4.10: Configurar valores por defecto para peticion HTTP en Jmeter para phpmyadmin

- Configuramos la petición HTTP y establecemos valores que se pasan por ella, que serán el usuario y contraseña para phpymyadmin:

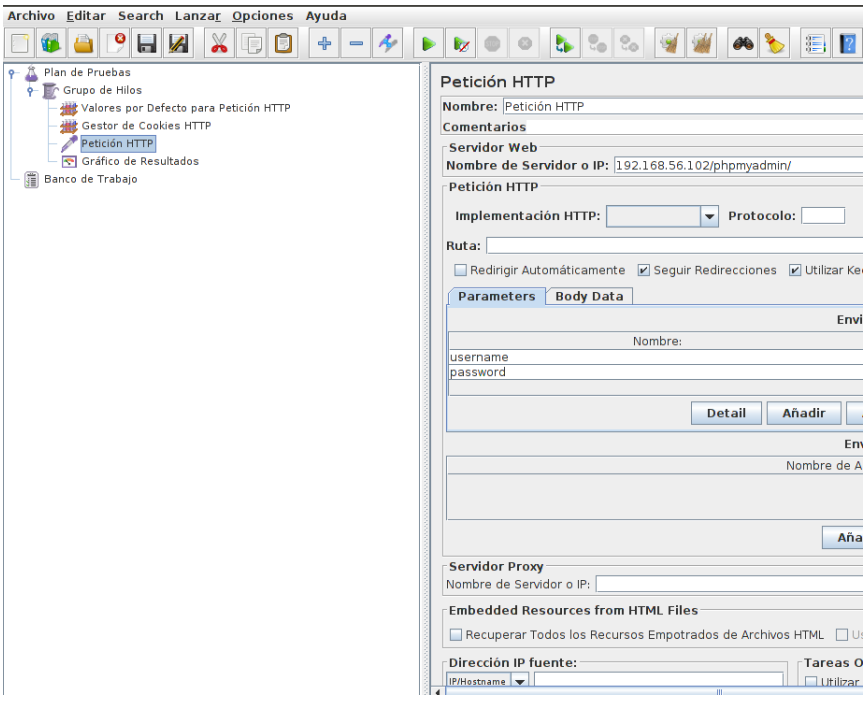

Figura 4.11: Configurar la petición HTTP en Jmeter para phpmyadmin

- Lanzamos el test y comprobamos el gráfico:

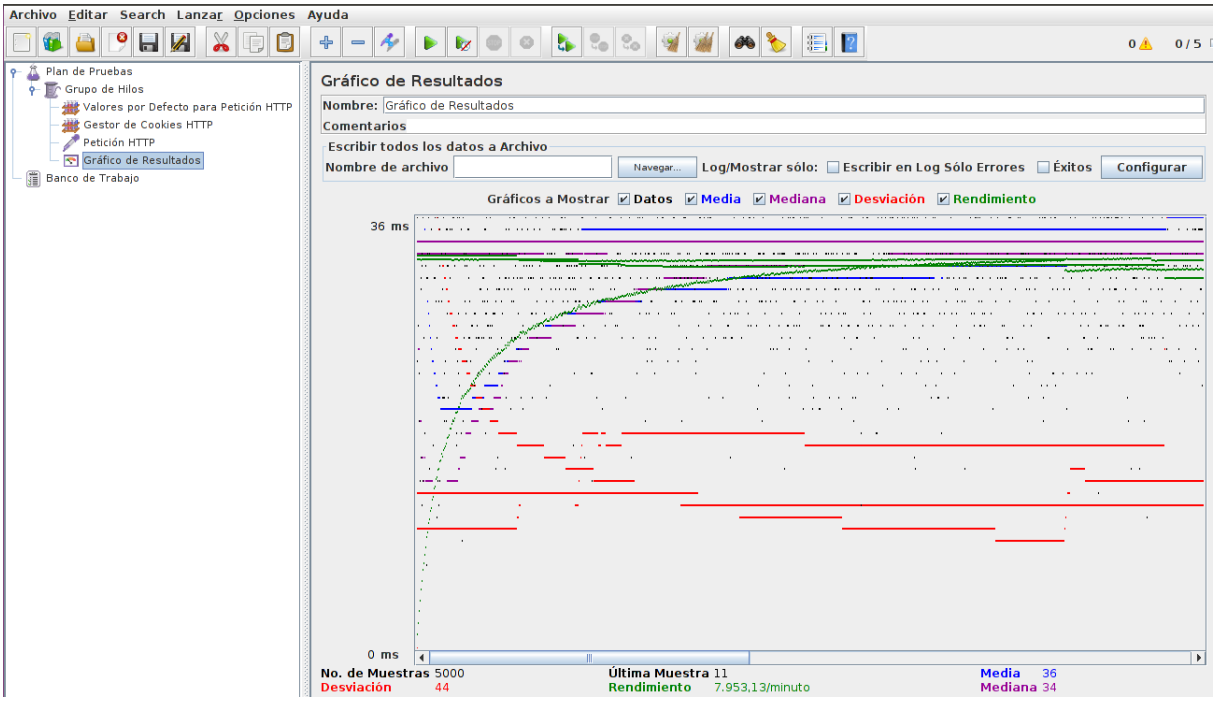

Figura 4.12: Gráfico generado tras en test en Jmeter para phpmyadmin

- Lanzamos con ab el mismo test, con 5000 muestras y 5 hebras.

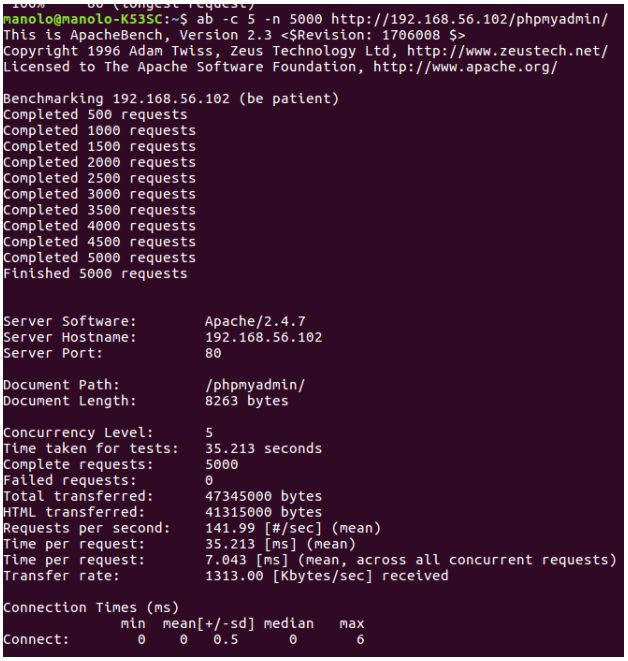

Figura 4.13: Imagen 1:Mismo test con ab que en Jmeter para phpmyadmin

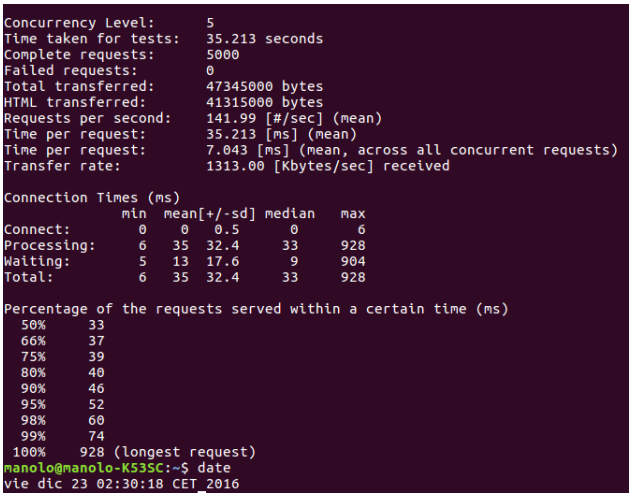

Figura 4.14: Imagen 2:Mismo test con ab que en Jmeter para phpmyadmin

En la gráfica nos sale media 36, mediana 34 y desviación 44 para 5000 muestras. Con ab sale una media de 35 y una mediana de 33 con una desviación de 32.4. Los resultados son muy parecidos por lo que se puede decir que casi prácticamente coinciden.

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.

5.1. 1) Objetivo del benchmark

El objetivo del benchmark consiste en mandar información por un socket para ver cual es la velocidad máxima de transferencia utilizando para ello diferentes tamaños de paquetes. Un socket es la transferencia básica de información en Internet, es un punto final de un enlace de comunicación bidireccional entre dos programas que se ejecutan en la red. Está enlazado a un número de puerto para que la capa TCP pueda identificar la aplicación a la que se destinan los datos[18, 14, 21, 20, 12, 17, 10, 13, 9, 19, 16, 15, 11].

5.2. 2) Métricas (unidades, variables, puntuaciones, etc.).

Para la velocidad son MB/s y tamaño de los paquetes en bytes.

5.3. 3) Instrucciones para su uso.

Para su uso basta con compilarlo con la orden gcc -o bench bench.c y posteriormente ejecutarlo con ./benchmark.

5.4. 4) Ejemplo de uso analizando los resultados.

Se crean un proceso padre y otro hijo los cuales crean una vía de comunicación para enviarse y recibir información usando para ello el localhost (127.0.0.1). Para diferentes tamaños de paquete va midiendo la velocidad de transferencia y muestra la velocidad más óptima y el tamaño de paquete que consigue dicha velocidad.

El código usado para el benchmark es el siguiente:

```c
#define _GNU_SOURCE
#define _CICLOS
#pragma GCC optimize("O0")
#include <sys/types.h>
#include <arpa/inet.h>
#include <sys/mman.h>
#include <sys/time.h>
#include <inttypes.h>
#include <stdlib.h>
#include <malloc.h>
#include <unistd.h>
#include <string.h>
#include <stdio.h>
#include <error.h>
#include <sched.h>
#include <netdb.h>
#include <time.h>

#define t_buf 10000000
char buf[t_buf];

int main()
{
    int port=7658;
    int datasocket;
    double serv, antser;
    int i;
    int *size = mmap(NULL,sizeof(int),PROT_READ|PROT_WRITE,MAP_SHARED|MAP_ANONYMOUS,-1,0);
    int *barrier = mmap(NULL,sizeof(int),PROT_READ|PROT_WRITE,MAP_SHARED|MAP_ANONYMOUS,-1,0);
    *buf=1;
    *size = 64;
    *barrier = 1;

    if(!fork())
    {
        struct sockaddr_in sock;

        datasocket = socket(AF_INET,SOCK_STREAM,0);
        sock.sin_family = AF_INET;
        sock.sin_addr.s_addr = inet_addr("127.0.0.1");
        sock.sin_port = htons(port);

        connect(datasocket,(struct sockaddr*)&sock,sizeof(sock));
```

Figura 5.1: Imagen 1: Código del benchmark

```c
    if(!fork())
    {
        struct sockaddr_in sock;

        datasocket = socket(AF_INET,SOCK_STREAM,0);
        sock.sin_family = AF_INET;
        sock.sin_addr.s_addr = inet_addr("127.0.0.1");
        sock.sin_port = htons(port);

        connect(datasocket,(struct sockaddr*)&sock,sizeof(sock));

        do{
            send(datasocket,buf,*size,0);
            while(*barrier);
            *barrier = 1;
        }while(*size);
        close(datasocket);
    }
    else
    {
        int cont_sock, from_len, so_reuseaddr = 1;
        struct sockaddr_in sock;

        cont_sock = socket(AF_INET,SOCK_STREAM,0);

        sock.sin_family = AF_INET;
        sock.sin_addr.s_addr = INADDR_ANY;
        sock.sin_port = htons(port);

        setsockopt(cont_sock,SOL_SOCKET,SO_REUSEADDR,&so_reuseaddr,sizeof(int));
        bind(cont_sock,(struct sockaddr*)&sock,sizeof(sock));
        listen(cont_sock,1);
        datasocket = accept(cont_sock,0,0);
```

Figura 5.2: Imagen 2:Código del benchmark

```
    else
    {
        int cont_sock, from_len, so_reuseaddr = 1;
        struct sockaddr_in sock;

        cont_sock = socket(AF_INET,SOCK_STREAM,0);

        sock.sin_family = AF_INET;
        sock.sin_addr.s_addr = INADDR_ANY;
        sock.sin_port = htons(port);

        setsockopt(cont_sock,SOL_SOCKET,SO_REUSEADDR,&so_reuseaddr,sizeof(int));
        bind(cont_sock,(struct sockaddr*)&sock,sizeof(sock));
        listen(cont_sock,1);
        datasocket = accept(cont_sock,0,0);

        antser=0;
        while(*size)
        {
            serv=0;
            struct timespec t1, t2;
            clock_gettime(CLOCK_REALTIME,&t1);
            recv(datasocket,buf,*size,0);
            clock_gettime(CLOCK_REALTIME,&t2);
            serv += (double)((t2.tv_nsec-t1.tv_nsec)*1e-9+t2.tv_sec-t1.tv_sec);
            serv = *size / (serv*1024*1024);
            *size *= 2;
            if(serv<antser)
            {
                *size /= 4;
                serv = antser;
                printf("Velocidad máxima socket = %.3f MB/s\n", serv);
                *size=0;
            }
            antser = serv;
            *barrier = 0;
        }
        close(datasocket);
        munmap(size, sizeof(int));
        munmap(barrier, sizeof(int));
    }

    return 0;
}
```

Figura 5.3: Imagen 3:Código del benchmark

Aquí vemos un ejemplo de ejecución:

```
manolo@manolo-K53SC:~/Escritorio/ISE(segundo año)/practica 4$ ./bench
Velocidad máxima socket = 5.818 MB/s
Paquete de tamaño 64 bytes
manolo@manolo-K53SC:~/Escritorio/ISE(segundo año)/practica 4$
```

Figura 5.4: Ejecución del benchmark

Podemos ver como para un socket el tamaño adecuado para la máxima velocidad es 64 bytes, no más ni menos ya que para tamaños superiores e inferiores no irá a la velocidad indicada por este, la cual es de 5.8188 MB/s.

6. Opcional 1: ¿Qué es Scala? Instale Gatling y pruebe los escenarios por defecto.

Referencias

[1] `https://linux.die.net/man/8/lspci`, consultado el 15 de Diciembre de 2016.

[2] `https://openbenchmarking.org/showdown/pts/gputest`, consultado el 15 de Diciembre de 2016.

[3] `http://www.intel.com/content/www/us/en/support/graphics-drivers.html`, consultado el 15 de Diciembre de 2016.

[4] http://www.phoronix-test-suite.com/?k=downloads, consultado el 15 de Diciembre de 2016.

[5] http://jmeter.apache.org/usermanual/build-web-test-plan.html, consultado el 22 de Diciembre de 2016.

[6] https://linux.die.net/man/1/ab, consultado el 22 de Diciembre de 2016.

[7] https://linux.die.net/man/1/ps, consultado el 22 de Diciembre de 2016.

[8] https://linux.die.net/man/1/wc, consultado el 22 de Diciembre de 2016.

[9] https://linux.die.net/man/2/accept, consultado el 22 de Diciembre de 2016.

[10] https://linux.die.net/man/2/bind, consultado el 22 de Diciembre de 2016.

[11] https://linux.die.net/man/2/close, consultado el 22 de Diciembre de 2016.

[12] https://linux.die.net/man/2/connect, consultado el 22 de Diciembre de 2016.

[13] https://linux.die.net/man/2/listen, consultado el 22 de Diciembre de 2016.

[14] https://linux.die.net/man/2/mmap, consultado el 22 de Diciembre de 2016.

[15] https://linux.die.net/man/2/munmap, consultado el 22 de Diciembre de 2016.

[16] https://linux.die.net/man/2/recv, consultado el 22 de Diciembre de 2016.

[17] https://linux.die.net/man/2/setsockopt, consultado el 22 de Diciembre de 2016.

[18] https://linux.die.net/man/2/socket, consultado el 22 de Diciembre de 2016.

[19] https://linux.die.net/man/3/clock_gettime, consultado el 22 de Diciembre de 2016.

[20] https://linux.die.net/man/3/htons, consultado el 22 de Diciembre de 2016.

[21] https://linux.die.net/man/3/inet_addr, consultado el 22 de Diciembre de 2016.