Servidor Web Tomcat

José Manuel Cuevas Muñoz, Daniel Ranchal Parrado, Carlos Romeo Muñoz

9 de diciembre de 2018

32 horas de trabajo

Índice general

Índice de figuras

Capítulo 1

Definición del Sistema

1.1. Objetivos y definición del sistema

El objetivo que se pretende es hacer pruebas exhaustivas, es decir, hacer un benchmarking para calcular el rendimiento del servicio web Tomcat. Para hacer este tipo de pruebas, se parte de la siguiente información. El sistema operativo bajo el que se están haciendo las pruebas es CentOS, cuyo servidor está alojado en Azure.
Para la instalación de Tomcat, se ha utilizado el servicio de virtualización docker para instalarlo de una manera sencilla. Para que el host, es decir, el servidor pueda dar el servicio que provee el contenedor de docker, hay que ligar el puerto del contenedor en el que está sirviendo Tomcat con el puerto que nosotros queramos en el host, aunque los autores de este guión y trabajo hemos elegido el 80, el puerto por excelencia para el servidor web.

1.2. Servicios y sus posibles resultados

El servidor web Tomcat nos provee una gran cantidad de utilidades para aprovechar al máximo este mismo. La primera y la más importante es el despliegue de una página web. Tomcat nos permite hacerlo de manera estática, es decir, configurar la aplicación antes de activar este servicio o de manera dinámica, que es lo más utilizado para servidores que están en producción.
Para poder realizar esto de una manera sencilla, Tomcat nos da una herramienta llamada "Manager", que cómo se ha comentado antes, se pueden hacer despliegues y eliminación (undeploy) de cualquier aplicación web además de darnos una lista de las aplicaciones que ya están desplegadas.

Pero las utilidades de "Manager"no son sólo esas, este gestor nos da la opción de poder ver las estadísticas de las sesiones de cualquier aplicación y el estado del servidor. Otra de las funcionalidades que tiene, al igual que el servidor httpd y apache2, es la posibilidad de trabajar con "Virtual Host".
Otra característica interesante que nos ofrece Tomcat es la gestión de los usuario y los roles en las distintas aplicaciones web. Esto evita que tengamos una tabla para cada aplicación web que se tenga. En temas de seguridad, Tomcat nos permite la configuración de los certificados SSL/TLS para que nuestra página tenga el protocolo https.
Para este experimento, se distinguirán los siguientes resultados:

- **La carga correcta de las imágenes en un tiempo considerablemente bueno.**
- **La carga correcta de las imágenes en un tiempo pésimo.**
- **La carga parcial de las imágenes.**
- **Fallo del servidor web. Ninguna imágen es servida.**

1.3. Métricas

Para poder medir la eficacia y la actuación del servidor web Tomcat se han considerado los siguientes parámetros a examinar:

- **Peticiones por segundo**
- **Tiempo por cada grupo de peticiones al mismo tiempo**
- **Tiempo por cada petición**
- **Uso de la CPU**
- **E/S Disco**
- **E/S Red**

1.4. Parámetros

Los parámetros que pueden afectar a la medición del rendimiento del servidor web Tomcat son los siguientes:

- **La lejanía con el centro de datos**
- **Los procesos que se ejecutan en segundo plano**
- **Estado de nuestra conexión a Internet**
- **Hardware empleado en el servidor**

Para poder mitigar los efectos que producen estos parámetros,la prueba se ejecutará varias veces y se harán los cáculos oportunos para representar de una manera correcta las estadísticas.

Capítulo 2

Evaluación del Sistema

Tras el montaje del servidor en Tomcat usando CentOS, procedemos a evaluar el sistema. En general hay muchas técnicas de evaluación posible. En nuestro caso vamos a realizar pruebas con Apache benchmark en el lado del cliente, y a monitorizar en el lado del servidor a través de sar y el monitor de Azure. Tras realizar esto, sacaremos los datos, tanto del lado del cliente como del servidor, y los analizaremos.

2.1. Técnicas de evaluación

Antes de proceder a describir los resultados, debemos mostrar cuáles fueron las técnicas utilizadas para obtener los resultados del benchmark. Para realizar estas pruebas, hemos hecho uso de Apache benchmark, debido a que es una de las principales utilidades a la hora de realizar benchmarking a un servidor web. Este permite mostrar pruebas con una concurrencia que nos sería difícil de imitar, además de sacar datos muy interesantes, como el tiempo medio de una petición y el número de peticiones del servidor. Para sacar los datos hemos utilizado el monitor sar, realizando un monitoreo completo a la vez que sucedían las sucesivas pruebas de Apache benchmark. El monitor sar nos permite sacar datos, tanto de la utilización de CPU, como de red, además de otros muchos datos, los cuales no resaltaremos. Por último, hemos utilizado datos que nos dá el propio Azure a la hora de medir el uso de disco.

2.2. Carga de trabajo

Hemos hecho uso de tres cargas distintas de ab. Cada carga se ha realizado con una concurrencia y una cantidad de peticiones distintas, además de que cada una de estas se ha realizado en cinco ocasiones en tres index: uno con fotos grandes, otro con fotos medianas y otro con fotos pequeñas, para comprobar si existen diferencias entre las tres versiones. Las cargas son:

- **ab -c 5 -n 20**: Pretende medir la capacidad del programa a la hora de tener una concurrencia media y una cantidad de peticiones moderada/alta.

- **ab -c 1 -n 10**: Pretende probar el servidor con una cantidad de peticiones más baja que la anterior prueba, pero sin concurrencia, para comprobar la diferencia entre una prueba con concurrencia o sin ella.

- **ab -c 10 -t 40 -s 40**: Pretende probar el servidor con una concurrencia alta en un tiempo, dándonos, por ejemplo, el número de peticiones que puede realizar el servidor en x segundos.

2.3. Diseño de Experimentos

Para realizar estos experimentos he creado un script que realiza en cinco ocasiones las pruebas anteriormente nombradas para cada tamaño de imagen. A su vez, el monitor sar está corriendo en el servidor y guardándolo en un fichero. Para mostrar gráficamente los resultados de estas pruebas se hace uso de gnuplot.

2.4. Análisis de los resultados

Tras realizar ab y a los tres index con las distintas cargas podemos proceder a mostrar sus datos. Los primeros datos que nos resultarían interesantes serían los del propio Apache benchmark. Los primeros datos que podemos mirar son los que nos da el propio apache benchmark. Estos datos, obtenidos de la propia salida del comando “ab” son los siguientes. De la primera prueba se extrae lo siguiente:

	Pequeños			Medianos			Grandes		
Ejecución	Pet/s	ms/Petición	Total(s)	Pet/s	ms/Petición	Total(s)	Pet/s	ms/Petición	Total(s)
1	0.17	28875.345	115.501	0.16	31396.260	125.585	0.18	27800.182	111.201
2	0.17	29610.067	118.440	0.17	30268.598	121.074	0.18	27508.504	110.034
3	0.17	29360.455	117.442	0.17	30065.704	120.263	0.18	27296.960	109.185
4	0.17	29445.476	117.782	0.16	32097.834	128.391	0.18	27261.818	109.047
5	0.16	28917.908	115.672	0.17	29742.525	118.970	0.18	26643.844	106.575
Media	0.168	29241.8502	116.9674	0.166	30714.1842	122.8566	0.188	27302.2616	109.2084

Es curioso el hecho de que la variación del tiempo no dependa tanto del tamaño como de la capacidad de la red en ese momento. Se puede ver que aunque en los valores medianos sube el tiempo, tanto por petición como el total, con respecto a los valores pequeños; en los grandes esta sube debido a que la red no está siendo utilizada.

Tras la segunda prueba nos da los siguientes resultados:

	Pequeños			Medianos			Grandes		
Ejecución	Pet/s	ms/Petición	Total(s)	Pet/s	ms/Petición	Total(s)	Pet/s	ms/Petición	Total(s)
1	0.17	5897.409	58.974	0.16	6207.914	62.079	0.17	5732.952	57.330
2	0.17	5952.298	59.523	0.16	6166.424	61.664	0.17	5792.583	57.926
3	0.17	5968.026	59.680	0.16	6159.262	61.593	0.17	5720.497	57.205
4	0.16	6904.65	60.946	0.16	6291.626	62.916	0.17	5761.08	57.611
5	0.17	6033.483	60.335	0.16	6239.134	62.391	0.17	5817.991	58.180
Media	0.168	5989.1642	59.8916	0.16	6212.8708	62.1286	0.17	5765.0206	57.6504

Se puede comprobar que el tiempo de respuesta en este caso, sin ninguna concurrencia, es sustancialmente menor al tiempo con concurrencia (de media, unas 4.853981 veces menor), debido a que con más de dos peticiones a la vez el servidor se satura debido a su baja capacidad de cómputo.

Por último, en el último test, siendo los dos únicos datos que nos interesan la media de peticiones que son capaces de resolver en ese tiempo y el tiempo de respuesta por petición.

	Pequeños		Medianos		Grandes	
Ejecución	Pet. hechas	ms/Petición	Pet. hechas	ms/Petición	Pet. hechas	ms/Petición
1	4	88795.308	4	110129.467	2	218473.645
2	3	107076.853	3	130129.422	3	157925.583
3	4	86692.71	4	108129.467	4	110129.562
4	5	66522.661	2	160291.622	3	142129.465
5	3	108506.811	4	110129.467	4	100345.234
Media	3.8	75914.1808	3.4	123761.889	3.2	145800.6978

Aqui si se puede notar una clara diferencia entre la performance entre la versión pequeña, la mediana y la grande, así como la del tiempo de respuesta entre las distintas ejecuciones. La principal diferencia de esta prueba con respecto a las otras dos es que, además de ser de tiempo, también tiene una concurrencia alta, sobre todo teniendo en cuenta, como veremos posteriormente, que la cpu a partir de dos concurrencias llega casi al 100 % de utilización.

A continuación procedemos a sacar con sar -u los datos de la CPU que se han sacado durante las distintas ejecuciones de ab. Con gnuplot los ponemos de manera visual, siendo la abscisa Y el porcentaje de uso de cpu y la abscisa X el tiempo.

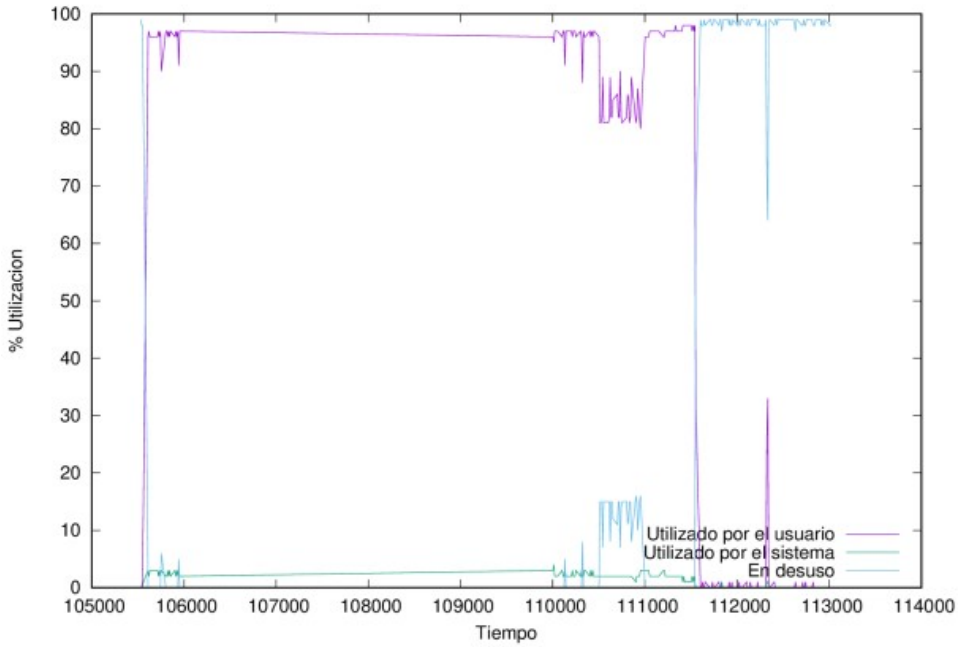

Figura 2.1: Uso de CPU con imágenes grandes

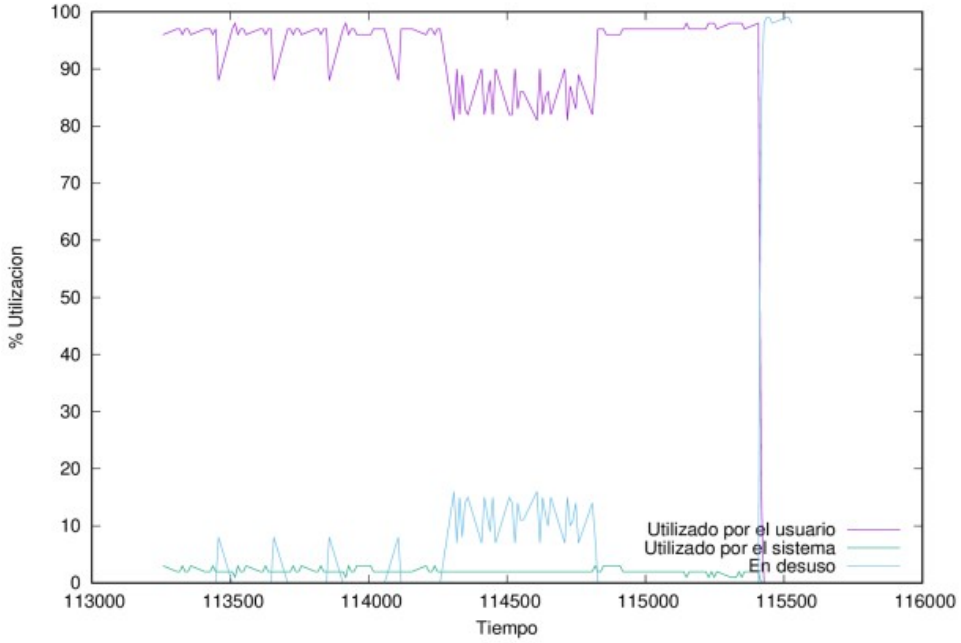

Figura 2.2: Uso de CPU con imágenes medianas

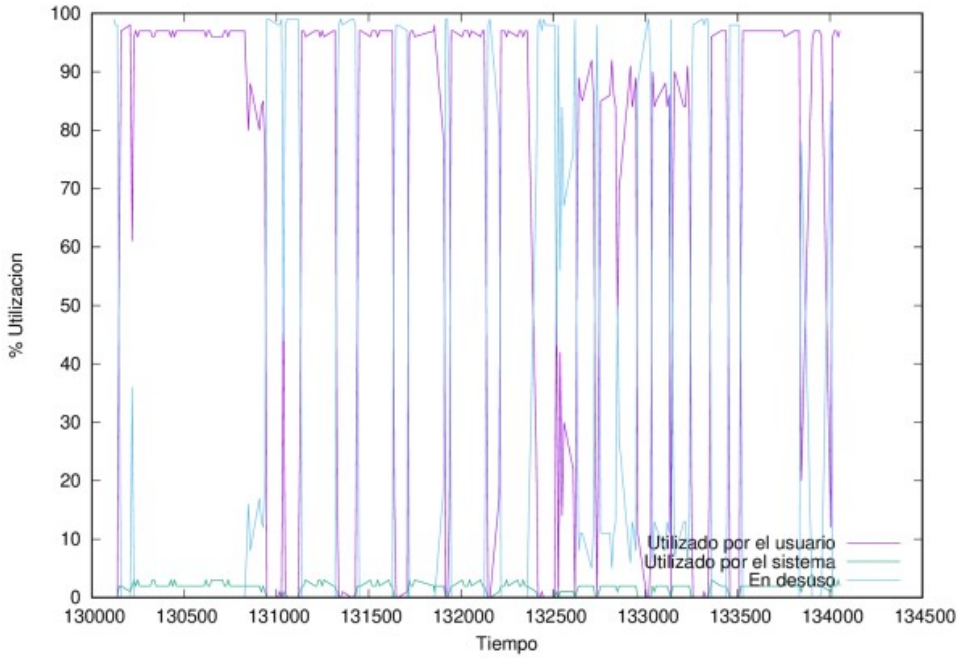

Figura 2.3: Uso de CPU con imágenes medianas

Lo primero que se puede notar es que para una concurrencia mayor que uno, el servidor utiliza la cpu casi al 100 %, siendo el único momento donde este uso baja, cuando llega el momento de la prueba 2, donde no hay concurrencia y solo utiliza un 80 % del disco.
En cuanto a las pruebas de red, he utilizado sar -n DEV para comprobar todos los paquetes que se han enviado. Además de esto, he usado sar -n EDEV para comprobar aquellos paquetes que no han sido recibidos, dando como resultado que en ninguna de las pruebas ha habido paquetes fallidos. Para mostrar los resultados, he vuelto a hacer uso de gnuplot, siendo la abscisa Y los kbs, tanto transmitidos como recibidos, y la abscisa X el tiempo.

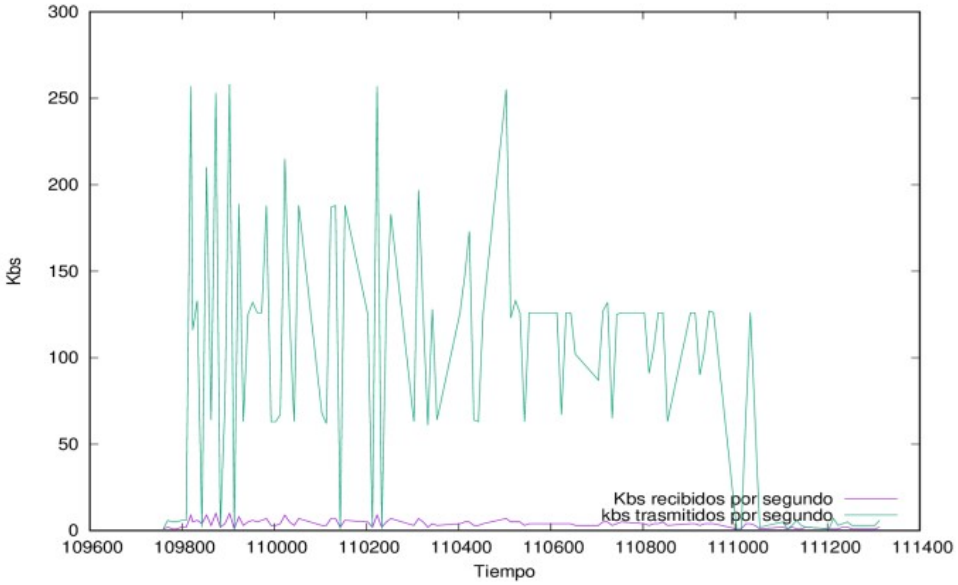

Figura 2.4: Uso de Red con imágenes grandes

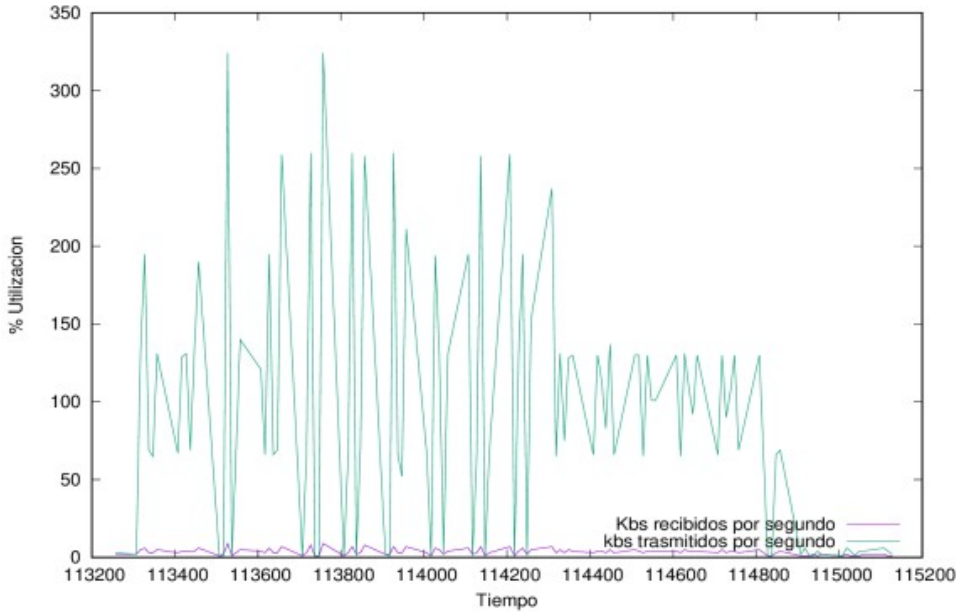

Figura 2.5: Uso de Red con imágenes medianas

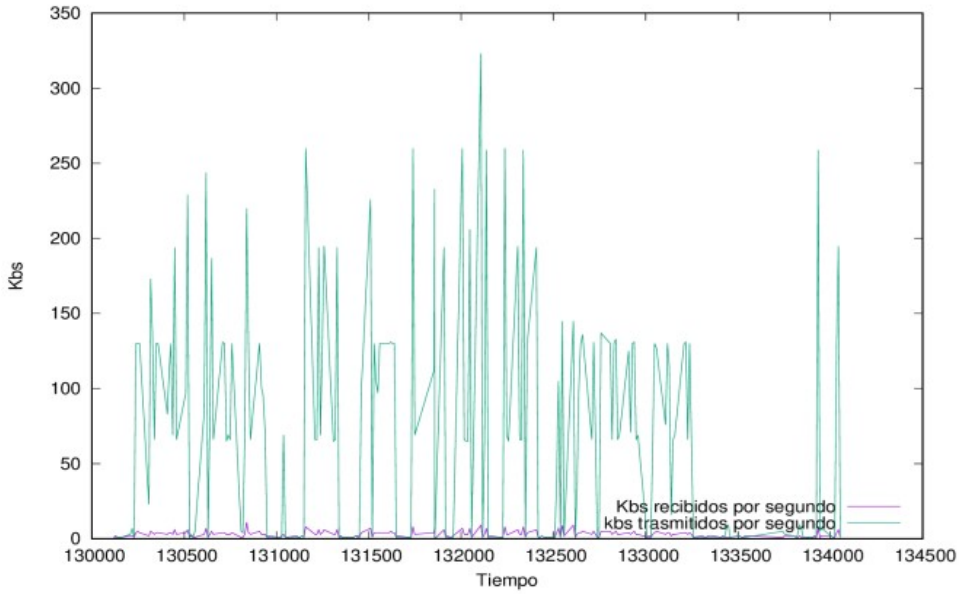

Figura 2.6: Uso de Red con imágenes pequeñas

Se puede notar como el ritmo de transferencia da muchas subidas y bajadas. Aun así los datos son más o menos estables, siendo el punto en el que es más baja

la transferencia durante el tercer test, mientras que el punto en el que alcanza sus picos es en el segundo test.
Por último, haciendo uso del monitor de azure y sacando media de los datos durante las ejecuciones de los benchmark, podemos sacar los datos de utilización de disco. Para mostrar estos datos, se ha hecho uso del excel, mostrando en ese caso la media por ejecución.

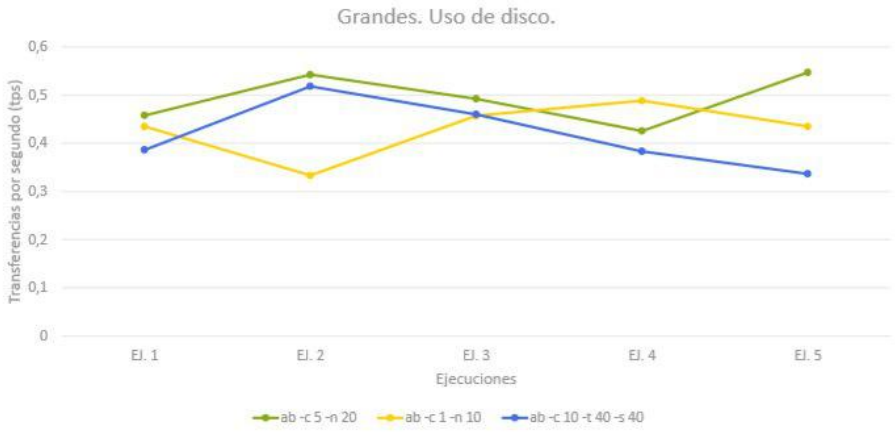

Figura 2.7: Uso de disco en las cinco ejecuciones con imágenes grandes

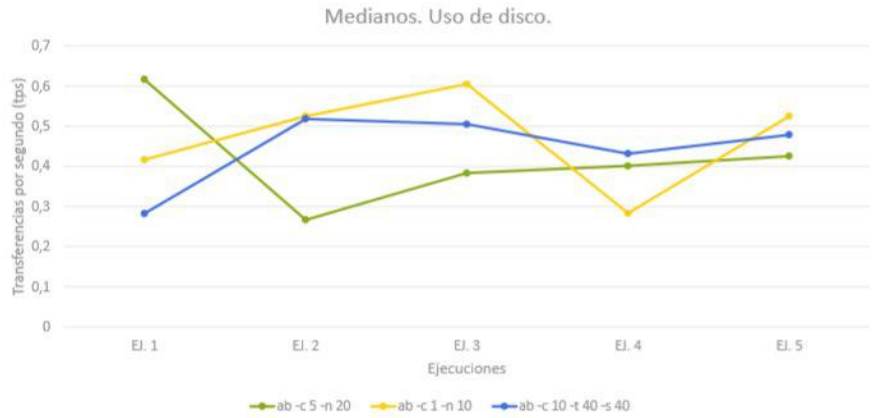

Figura 2.8: Uso de disco en las cinco ejecuciones con imágenes medianas

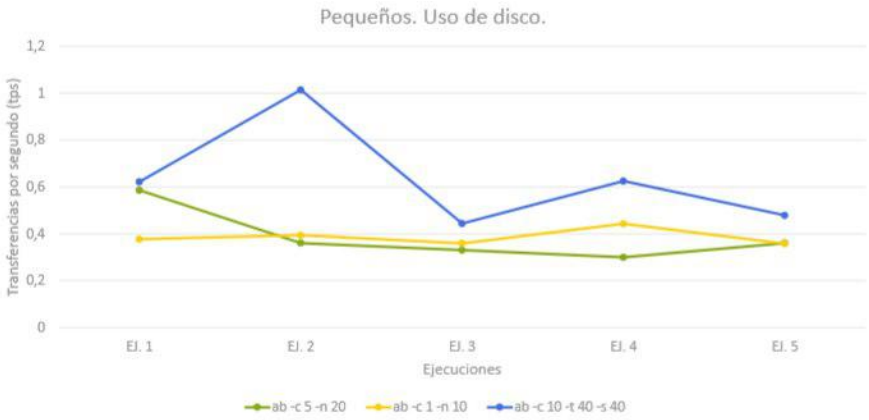

Figura 2.9: Uso de disco en las cinco ejecuciones con imágenes pequeñas

Se pueden ver como las transferencias son muy parecidas, tanto entre benchmark como entre los distintos index.

Capítulo 3

Conclusiones y discusión

En primer lugar, se aprecia un fuerte problema en el hardware de este servidor debido al cuello de botella que aparece al ejecutar los bencharmarks en la CPU, esto es fácil de comprender, debido a que la concurrencia toma protagonismo en estos tests, con algunos fijándola en valores como 10 peticiones concurrentes. Esta teoría es afirmada por las gráficas mostradas, donde vemos fácilmente como el procesador llega a funcionar a niveles cercanos al 100 % de utilidad durante la duración de las pruebas efectuadas.

Además de esto, otro factor determinante en este caso es el bajo número de transferencias por segundo que el servidor es capaz de aguantar y que está fuertemente relacionado con el almacenamiento del mismo en el cual se almacenan las fotografías que sirven como cargas de prueba, el cual en ninguna prueba es capaz de llegar ni tan siquiera a la unidad, y eso es algo que los encargados de testear su funcionamiento hemos conocido de primera mano al tener que esperar grandes cantidades de tiempo a que todos los tests fueran completados varias veces con éxito. En cuanto a las peticiones por segundo, casi nunca superan el cuarto de unidad, hecho que no hace más que corroborar las afirmaciones anteriores.

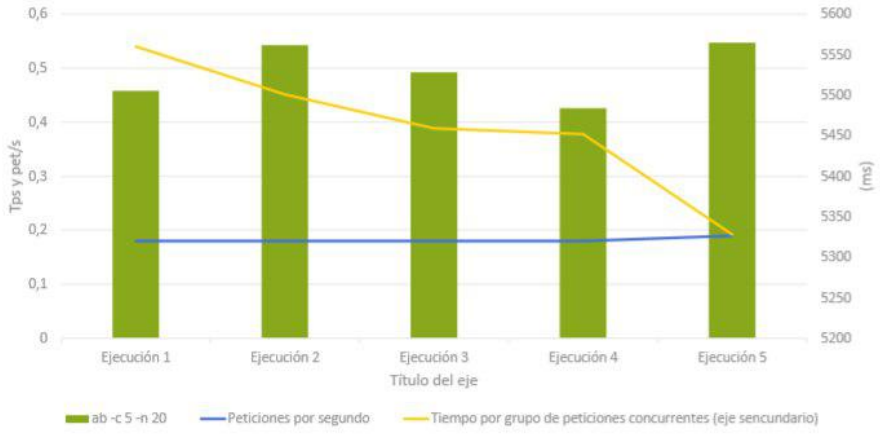

Figura 3.1: Transferencias por Segundo, peticiones por segundo y tiempo empleado en un grupo de peticiones concurrentes.

Si contemplamos el gráfico 3.1, nos damos cuenta de que el número de peticiones por segundo, al ser un parámetro íntimamente ligado al hardware sobre el que se esté ejecutando el banco de pruebas escogido, apenas varía, puesto que la CPU solo es capaz de ejecutar un número determinado de ellas, siendo todas iguales. Lo interesante de este gráfico, es ver como el tiempo empleado en servir un grupo de peticiones concurrentes decae a lo largo de las cinco ejecuciones del test. Tras indagar en los motivos de este aumento del rendimiento, nos damos cuenta de que Azure es un servicio de computación en la nube que alardea de proveer de una caché inteligente a sus máquinas virtuales que estén corriendo en sus servidores, como es nuestro caso, eso por ello por lo que se aprecia esta mejora en el rendimiento, debido a que gran parte de los datos que son necesarios para la ejecución de la prueba, ya están cargados en la caché del servidor y son accesibles de manera mucho más rápida para la siguiente ejecución.

Si contemplamos las gráficas 2.7, 2.8 y 2.9, no hay gran diferencia entre ellas en cuanto al parámetro de medición, el número de transferencias por segundo del disco del servidor. Para analizar las gráficas, recordemos que estos datos han sido tomados con el monitor sar, mientras que ab era ejecutado, tras ello, se ha realizado la media aritmética de los valores del parámetro interesado que sar tomaba cada 10 segundos durante la ejecución del test. Entonces, podemos ver como apenas hay diferencias significativas entre las ejecuciones, por ejemplo, en la gráfica 2.7, vemos como normalmente el test de menor concurrencia y menos repeticiones

(línea amarilla), tarda normalmente menos tiempo que el test con más concurrencia y el doble de repeticiones (línea verde), aunque veamos ejecuciones donde estas posiciones se inviertan, esto se debe más al margen de error que a otra cosa. En cuanto a la gráfica 2.9, la referida a los resultados del mosaico de imágenes pequeñas, observamos como la línea azul, la correspondiente con el uso de https, lo cual es curioso debido a que en las dos últimas pruebas se habían mantenido por debajo, sin embargo, no es de preocupación, ya que no se aprecian diferencias muy significativas en la mayoría de las ejecuciones, sin embargo, vemos como esa línea sube hasta una transferencia por segundo en la segunda ejecución, lo cual es muy significativo. La imagen general de estas pruebas es que mientras más grandes sean las imágenes del banco de pruebas, menos transferencias por segundo se llevarán a cabo, lo cual es lógico, puesto que son archivos más grandes.
Si analizamos ahora el uso de la red del servidor, aquí no tenemos ningún problema de cuello de botella, Azure nos proporciona una velocidad de conexión veloz y estable y no es algo que un administrador de un servidor pequeño como este tenga que preocuparse, por supuesto, mientras más exigente sea el test con la CPU, lo será con el uso de red que éste requerirá, puesto que estamos hablando de un servidor web.

En conclusión, hemos detectado que nuestro cuello de botella es una CPU que a pesar de ser muy potente en el conjunto de todos sus núcleos, apenas aguantaría un servidor web pequeño como este ante un importante número de peticiones al servidor, esto es debido a que la cuenta de estudiante de Microsoft Azure que estamos empleando para este trabajo solo ofrece un núcleo de procesador, potente, si, pero que flaquea en tareas que involucren una alta concurrencia de peticiones. Y es que ese es el argumento por el cual los procesadores empleados para su uso en servidores de cualquier tipo suelen contar con un gran número de núcleos tanto físicos como lógicos y poder procesar rápidamente todas las peticiones que reciba. En cuanto a la referenciación, sabemos que los resultados aquí mostrados corresponden con los de un servidor de especificaciones limitadas, útil para este propósito, pero de limitado uso, y es que, tras haber ejecutado numerosos bancos de pruebas centrados en multitud de aspectos de un sistema informático, entre ellos Apache Benchmark, hemos encontrado que fácilmente, un ordenador doméstico con todos sus núcleos activados puede sobrepasar fácilmente el rendimiento de este procesador, organizaciones como SPEC tienen precisamente esa función, la de examinar este tipo de equipos informáticos para poder realizar comparativas,

lo cual nos es muy útil a la hora de sacar conclusiones sobre el desempeño de este hardware. Por último, hemos de decir, que Microsoft Azure es una útil herramienta para alojar cualquier tipo de servidor o aplicación web y que nos ha sido muy útil para la realización de este trabajo y que, previo pago de la capacidad necesaria, se puede obtener muchísima potencia de cómputo de ella.

3.1. Cuestiones

- **¿Qué factores pueden hacer variar el tiempo de respuesta a la hora de realizar una petición? ¿Como puede el servidor mejorar este?**
  El tiempo de respuesta puede variar por factores, tanto del lado del cliente como del del servidor. Del lado del cliente puede variar por la lejanía al centro del servidor o el estado de la conexión a internet. Del lado del servidor, la cantidad de llamadas en un instante y el hardware son las causas de esta variación. El servidor puede mejorar el tiempo de respuesta variando el hardware, viendo cual es el cuello de botella y mejorando sus componentes, de tal manera que pueda encontrar un punto en el que sea capaz de encontrar una capacidad de cómputo suficiente para sus peticiones.

- **¿Por qué repetimos varias veces el test de ab y con varias configuraciones?**
  Al ejecutar las distintas configuraciones de ab varias veces estamos logrando dos cosas: la primera, bajo una misma configuración de prueba tomamos varias medidas en cada una de las cinco ejecuciones del test y aplicamos la media artimética sobre cada una de ellas con el fin de obtener la máxima fidelidad de los datos a una hipotética situación real de trabajo del servidor. El segundo logro corresponde con el comprobamiento del funcionamiento del servidor bajo diferentes condiciones de trabajo, por ejemplo variando la concurrencia de trabajos que ha de procesar. Combinando estos dos procedimientos, nos hacemos una idea de la potencia que tiene el servidor y su desempeño en una situación real.

- **Teniendo en cuenta todos los datos extraidos del servidor web, ¿Es rentable y de utilidad Tomcat?**
  Respecto a la utilidad, Tomcat restringe o más bien, imposibilita aquellas aplicaciones web que no sean hechas con Java y el lenguaje de marcado html, y por lo tanto, deja atrás lenguajes útiles para este campo de la infor-

mática como PHP. Respecto a la correcta configuración de Tomcat, es muy fácil y nos proporciona herramientas ya mencionadas en la memoria. Pero el mayor problema para utilizar Tomcat en un sistema basado en Linux es la instalación de Java Virtual Machine, es bastante complicado de instalar y por ello, a lo mejor, Tomcat no es tán utilizado como nginx, apache2 o httpd. Teniendo en cuenta las características del servidor que sirve Tomcat, es lo suficientemente bueno para soportar la funcionalidad de los archivos jsp.

Bibliografía

[1] Michael Kerrisk. *The Linux man-pages project*.

[2] Aula de Software Libre de la Universidad de Córdoba. *Taller de Docker*, 2018.

[3] Microsoft Azure. *Máquinas virtuales Linux*, 2018.