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

Memoria Práctica 4

Jesús Ángel González Novez

16 de diciembre de 2014

Índice

Índice de figuras

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

Para instalar Phoronix Suite:

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

Para listar los tests[1] disponibles:

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

Para listar las suites[2] de tests disponibles:

```
phoronix-test-suite list-available-suites
```

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

El parámetro -c lo que nos indica es la concurrencia de peticiones, es decir, simulamos en este caso(-c 30) que nos hagan 30 peticiones simultáneas a nuestro servidor, por defecto sería una, para más información podemos consultar el manual[3]
El parámetro -n lo que nos indica el número total de peticiones que haremos durante el test, por defecto es una sola, lo cual no es representativo en un benchmark por lo que se usan valores mayores como en este caso(-n 1000), para mas información podemos consultar también el manual[4] Algunos ejemplos de ejecución serían los siguientes:

[1] man phoronix-test-suite, línea 225
[2] man phoronix-test-suite, línea 221
[3] man ab, línea 41
[4] man ab, línea 76

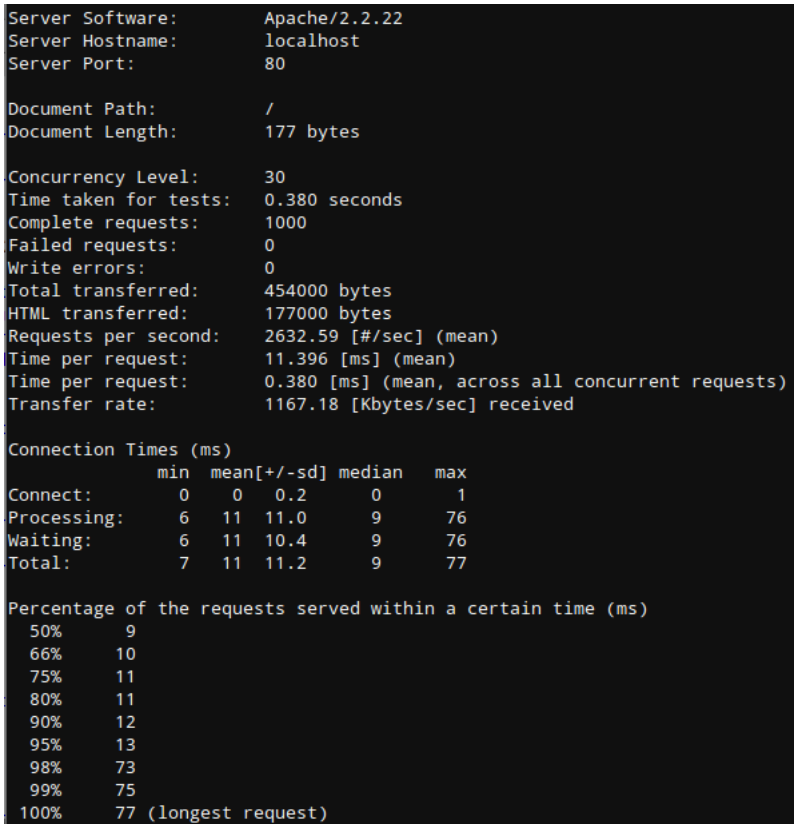

Figura 2.1: ab -n 1000 -c 30 http://localhost/

Nota: cuando hacemos benchmark a sitios externos afectan otros factores, el primero es que puede ni tener Apache como servidor, después influyen el ancho de banda tanto nuestro como de ellos, la seguridad que tengan para evitar filtrar datos, el rechazo a peticiones continuas de un solo origen(puede ser interpretado como ataque de stress), etc ...

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?

El test lo haré en Windows Server y en Ubuntu, omito el de CentOS porque la diferencia significativa estará entre Linux-Windows, si bien los puristas me dirán que CentOS superará en algunas décimas a Ubuntu...

En ambos casos el comando exacto utilizado(que ya he explicado en el ejercicio anterior) es el siguiente:

```
ab -n 1000 -c 30 ip
```

A continuación muestro las capturas de Windows y de Ubuntu:

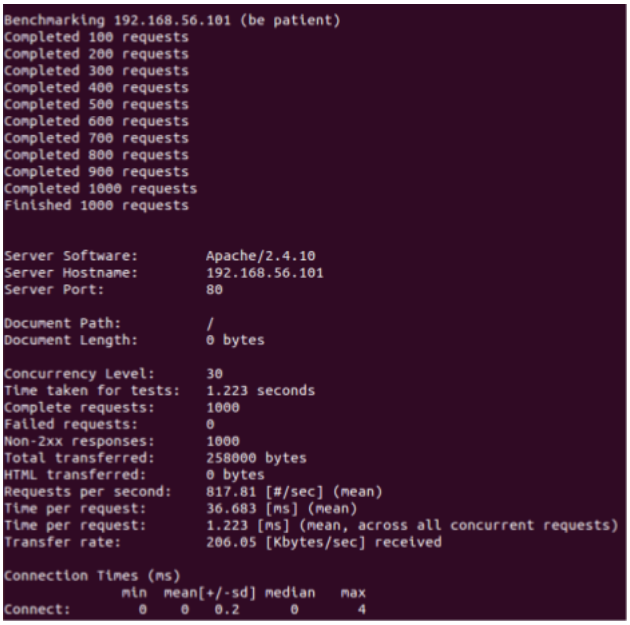

Figura 3.1: Comando ab contra Windows Server

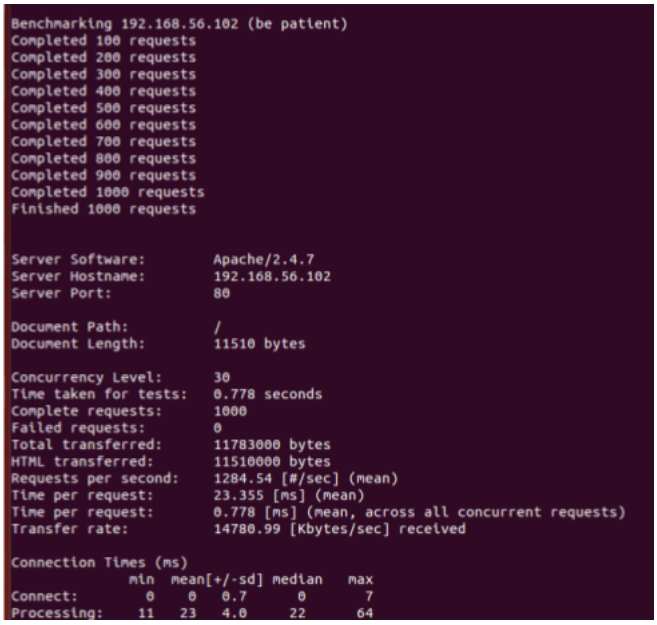

Figura 3.2: Comando ab contra Ubuntu Server

Vemos que hay diferencias significativas en los tiempos por petición, así como en las transferencias de datos, aunque esto último es más bien debido al propio formato de la página que nos sirve Windows y la que nos sirve Ubuntu, pero en general, y como era de esperar Ubuntu sale ganando para variar.

4. Jmeter. 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.).

Dado que el programa viene un jar listo para ser ejecutado en la máquina de java, tras descargarlo solo debemos hacer:

```
java -jar ApacheJmeter.jar
```

Primero he probado con una página sencilla servida en localhost(127.0.0.1), por tanto he configurado un grupo de hilos, que haga peticiones a la ip 127.0.0.1 y que tenga dos peticiones una que apunte a /, que sería el index de la web, y otra que apunte a /ficticia, que no existe para que reciba error también.

Seguidamente he realizado lo mismo con una web pública, en concreto `http://www.fermasa.org` que pertenece a la feria de muestras de Armilla. Los parámetros para realizar el test han sido en ambos casos:

```
Nº Hilos: 5
Método: GET
Urls: '/', '/ficticia'
Número repeticiones: 100
```

A continuación se muestran unas capturas de la configuración de Jmeter:

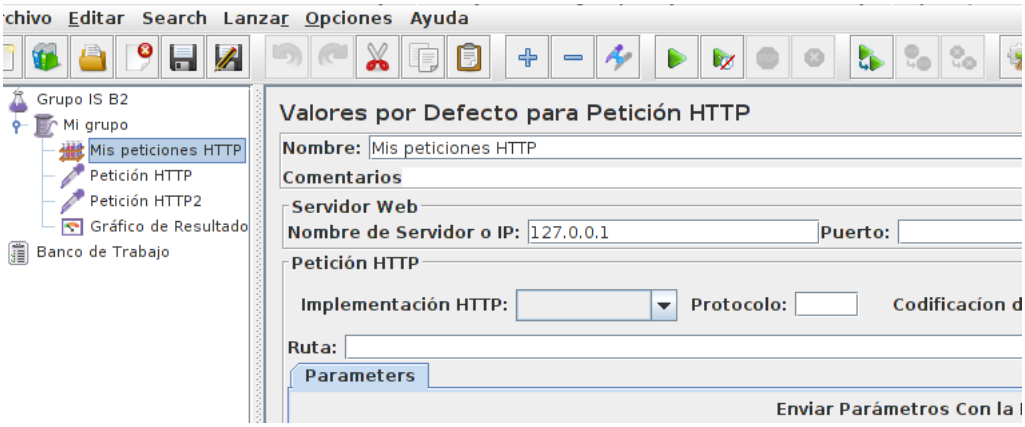

Figura 4.1: Configuración del servidor destino

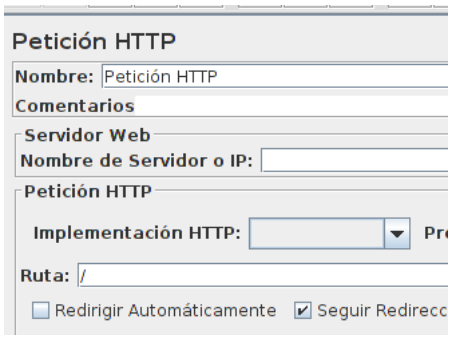

Figura 4.2: Configuración de la petición 1

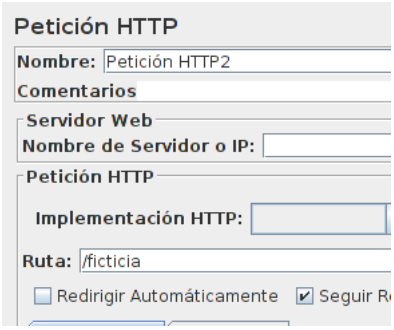

Figura 4.3: Configuración de la petición 2

Resultados del test:

Figura 4.4: Gráfico de la web servida en localhost

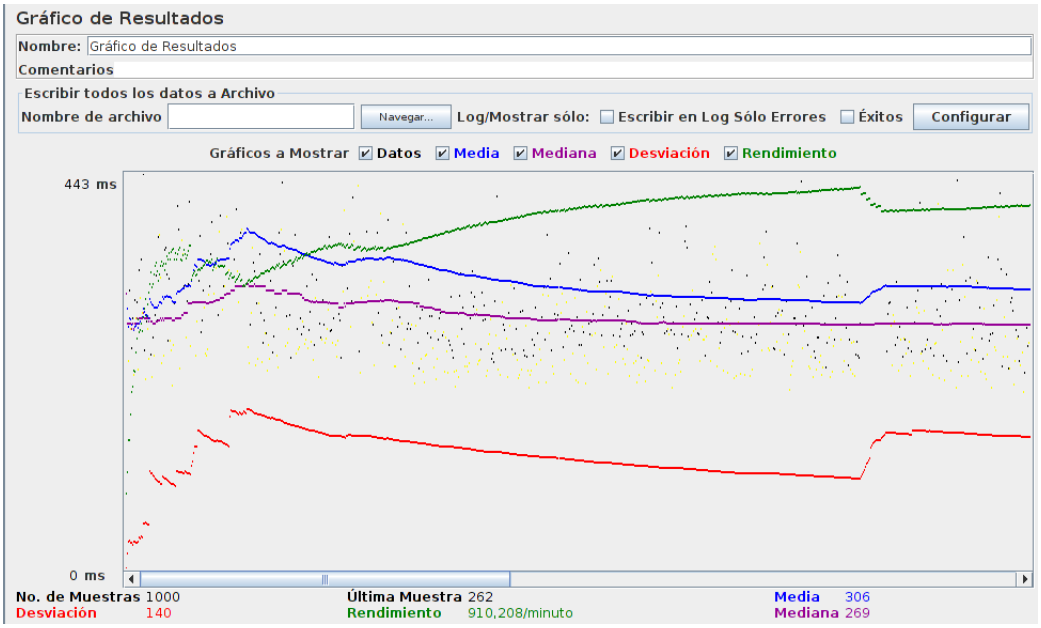

Figura 4.5: Gráfico de `www.fermasa.org`

Como vemos se nota bastante la diferencia de hacer un simple GET en un entorno local a un entorno externo, vemos que en local es uniforme mientras que en la url externa dependemos de otros factores como son el tiempo de respuesta del servidor, nuestra conexión a internet, etc ...

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

Tenga en cuenta que puede comparar varios gestores de BD, lenguajes de programación web (tiempos de ejecución, gestión de memoria, ...), duraciónde la batería, etc.

Supongamos que tenemos una aplicación en nuestro servidor que necesitará captar tuplas de datos de mil en mil, que recibirá de una fuente externa. Necesitamos almacenar esas mil tuplas en una base de datos, pero necesitamos que se almacenen rápidamente pues si llega otra tanda de mil tuplas y aún estamos guardando datos se despreciarían las nuevas, perdiendo información, por tanto necesitamos rapidez en esta operación de almacenaje.Las tandas de 1000 tuplas llegan con una frecuencia de 1 o 2 minutos. El sistema

esta montado con Python[5], y las opciones para gestor de base de datos que tenemos son Sqlite[6] y MySQL[7]. El formato de cada tupla es el siguiente:

`id(int),valor(hash md5)`

La creación del hash en md5 la realizará el propio Python también, pero al ser constante en ambas opciones se desprecia el tiempo que emplea en hacer la conversión, mi interés es ver cuanto tarda en insertar las tuplas. Se ha aislado el código encargado de insertar las tuplas únicamente en dos scripts para medir con precisión el tiempo que tardan en ejecutarse para mil tuplas. Para medir el tiempo usaremos el comando **time**[8] propio de Linux. A continuación detallo el código de ambos scripts:

```
# script mysql.py
import MySQLdb
from hashlib import md5
DB_HOST = 'localhost'
DB_USER = 'root'
DB_PASS = 'pass'
DB_NAME = 'datos_usuario'
datosc = [DB_HOST, DB_USER, DB_PASS, DB_NAME]
conn = MySQLdb.connect(*datosc) # Conectar a la base de datos
cursor = conn.cursor()          # Crear un cursor
for i in range(0,1000):
    id = str(i)
    hash = id + "relleno"
    cursor.execute("""INSERT INTO datos VALUES (%s,%s)""", (id,md5(hash).hexdigest()))
    conn.commit()
cursor.close()                  # Cerrar el cursor
conn.close()                    # Cerrar la conexion
print "1000 tuplas introducidas!"

# script sqlite.py
import web
from hashlib import md5
db = web.database(dbn="sqlite", db="datos.db")
for i in range(0,1000):
    id = str(i)
    hash = id + "relleno"
    db.insert("datos",ID=id,password=md5(hash).hexdigest())
    i += 1
print "1000 tuplas introducidas!"
```

[5] https://www.python.org/
[6] http://www.sqlite.org/
[7] http://www.mysql.com/
[8] El comando time muestra por la terminal el tiempo de ejecución de la orden que lo acompañe, ver man time para más información.

La ejecución del benchmark sería como sigue:

```
time python mysql.py
time python sqlite.py
```

Tras ejecutar 5 veces las órdenes anteriores vemos que nos da los siguientes tiempos medios para cada benchmark:

```
Usando MySQL: 45.313 segundos
Usando Sqlite: 2 minutos y 50.561 segundos
```

Vemos que usando MySQL obtenemos un tiempo casi 4 veces menor, por tanto elegiré MySQL para almacenar los datos sin lugar a dudas.

6. Cuestiones Opcionales

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

Scala es un acrónimo de 'Scalable Language'. Esto viene a ser un lenguaje que se adapta a nuestras necesidades de escalabilidad. Entre sus características principales podríamos destacar:
Orientado a objetos, todo en Scala es un objeto, al estilo de Java (de hecho correo en máquina Java y es compatible al 100 % con aplicaciones Java). Funcional, era de esperar que podamos tener funciones de todo tipo en un lenguaje orientado a objetos. Tipado estrictamente, al más puro estilo C. En general es similar a Java salvando algunos detalles.
Para instalar Gatling en Ubuntu basta con tirar de repositorios oficiales, es decir:

```
sudo apt-get install gatling
```

Gatling sirve básicamente para dar servicio HTTP y FTP, para más información sobre su uso podemos consultar 'man gatling'

6.2. Ha sido comparado por la empresa Flood.io con Gatling obteniendo la conclusión de que ambos proporcionan tienen un comportamiento y capacidades similares (https://flood.io/blog/11-benchmarking-jmeter-and-gatling). Cuestión opcional 3: Lea el artículo y elabore un breve resumen.

Se van a medir dos entornos distintios, ambos usando nginx. Usará flood para generar carga. Dicha carga vemos que se compone de 20 % instrucciones fetch, 40 % saltos condicionales, 30 % fetch sin caché y el 10 % restante de instrucciones posting. Simulamos que hubiése 10000 usuarios simultáneos con 30000 pet/min, tras pasarle el gatling vemos que nos da 1788s, con jmeter 1625s y con jmeter 2.10 nos da 1698s.
Podemos ver resultados de red bastante similares, sin embargo uno de los entornos requirió bastante más CPU que el otro para obtener dichos resultados. Por tanto, y personalmente, yo eligiría el que menos CPU ha utilizado, claro que habría que ver otros aspectos como si me van a cobrar mucho más por éste y cosas así.