

MEMORIA PRÁCTICA 2

Rendimiento

Alejandro Santorum Varela - alejandro.santorum@estudiante.uam.es

Rafael Sánchez Sánchez - rafael.sanchezs@estudiante.uam.es

Prácticas Sistemas Informáticos II

Práctica 2 Pareja 7 Grupo 2401

8 de abril de 2020

Contents

I	Introducción	2
1	Ejercicio 1	2
2	Ejercicio 2	3
3	Ejercicio 3	7
4	Ejercicio 4	9
5	Ejercicio 5	10
6	Ejercicio 6	10
7	Ejercicio 7	14
8	Ejercicio 8	15
9	Ejercicio 9	16
III	Conclusiones	18
IV	Bibliografía	18

I Introducción

El objetivo de esta práctica es aprender a medir el rendimiento de una aplicación JAVA EE, así como conocer las implicaciones que algunas decisiones de diseño o de arquitectura pueden tener en el rendimiento de la aplicación. En ella se estudiarán los siguientes aspectos:

- Aspectos generales del rendimiento de una aplicación J2EE.
- Influencia en el rendimiento de la configuración del servidor de aplicaciones.
- Cadena de procesamiento dentro del servidor de aplicaciones.
- Elementos que la componen.
- Puntos de encolamiento.
- Obtención de la curva de productividad.
- Configuración del servidor de aplicaciones de cara al rendimiento · Mecanismos de evaluación del rendimiento y realización de pruebas unitarias.
- Pruebas de stress de una aplicación Java EE: Automatización de un plan de pruebas con la herramienta JMeter.

1 Ejercicio 1

Enunciado:

Siguiendo todos los pasos anteriores, defina el plan completo de pruebas para realizar las tres ejecuciones secuenciales sobre los tres proyectos definidos hasta ahora (P1-base, P1-ws, P1-ejb). Adjunte el fichero generado P2.jmx al entregable de la práctica.

Importante: Para comprobar el correcto funcionamiento de la simulación y detectar posibles fallos, se recomienda añadir también al elemento P2 Test un “árbol de resultados” (View Results Tree). Para ello, sobre el plan de pruebas, botón derecho, *Add > Listener > ViewResults > Tree*. Una vez se tenga la certeza de que la simulación funciona correctamente se desactivará el “árbol de resultados” (pulsando encima con el botón derecho del ratón) y se realizará de nuevo la simulación. El árbol de resultados permite inspeccionar los datos enviados en cada petición HTTP y la respuesta obtenida del servidor, que deberán ser correctas. Por ejemplo, no deberá aparecer ningún pago incorrecto en las respuestas.

Respuesta a la cuestión:

Seguimos todos los pasos del apartado 6 del enunciado de la práctica, los cuales son resumidos en los siguientes puntos:

- Instalamos JMeter.
- Desplegamos las prácticas P1-base, P1-ws y P1-ejb (servidor remoto), sin olvidar cambiar el nombre a la P1-base para que se despliegue con dicho nombre.
- Creamos un plan de pruebas llamado P2 test, añadiendo el elemento de configuración “HTTP Request Defaults”.
- En el cuadro de diálogo configuramos la dirección IP de PC2VM (10.1.7.2), el puerto del servidor (8080) y los parámetros que en principio no cambian entre pruebas.
- Añadimos la variable de usuario **samples**, que nos permitirá indicar el número de muestras que queremos tomar en todas las pruebas, dándole el valor inicial 1000.
- Introducimos un thread group (grupo de hilos) que simulará el conjunto de usuarios que accede a la aplicación. En esta primera prueba, simularemos un único usuario.
- Añadimos una variable aleatoria que permitirá que el importe de cada pago tenga un valor comprendido entre 1 y 1000, relacionando éste como la variable “importe” para uso en adelante.
- “Simulamos” la generación del identificador de las transacciones desde JMeter, para proporcionárselo a la página de pagos.
- El resto de los datos del pago (número de tarjeta, titular, fecha de emisión, caducidad y código de verificación de la tarjeta) los tomaremos aleatoriamente de un archivo de datos .csv. Se proporciona un fichero de ejemplo (datagen/listado.csv).

- Especificaremos la ruta relativa al archivo, el nombre de las variables que se definirán para cada campo separadas por comas y el carácter de separador (barra vertical).
- Añadimos el generador de peticiones HTTP.
- Añadimos al plan de pruebas el *listener* "Aggregate Report" que presenta información muy útil.
- Generamos el resto de casos copiando y pegando lo hecho anteriormente. A continuación deberemos renombrar los generadores de peticiones, el *path* de los generadores de peticiones, el valor inicial del idTransaccion y el valor del identificador del comercio.
- Pulsamos sobre el plan de pruebas y marcamos "Run thread groups consecutively" para que los tests se ejecuten de forma secuencial.
- Finalmente, añadimos el árbol de resultados para comprobar que la simulación se satisface.

Después de realizar todo esto ejecutamos secuencialmente los planes de pruebas y añadimos el **fichero P2.jmx** al entregable de la práctica.

2 Ejercicio 2

Enunciado:

Ejercicio 2: Preparar los PCs con el esquema descrito en la Figura 22. Para ello:

- Anote en la memoria de prácticas las direcciones IP asignadas a cada PC.
 - **Detenga el servidor de GlassFish de los PCs físicos**
-

- Inicie los servidores **GlassFish en las máquinas virtuales**
- **Repliegue** todas las aplicaciones o pruebas anteriores (P1-base, P1-ws, etc), para limpiar posibles versiones incorrectas.
- **Revise y modifique** si es necesario los ficheros build.properties (propiedad "nombre") de cada versión, de modo que todas las versiones tengan como URL de despliegue las anteriormente indicadas.
- **Revise y modifique** si es necesario el fichero glassfish-web.xml, para indicar la IP del EJB remoto que usa P1-ejb-cliente.
- **Despliegue** las siguientes prácticas: **P1-base, P1-ws, P1-ejb-servidor-remoto y P1-jeb-cliente-remoto, con el siguiente esquema:**
 - El destino de despliegue de la aplicación P1-base será PC2VM con IP 10.X.Y.2 (as.host)
 - El destino del despliegue de la parte cliente de P1-ws y de P1-ejb-cliente-remoto será PC2VM con IP 10.X.Y.2 (as.host.client de P1-ws y as.host de P1-ejb-cliente-remoto)
 - El destino del despliegue de la parte servidor de P1-ws y de P1-ejb-servidor-remoto será PC1VM con IP 10.X.Y.1 (as.host.server de P1-ws y as.host.server y as.host.client de P1-ejb-servidor-remoto)
 - La base de datos en todos ellos será la de PC1VM con IP 10.X.Y.1 (db.host)

Tras detener/iniciar todos los elementos indicados, anotar la salida del comando "**free**" así como un pantallazo del comando "**nmon**" (pulsaremos la tecla "m" para obtener el estado de la RAM) tanto en las máquinas virtuales como los PCs físicos. Anote sus comentarios en la memoria.

Pruebe a ejecutar un pago "de calentamiento" por cada uno de los métodos anteriores y verifique que funciona a través de la página testbd.jsp.

Respuesta a la cuestión:

Para este ejercicio (y para todos los que siguen) tendremos que realizar un pequeño cambio debido a las circunstancias en las que nos encontramos: trabajando telemáticamente desde casa y con menos recursos que en los laboratorios. Por consiguiente, el esquema de dos PC's propuestos se ha sustituido por un único PC (con **dirección IP asignada 10.10.0.18**) donde ejecutamos JMeter, dos máquinas virtuales (una con glassfish y PostgreSQL, y la otra únicamente con glassfish activado) y las demás utilidades, como **nmon**.

Seguimos las instrucciones del enunciado anteriormente citadas para desplegar las siguientes prácticas: P1-base, P1-ws, P1-ejb-servidor-remoto y P1-jeb-cliente-remoto.

Tras configurar todos los elementos indicados, mostramos a continuación la salida del comando **free** así como un pantallazo del comando **nmon** (pulsaremos la tecla “m” para obtener el estado de la RAM) tanto en las máquinas virtuales como en el PC físico.

Primero mostramos las salidas del comando free. Empecemos con el PC físico:

```
lletfrix@ps42:~$ free
              total usado libre compartido búfer/caché disponible
Memoria:    16323944    3564956    8193048    1786984    4565940    10662204
Swap:       3999740         0    3999740
```

Los valores que se muestran en las salidas son el Kibibytes(KiB), es decir, $1024B = 1MB$. Con esto podemos ver que según este comando el ordenador físico tiene 16.72 GB de memoria RAM, y actualmente tiene 3.56 GB en uso y 8.19 GB libres. La memoria *swap* es un espacio de intercambio entre la memoria física y la memoria virtual, en este caso con un tamaño de 3.99 GB.

A continuación mostramos la salida del comando **free** en la MV1.

```
si2@si2srv01:~$ free
              total      used      free      shared    buffers    cached
Mem:         767168    558188    208980         0        24500    182676
-/+ buffers/cache:    351012    416156
Swap:        153592         0    153592
```

Para el caso de las máquinas virtuales vemos unas estadísticas muy parecidas, con un espacio total de aproximadamente 750MB , 558MB en uso en la MV1 y 532MB en la MV2; y 210MB libres en la MV1 y 235MB en la MV2. Los datos de la MV2 se puede observar en la siguiente imagen.

```
si2@si2srv02:~$ free
              total      used      free      shared    buffers    cached
Mem:         767168    532596    234572         0        17612    163188
-/+ buffers/cache:    351796    415372
Swap:        153592         0    153592
```

Por otro lado, también podemos recaudar partes de los datos obtenidos con el comando free, pero esta vez utilizando el comando **nmon -m**. En las tres siguientes imágenes se nos muestran los resultados de dicho comando para el PC físico, MV1 y MV2 respectivamente.

```
nmon-16g [H for help] Hostname=ps42 Refresh= 2secs 19:59.17
Memory and Swap
PageSize:4KB RAM-Memory Swap-Space High-Memory Low-Memory
Total (MB) 15941.4 3906.0 - not in use - not in use
Free (MB) 7966.2 3906.0
Free Percent 50.0% 100.0%
Linux Kernel Internal Memory (MB)
Cached= 4304.7 Active= 3758.0
Buffers= 91.7 Swapcached= 0.0 Inactive = 3616.0
Dirty = 1.2 Writeback = 0.0 Mapped = 2479.8
Slab = 200.2 Commit_AS = 10984.3 PageTables= 73.1
```

```
nmon-12f [H for help] Hostname=si2srv01 Refresh= 2secs 10:58.07
Memory Stats
RAM High Low Swap
Total MB 749.2 0.0 749.2 150.0
Free MB 201.9 0.0 201.9 150.0
Free Percent 26.9% 0.0% 26.9% 100.0%
MB MB MB MB
Cached= 178.5 Active= 396.7
Buffers= 24.1 Swapcached= 0.0 Inactive = 127.3
Dirty = 0.0 Writeback = 0.0 Mapped = 26.5
Slab = 14.2 Commit_AS = 1039.3 PageTables= 1.6
```

```
nmon-12f [H for help] Hostname=si2srv02 Refresh= 2secs 10:58.25
Memory Stats
RAM High Low Swap
Total MB 749.2 0.0 749.2 150.0
Free MB 227.4 0.0 227.4 150.0
Free Percent 30.4% 0.0% 30.4% 100.0%
MB MB MB MB
Cached= 159.5 Active= 386.1
Buffers= 17.3 Swapcached= 0.0 Inactive = 113.3
Dirty = 0.0 Writeback = 0.0 Mapped = 22.6
Slab = 13.6 Commit_AS = 960.4 PageTables= 1.4
```

Los resultados son prácticamente con ambos comandos. Las diferencias pueden deberse a varios factores: modificación del estado de la RAM al ejecutarse diferentes comandos, errores de aproximación o cambios en las aplicaciones que se estaban ejecutando (como Discord, aplicación de *meetings* online para la comunicación telemática, o el navegador web).

Visto esto, vamos a realizar un pago de "calentamiento" con cada uno de los métodos especificados en el enunciado. Los resultados se pueden ver a continuación:
Pago de calentamiento con la aplicación P1-base:

←

→

↺

🏠

🔒 10.1.7.2:8080/P1-base/testbd.jsp

Pago con tarjeta

Proceso de un pago

Id Transacción:

1

Id Comercio:

10

Importe:

100

Numero de visa:

1111 2222 3333 4444

Titular:

Jose Garcia

Fecha Emisión:

11/09

Fecha Caducidad:

11/20

CVV2:

123

Modo debug:

True

False

Direct Connection:

True

False

Use Prepared:

True

False

Pagar

Pago de calentamiento con la aplicación P1-ws-cliente:

←

→

↺

🏠

🔒 10.1.7.2:8080/P1-ws-cliente/testbd.jsp

Pago con tarjeta

Proceso de un pago

Id Transacción:

2

Id Comercio:

10

Importe:

200

Numero de visa:

1111 2222 3333 4444

Titular:

Jose Garcia

Fecha Emisión:

11/09

Fecha Caducidad:

11/20

CVV2:

123

Modo debug:

☐ True ☐ False

Direct Connection:

☐ True ☐ False

Use Prepared:

☐ True ☐ False

Pagar

Pago de calentamiento con la aplicación P1-ejb-cliente-remoto:

←

→

↺

🏠

🔒 10.1.7.2:8080/P1-ejb-cliente-remoto/testbd.jsp

Pago con tarjeta

Proceso de un pago

Id Transacción:

3

Id Comercio:

10

Importe:

300

Numero de visa:

1111 2222 3333 4444

Titular:

Jose Garcia

Fecha Emisión:

11/09

Fecha Caducidad:

11/20

CVV2:

123

Modo debug:

☐ True ☐ False

Direct Connection:

☐ True ☐ False

Use Prepared:

☐ True ☐ False

Pagar

Finalmente, la comprobación de que los tres pagos se han realizado satisfactoriamente:

←

→

↺

🏠

🔒 10.1.7.2:8080/P1-base/getpagos

Pago con tarjeta

Lista de pagos del comercio 10

idTransaccion	Importe	codRespuesta	idAutorizacion
1	100.0	000	1
2	200.0	000	2
3	300.0	000	3

[Volver al comercio](#)

Prácticas de Sistemas Informáticos II

Se puede comprobar que realmente estamos realizando los pagos con las aplicaciones mencionadas si nos fijamos en la URL del navegador, donde se encuentra mencionada la aplicación en uso.

3 Ejercicio 3

Enunciado:

Ejercicio 3: Ejecute el plan completo de pruebas sobre las 3 versiones de la práctica, empleando el esquema de despliegue descrito anteriormente. **Realice la prueba tantas veces como necesite para eliminar ruido relacionado con procesos periódicos del sistema operativo, lentitud de la red u otros elementos.**

- Compruebe que efectivamente se han realizado todos los pagos. Es decir, la siguiente consulta deberá devolver “3000”:
SELECT COUNT(*) FROM PAGO;
- Compruebe que ninguna de las peticiones ha producido un error. Para ello revise que la columna %Error indique 0% en todos los casos.

Una vez que los resultados han sido satisfactorios:

- Anote los resultados del informe agregado en la memoria de la práctica.
- Salve el fichero server.log que se encuentra en la ruta glassfish/domains/domain1/logs de Glassfish y adjúntelo con la práctica.
- Añada a la memoria de prácticas la siguiente información: ¿Cuál de los resultados le parece el mejor? ¿Por qué? ¿Qué columna o columnas elegiría para decidir este resultado?

Incluir el directorio P2 en la entrega.

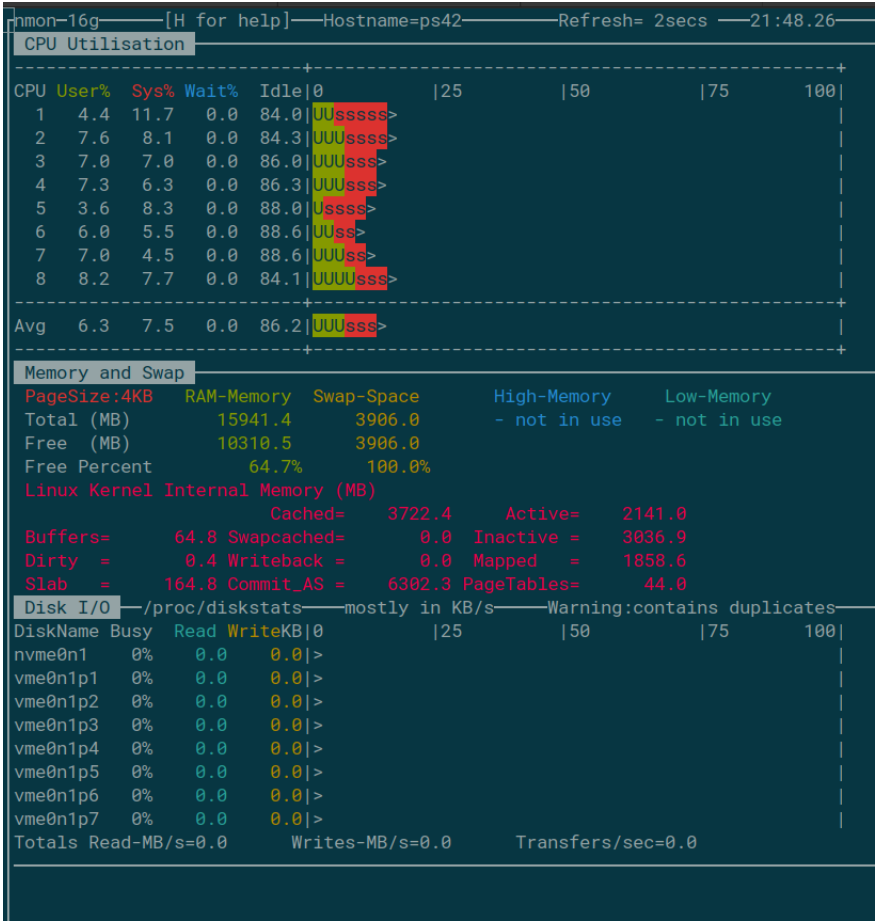
Repita la prueba de P1-ejb (inhabilite los ‘Thread Group’ P1-base y P1-ws) con el **EJB local** incluido en P1-ejb-servidor-remoto. Para ello, cambie su ‘HTTP Request’, estableciendo su ‘Server Name or IP’ a 10.X.Y.1 (VM1) y su ‘Path’ a ‘P1-ejb-cliente/procesapago’. **Compare los resultados obtenidos con los anteriores.**

El fichero P2.jmx entregado no debe contener estos cambios, es decir, debe estar configurado para probar el EJB remoto.

Respuesta a la cuestión:

Este ejercicio consiste en la **ejecución del plan completo de pruebas** sobre las **3 versiones** de la práctica utilizando JMeter y todo lo realizado (configuración del plan de prueba) en el primer ejercicio.

Antes de nada, se nos advierte que para que las pruebas se ejecuten correctamente, la base de datos debe estar totalmente vacía al inicio, así como debemos asegurarnos de que el uso de la CPU y memoria RAM sean prácticamente nulos, algo que podemos comprobar con **nmon**:



Se puede comprobar que el uso de la CPU por parte del usuario es bajo (6%) y que la memoria RAM está relativamente vacía, entorno al 65% (recordar las aplicaciones que teníamos que tener abiertas).

Dicho esto ejecutamos el plan de pruebas, obteniendo los siguientes resultados:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Etiqueta	# Muestras	Media	Mediana	90% Line	95% Line	99% Line	Mín	Máx	% Error	Rendimiento	Kb/sec	Sent KB/sec
2	P1-base	1000	10	7	22	30	57	3	62	0,000%	92,83327	119,37	0,00
3	P1-ws	1000	30	27	42	53	81	22	452	0,000%	32,43383	42,09	0,00
4	P1-ejb	1000	11	10	17	21	41	7	86	0,000%	84,32414	64,90	0,00
5	Total	3000	17	12	30	41	64	3	452	0,000%	56,10938	62,71	0,00
6	P1-base	1000	5	4	9	11	23	3	61	0,000%	176,24251	226,62	0,00
7	P1-ws	1000	31	27	41	53	87	24	586	0,000%	31,93562	41,44	0,00
8	P1-ejb	1000	10	9	16	20	43	6	67	0,000%	90,66183	69,77	0,00
9	Total	3000	15	9	29	37	62	3	586	0,000%	62,47007	69,82	0,00
10	P1-base	1000	7	6	13	16	25	3	37	0,000%	132,46788	170,33	0,00
11	P1-ws	1000	40	30	71	88	115	24	147	0,000%	24,88986	32,30	0,00
12	P1-ejb	1000	23	18	47	61	87	7	98	0,000%	41,66493	32,07	0,00
13	Total	3000	23	19	49	68	97	3	147	0,000%	41,82350	46,75	0,00
14	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----
15	Etiqueta	# Muestras	Media	Mediana	90% Line	95% Line	99% Line	Mín	Máx	% Error	Rendimiento	Kb/sec	Sent KB/sec
16	P1-ejb	1000	8	5	11	16	34	3	1695	0,000%	118,04982	153,41	0,00
17	Total	1000	8	5	11	16	34	3	1695	0,000%	118,04982	153,41	0,00
18	P1-ejb	1000	5	4	10	14	23	2	50	0,000%	175,19271	227,67	0,00
19	Total	1000	5	4	10	14	23	2	50	0,000%	175,19271	227,67	0,00
20	P1-ejb	1000	6	5	10	13	24	2	33	0,000%	161,08247	209,33	0,00
21	Total	1000	6	5	10	13	24	2	33	0,000%	161,08247	209,33	0,00

En la imagen se puede observar que se han ejecutado tres veces las pruebas de las aplicaciones P1-base, P1-ws y P1-ejb (remoto) y, separado por una fila de guiones, tres veces las pruebas de la P1-ejb (local), con el objetivo de realizar una mejor comparación de los resultados y que no haya dudas entre simulaciones.

En primer lugar, fijarse que el porcentaje de **error** es **nulo** en todas las pruebas, por lo que queda claro que se han ejecutado satisfactoriamente. Ahora, centrándonos en los resultados, prestamos atención a la columna **”Rendimiento”** que es la que al fin y al cabo califica la calidad de la aplicación en cuanto a prestaciones. En todas las pruebas vemos que la aplicación **P1-base** es la que muestra una mejor *performance*, seguido de la P1-ejb y finalmente la P1-ws. Estos resultados son ratificados si nos fijamos en la columna **”Media”**, que indica el tiempo medio de respuesta en milisegundos, por lo que a menor tiempo medio de respuesta mayor rendimiento y la clasificación anterior se mantiene.

Adicionalmente, comparamos los resultados anteriores con los de las pruebas de la P1-ejb (local). El rendimiento es muy **parecido** a la aplicación P1-base (la mejor de las anteriores pruebas), pero podemos sostener que, de media, el rendimiento de la **P1-ejb local** es **lig-eramente mejor** que la P1-base. Esto también se cumple si miramos los valores medios de la columna **”Media”**, como hemos hecho anteriormente.

Finalmente nos podemos preguntar que qué tienen las aplicaciones **P1-base y la P1-ejb local** para ser claramente las que mejor rendimiento muestran. Esta respuesta no es fácil, ya que depende de muchos factores como puede ser la máquina en que se han ejecutado las pruebas (nuestro ordenador personal en este caso) o por la propia arquitectura de la aplicación. No obstante, decir que la **P1-base y P1-ejb local** son implementadas con **métodos y técnicas más eficientes** no es nada descabellado, y es algo que deberemos tener en cuenta en el futuro si queremos crear un **servicio de calidad**.

4 Ejercicio 4

Enunciado:

Ejercicio 4: Adaptar la configuración del servidor de aplicaciones a los valores indicados. Guardar, como referencia, la configuración resultante, contenida en el archivo de configuración localizado en la máquina virtual en `$opt/glassfish4/glassfish/domains/domain1/config/domain.xml`³. Para obtener la versión correcta de este archivo es necesario detener el servidor de aplicaciones. **Incluir este fichero en el entregable de la práctica. Se puede copiar al PC del laboratorio con scp.**

Revisar el script `si2-monitor.sh` e indicar los mandatos `asadmin`⁴ que debemos ejecutar en el Host PC1 para averiguar los valores siguientes, mencionados en el Apéndice 1, del servidor PC1VM1:

1. **Max Queue Size del Servicio HTTP**
2. **Maximum Pool Size del Pool de conexiones a nuestra DB**

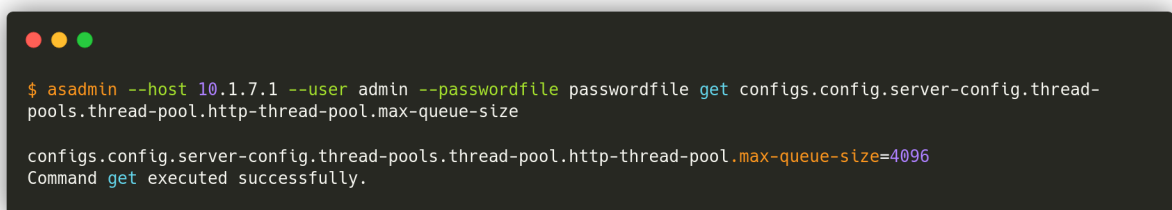
Así como el mandato para monitorizar el número de errores en las peticiones al servidor web.

Respuesta a la cuestión:

En este ejercicio deberemos adaptar la configuración del servidor a los valores indicados y guardar, como referencia, la configuración resultante, contenida en el archivo de configuración localizado en la máquina virtual en:

`$opt/glassfish4/glassfish/domains/domain1/config/domain.xml`.

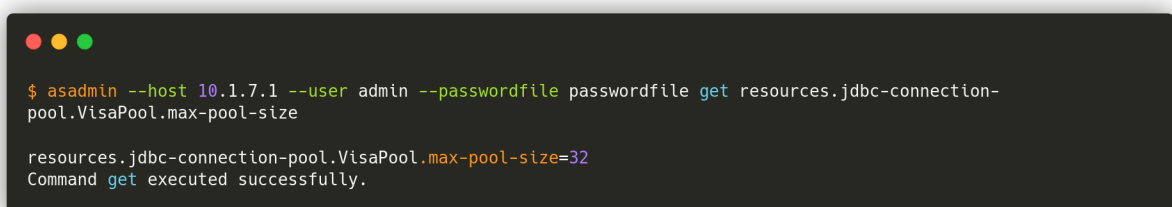
Adicionalmente deberemos revisar el script `si2-monitor.sh` e indicar los mandatos `asadmin` que debemos ejecutar en el Host PC1 para averiguar los valores de **Max Queue Size del Servicio HTTP** y **Maximum Pool Size del Pool de conexiones a nuestra DB**. Las siguientes imágenes muestran el comando ejecutado y los valores obtenidos.



```
$ asadmin --host 10.1.7.1 --user admin --passwordfile passwordfile get configs.config.server-config.thread-pools.thread-pool.http-thread-pool.max-queue-size

configs.config.server-config.thread-pools.thread-pool.http-thread-pool.max-queue-size=4096
Command get executed successfully.
```

Como podemos ver en el resultado, el valor de **Max Queue Size del Servicio HTTP** es 4096.

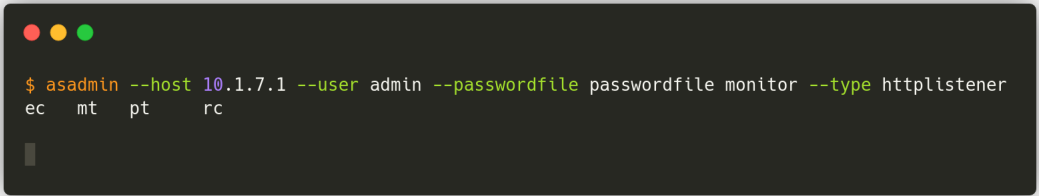


```
$ asadmin --host 10.1.7.1 --user admin --passwordfile passwordfile get resources.jdbc-connection-pool.VisaPool.max-pool-size

resources.jdbc-connection-pool.VisaPool.max-pool-size=32
Command get executed successfully.
```

Finalmente, el comando anterior nos da el valor de **Maximum Pool Size del Pool de conexiones a nuestra DB**, que es 32.

Finalmente el mandato para monitorizar el número de errores en las peticiones al servidor web es:



El resultado de dicho comando son cuatro columnas:

- **ec**: Contador de errores.
- **mt**: Tiempo máximo. Es el tiempo máximo de una respuesta.
- **pt**: Tiempo de proceso. Valor acumulativo de tiempo requerido para contestar cada solicitud.
- **rc**: Contador de solicitudes. Valor acumulativo del número de solicitudes.

En la imagen todas las columnas están vacías, es decir, 0 todos los valores, debido a que no se ha capturado tráfico durante la ejecución (no se requería).

5 Ejercicio 5

Enunciado:

Registrar en la hoja de cálculo de resultados los valores de configuración que tienen estos parámetros.

Respuesta a la cuestión:

Conseguimos todos los valores de los parámetros de configuración usando la consola de administración en las rutas especificadas en el enunciado antes de este ejercicio. Los valores obtenidos se pueden ver en la hoja de cálculo aportada **SI2-P2-curvaProductividad.ods** y en la siguiente imagen:

Parámetros de configuración		
Elemento	Parámetro	Valor
JVM Settings	Heap Máx. (MB)	512
JVM Settings	Heap Mín. (MB)	512
HTTP Service	Max.Thread Count	5
HTTP Service	Queue size	4096
Web Container	Max.Sessions	-1
Visa Pool	Max.Pool Size	32

6 Ejercicio 6

Enunciado:

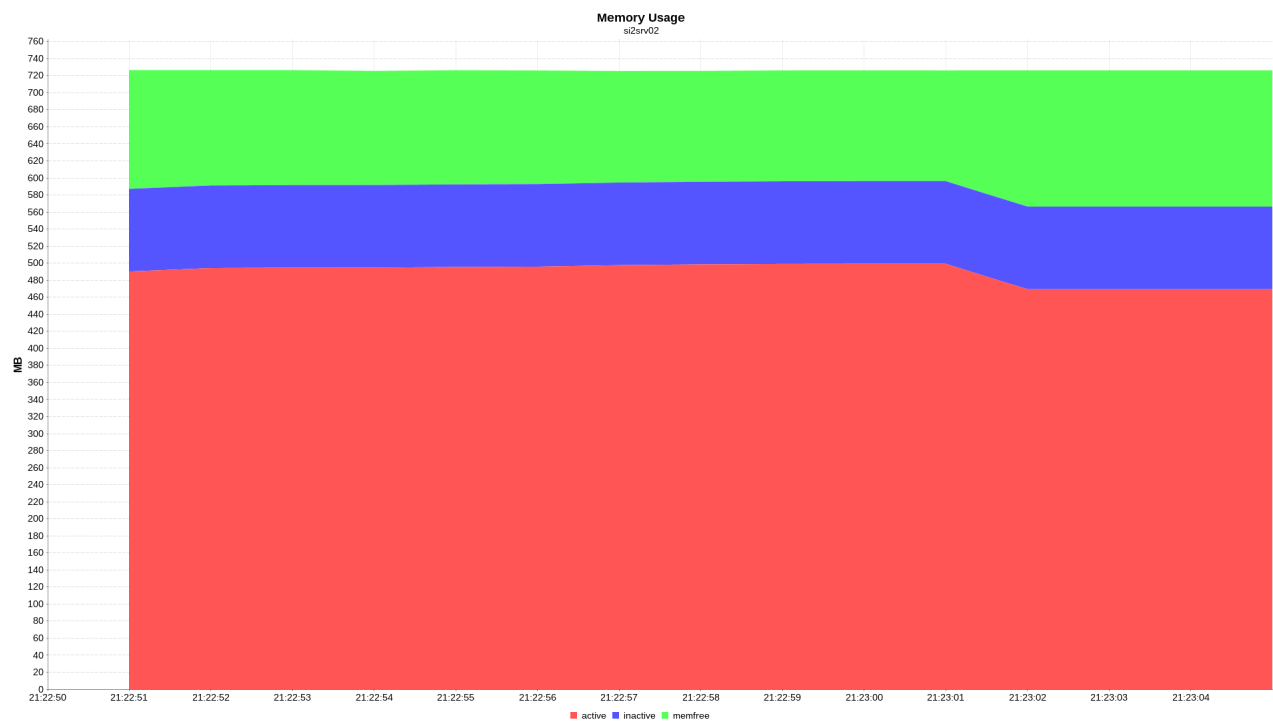
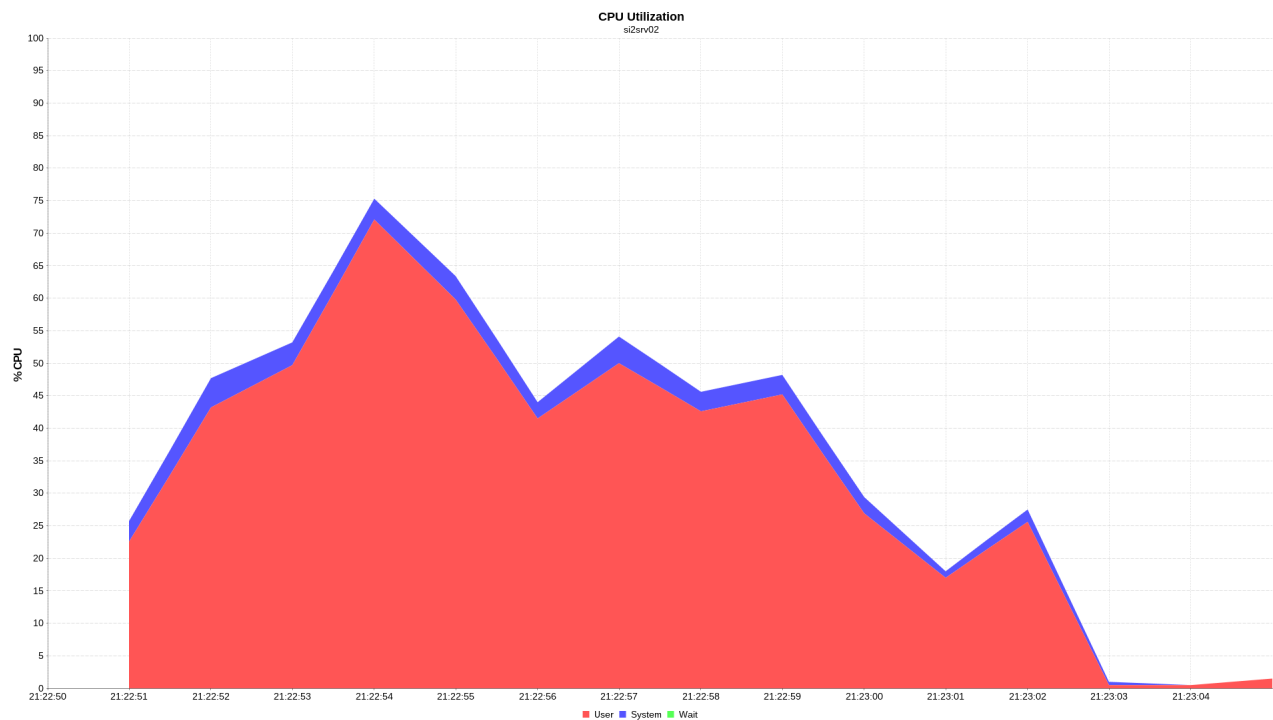
Ejercicio 6: Tras habilitar la monitorización en el servidor, repita la ejecución del plan de pruebas anterior. Durante la prueba, vigile cada uno de los elementos de monitorización descritos hasta ahora. Responda a las siguientes cuestiones:

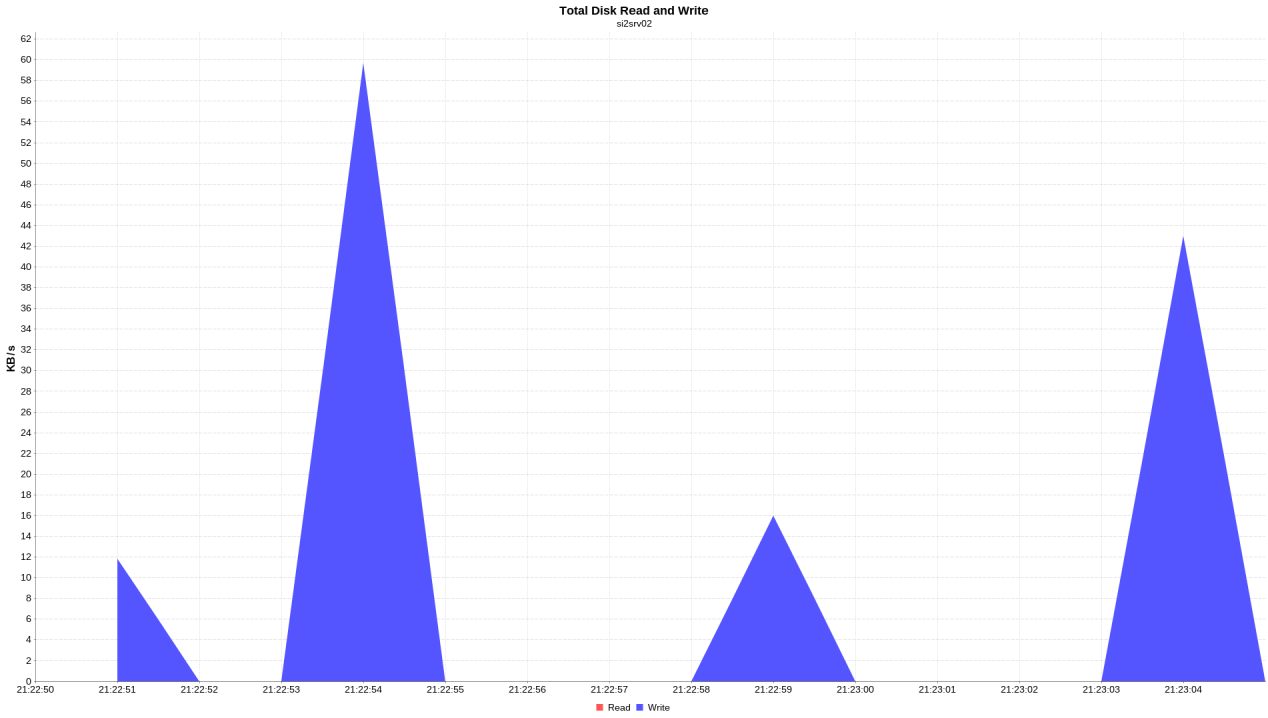
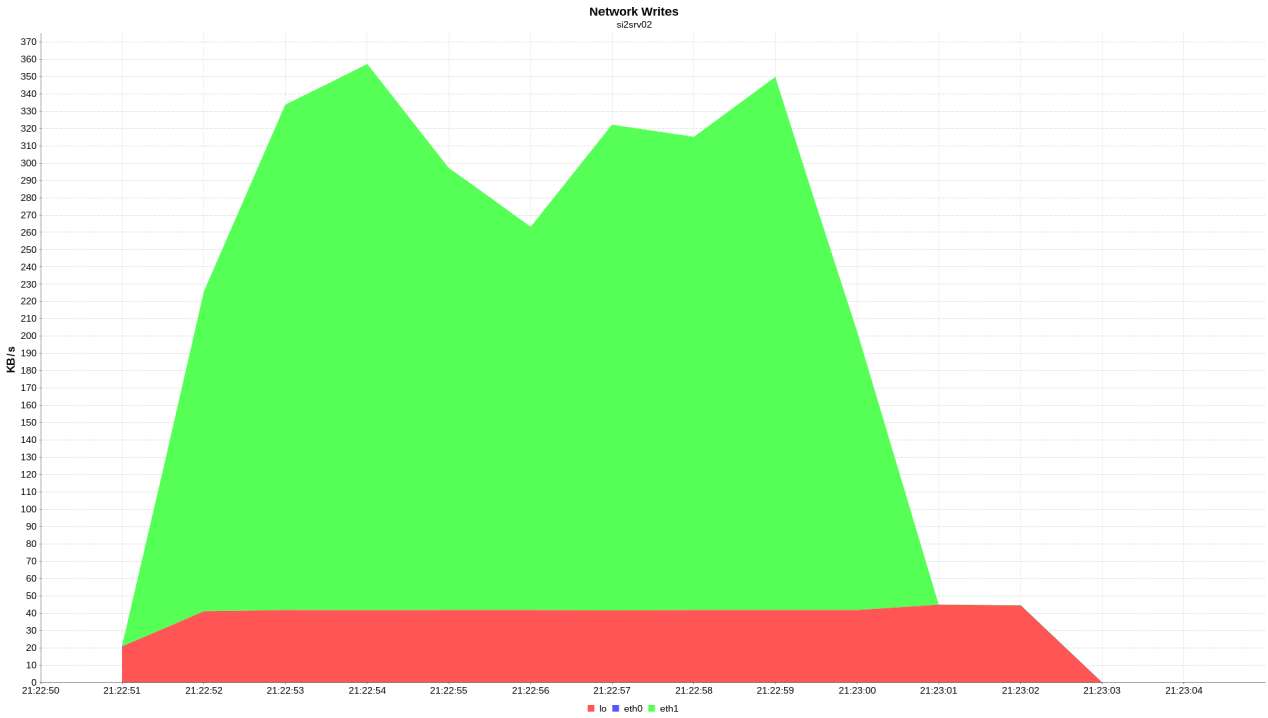
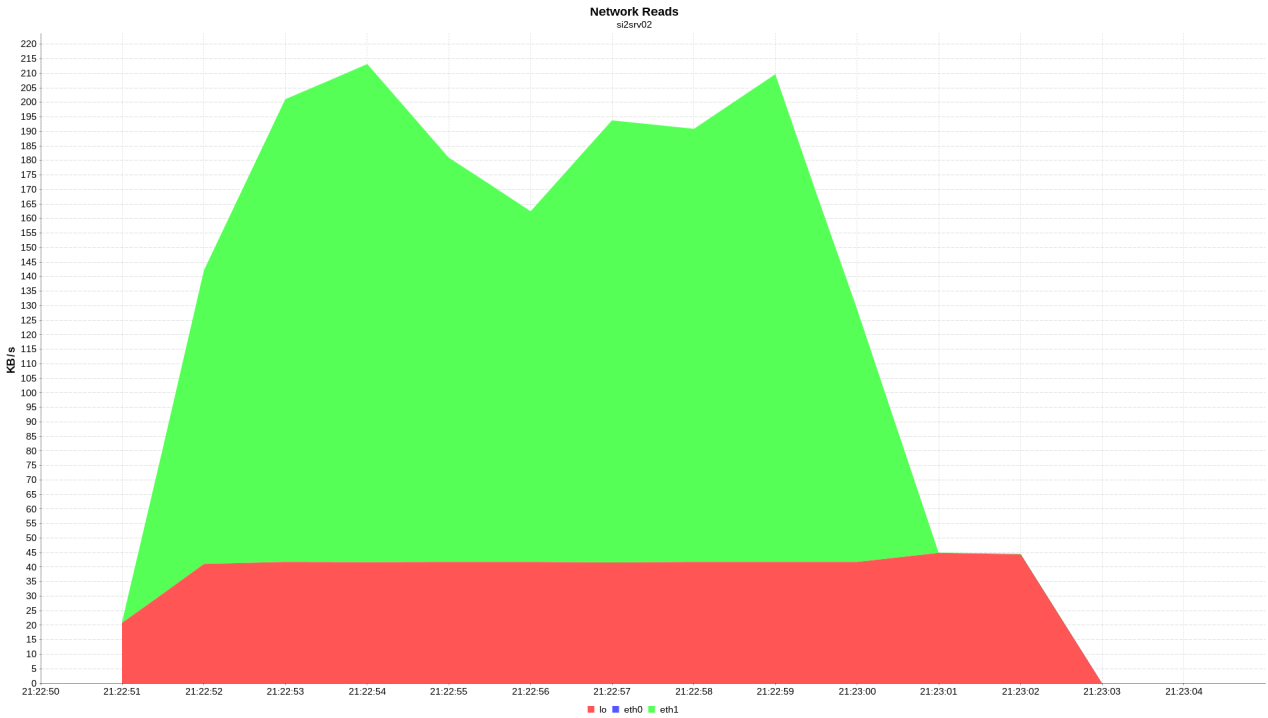
- A la vista de los resultados, ¿qué elemento de proceso le parece más costoso? ¿Red? ¿CPU? ¿Acceso a datos? En otras palabras, ¿cuál fue el elemento más utilizado durante la monitorización con *nmon* en un entorno virtual? (CPU, Memoria, disco ...)
- ¿Le parece una situación realista la simulada en este ejercicio? ¿Por qué?
- Teniendo en cuenta cuál ha sido el elemento más saturado, proponga otro esquema de despliegue que resuelva esa situación.
-

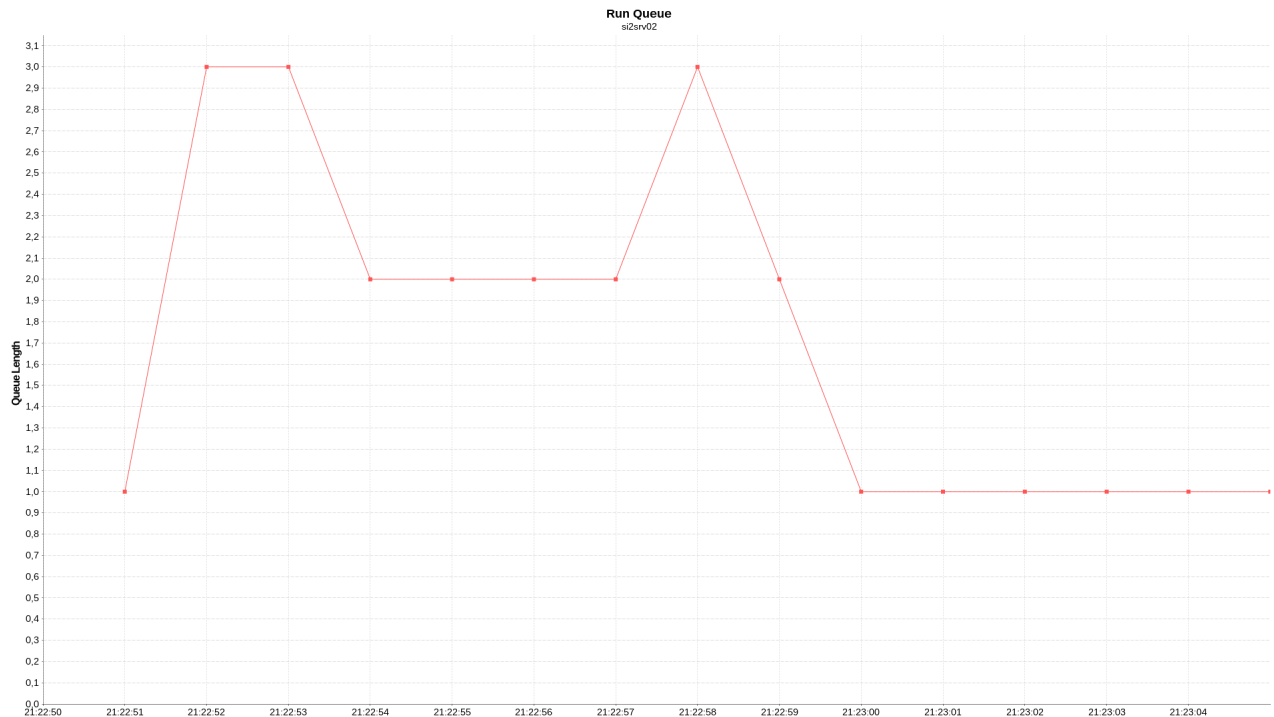
Nota: Las respuestas a estas cuestiones deben estar acompañadas por datos que respalden la argumentación, en forma de pantallazos de *nmon* (o gráficas de *Nmon Visualizer*) y pantallazos de *si2-monitor-sh*.

Respuesta a la cuestión:

Para dar una respuesta en condiciones en este ejercicio nos basaremos en las gráficas aportadas por **NMON Visualizer**, las cuales se exponen a continuación. Comentar que aunque no se indique explícitamente, creemos que los elementos a analizar son los elementos del servidor, ya que es el objeto de atención cuando tratamos de encontrar los cuellos de botella de nuestro servicio.







La **primera gráfica** se corresponde con el **uso de la CPU**, donde podemos ver que existen picos de alto rendimiento cuando se comienzan a ejecutar las pruebas, algo a tener en cuenta para el veredicto final. La **segunda imagen** representa el **uso de memoria** durante las pruebas, y si nos fijamos podemos ver que aumenta su uso ligeramente al principio de las pruebas y baja de nuevo al final de las mismas. En la **tercera gráfica** tenemos las **lecturas de red del servidor**, y en la **cuarta** las **escrituras de red** del mismo. Estas dos últimas gráficas están resumidas en una, la **quinta**, que muestra las **lecturas y escrituras totales del disco**. Finalmente, en el **sexto gráfico**, tenemos el **número de peticiones en cola** en función del tiempo.

Con esto en mente, podemos ahora debatir que elemento es el más utilizado durante la monotorización. A primera vista parece que el uso de memoria es bastante alto, ya que se trabaja con el 67% de la misma, pero esto es una realidad un tanto falsa, ya que si nos fijamos bien, el aumento (aunque se vea al inicio un poco cortado debido a que hubo un *delay* entre el inicio de las pruebas y el inicio de la monotorización) de uso de la memoria, así como el descenso del uso de la misma al final de las pruebas fue tan solo de 40-50 MB, algo despreciable para la época actual. Por el contrario, sí existe un gran cambio de rendimiento en el uso de la CPU, que aumenta desde un 25% de utilización hasta un pico de 75% con una media de 65%, decayendo hasta menos del 5% al finalizar las pruebas. Dicho esto, está claro que el uso de la CPU es un factor a tener muy en cuenta.

Adicionalmente, las gráficas de lecturas/escrituras en disco no son muy significativas, ya que sus unidades están expresadas en KB/s algo que es muy situacional del equipo. Apostaríamos que hoy en día unas lecturas/escrituras del orden de 300 KB/s no es nada significativo como para considerarlo un cuello de botella.

Finalmente, el número medio de peticiones en cola alcanza un pico de 3, lo que indica que tampoco es un factor de saturación inminente cuando tenemos una *pool* de hilos de 5.

Por todo lo comentado anteriormente, sostenemos que el elemento más utilizado es la **CPU** y en la que habría que focalizar esfuerzos si quisiéramos mejorar el rendimiento de nuestro servicio.

Por otro lado y contestando a la segunda pregunta, el esquema utilizado no es para nada realista. En primer lugar, en las circunstancias actuales (COVID-19) nos vemos forzados a ejecutar todo en un mismo ordenador con dos máquinas virtuales distintas, algo que ya no parece indicar una simulación muy realista. Además, la aplicación P1-base que es la que se monotoriza durante este ejercicio, tiene integrado el proceso cliente como el proceso servidor en la misma máquina, lo que acaba por afirmar la poca realidad del asunto.

Finalmente, viendo que la CPU es el elemento más utilizado y junto al esquema de despliegue, pensamos que si utilizamos otra aplicación (como la P1-ejb) donde podemos desplegar el cliente y el servidor por separado y en máquinas diferentes, las CPU's de ambas máquinas tendrán más recursos disponibles y no será tan notorio dicho cuello de botella.

7 Ejercicio 7

Enunciado:

Ejercicio 7: Preparar el *script* de JMeter para su ejecución en el entorno de pruebas. Cambiar la dirección destino del servidor para que acceda al host en el que se encuentra el servidor de aplicaciones. Crear también el directorio `datagen` en el mismo directorio donde se encuentre el *script*, y copiar en él el archivo

`listado.csv`, ya que, de dicho archivo, al igual que en las prácticas anteriores, se obtienen los datos necesarios para simular el pago.

A continuación, realizar una ejecución del plan de pruebas con un único usuario, una única ejecución, y un *think time* bajo (entre 1 y 2 segundos) para verificar que el sistema funciona correctamente. Comprobar, mediante el *listener View Results Tree* que las peticiones se ejecutan correctamente, no se produce ningún tipo de error y los resultados que se obtienen son los adecuados.

Una vez comprobado que todo el proceso funciona correctamente, desactivar dicho *listener* del plan de pruebas para que no aumente la carga de proceso de JMeter durante el resto de la prueba.

Este ejercicio no genera información en la memoria de la práctica, realízelo únicamente para garantizar que la siguiente prueba va a funcionar.

Respuesta a la cuestión:

Tal y como se indica en el enunciado, este ejercicio no produce una salida en este informe. No obstante se puede comprobar la realización del mismo en los ficheros fuente entregados y en la ejecución de los próximos ejercicios.

8 Ejercicio 8

Enunciado:

Ejercicio 8: Obtener la curva de productividad, siguiendo los pasos que se detallan a continuación:

- Previamente a la ejecución de la prueba se lanzará una ejecución del *script* de pruebas (unas 10 ejecuciones de un único usuario) de la que no se tomarán resultados, para iniciar el sistema y preparar medidas consistentes a lo largo de todo al proceso.
Borrar los resultados de la ejecución anterior. En la barra de acción de JMeter, seleccionar Run -> Clear All.
- Borrar los datos de pagos en la base de datos VISA.
- Ejecutar la herramienta de monitorización *nmon* en ambas máquinas, preferiblemente en modo "Data-collect" (Ver 8.2.2).
- Seleccionar el número de usuarios para la prueba en JMeter (parámetro **C** de la prueba)
- Conmutar en JMeter a la pantalla de presentación de resultados, *Aggregate Report*.
- Ejecutar la prueba. En la barra de acción de JMeter, seleccionar Run -> Start.
- Ejecutar el programa de monitorización *si2-monitor.sh*
 - Arrancarlo cuando haya pasado el tiempo definido como rampa de subida de usuarios en JMeter (el tiempo de ejecución en JMeter se puede ver en la esquina superior derecha de la pantalla).
 - Detenerlo cuando esté a punto de terminar la ejecución de la prueba. Este momento se puede detectar observando cuando el número de hilos concurrentes en JMeter (visible en la esquina superior derecha) comienza a disminuir (su máximo valor es C).
 - Registrar los resultados que proporciona la monitorización en la hoja de cálculo.
- Durante el periodo de monitorización anterior, vigilar que los recursos del servidor *si2srv02* y del ordenador que se emplea para realizar la prueba no se saturen. En caso de usar *nmon* de forma interactiva, se deben tomar varios pantallazos del estado de la CPU durante la prueba, para volcar en la hoja de cálculo del dato de uso medio de la CPU (*CPU average %*). En caso de usar *nmon* en modo "Data-collect", esta información se puede ver posteriormente en *NMonVisualizer*. Una tercera opción (recomendada) es ejecutar el comando *vmstat* en una terminal remota a la máquina *si2srv02*, para extraer directamente el valor de uso medio de su CPU ⁵.
- Finalizada la prueba, salvar el resultado de la ejecución del *Aggregate Report* en un archivo, y registrar en la hoja de cálculo de resultados los valores *Average*, *90% line* y *Throughput* para las siguientes peticiones:
 - *ProcesaPago*.
 - *Total*.

Una vez realizadas las iteraciones necesarias para alcanzar la saturación, representar la curva de *Throughput* versus usuarios. Incluir el fichero P2-curvaProductividad.jmx en la entrega.

Nota: Todos los datos de monitorización que se entreguen como parte de la curva de rendimiento (derivados de JMeter, *nmon* y *si2-monitor.sh*) deben estar respaldados por pruebas que demuestren su veracidad, ya sea en la propia memoria o en ficheros adicionales (recomendado). En el caso de los recursos del sistema, se pueden mostrar tanto pantallazos del modo interactivo de *nmon* como gráficas de *NMONvisualizer*.

Respuesta a la cuestión:

Este ejercicio no genera un gran contenido en esta memoria, solo nos sirve de guía para obtener ciertos resultados, los cuales se describen a continuación. Adicionalmente, se añadirán más respuestas y explicaciones en el ejercicio 9.

Las pruebas se aportan en un directorio llamado *data-ex-8*, que tiene un subdirectorio por cada prueba realizada con nombre *<num-usuarios>u*. Dentro de estos directorios se encuentran los archivos que sirven como prueba a los datos:

- *cpu-host.txt* - Salida de *vmstat* en el host.
- *cpu-vm.txt* - Salida de *vmstat* en la máquina virtual.
- *monitor.txt* - Salida del comando *si2-monitor.sh*.
- *test-results.jtl* - Salida de la ejecución de *jmeter* en modo *no-gui*.
- *aggregate.csv* - Tabla en formato csv de *Aggregate Report* al importar el fichero *test-results.jtl* en *jmeter*.

9 Ejercicio 9

Enunciado:

Ejercicio 9: Responda a las siguientes cuestiones:

- A partir de la curva obtenida, determinar para cuántos usuarios conectados se produce el punto de saturación, cuál es el *throughput* que se alcanza en ese punto, y cuál el *throughput* máximo que se obtiene en zona de saturación.
- Analizando los valores de monitorización que se han ido obteniendo durante la elaboración de la curva, sugerir el parámetro del servidor de aplicaciones que se cambiaría para obtener el punto de saturación en un número mayor de usuarios.
- Realizar el ajuste correspondiente en el servidor de aplicaciones, reiniciarlo y tomar una nueva muestra cercana al punto de saturación. ¿Ha mejorado el rendimiento del sistema? Documente en la memoria de prácticas el cambio realizado y la mejora obtenida.

Respuesta a la cuestión:

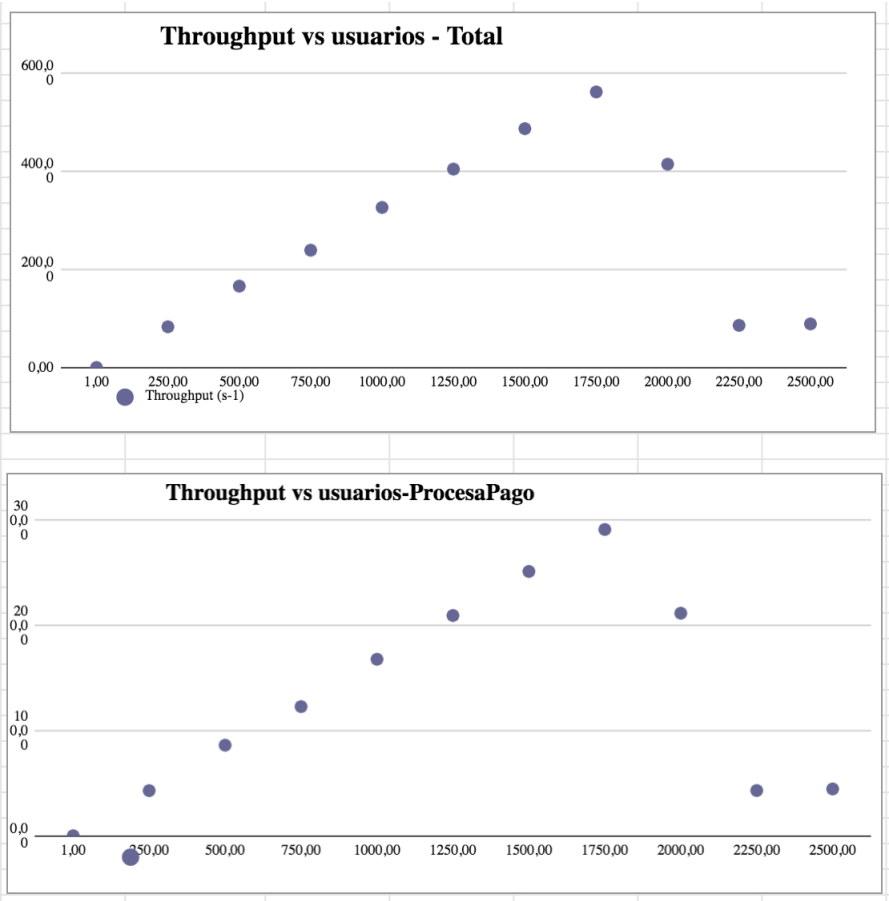
En este ejercicio contestaremos a las preguntas del enunciado, relacionadas con las pruebas ejecutadas siguiendo los pasos del ejercicio 8.

En la primera cuestión se nos pregunta por la cantidad máxima de usuarios antes de alcanzar la saturación, el *throughput* en ese punto y el *throughput* máximo que se alcanza en la zona de saturación. En nuestro ordenador personal obtenemos que entre 1750 y 2000 usuarios entramos en la zona de saturación. Podemos decir, sin mucho error, que con **1800 usuarios entramos en zona de saturación**, donde **alcanzamos un *throughput*** de, aproximadamente, **570 s⁻¹**. Adicionalmente, vemos en la simulación que este valor **puede ser el *throughput* máximo**. Desconocemos los valores exactos de *throughput* entre 1750 y 2000 usuarios, pero el **valor máximo oscila ligeramente en ese rango**.

Los resultados del plan de pruebas se recogen en la hoja de cálculo aportada inicialmente por el equipo docente, pero a continuación se muestra una imagen por si resultara ser de ayuda.

Prueba de rendimiento										
	Sistema	Monitores			Total			ProcesaPago		
Usuarios	CPU, average (%)	Visa Pool used Conns	HTTP Current Threads Busy	Conn queued, instant	Average (ms)	90% line (ms)	Throughput (s-1)	Average (ms)	90%line (ms)	Throughput (s-1)
1,00	23,21	0,00	0,00	0,00	160,00	58,00	0,42	140,00	58,00	0,23
250,00	25,95	0,08	0,23	0,00	9,00	22,00	83,20	13,00	28,00	43,10
500,00	30,05	0,23	0,85	0,13	8,00	17,00	165,90	10,00	22,00	86,30
750,00	35,25	0,34	1,18	2,11	117,00	29,00	239,00	123,00	32,00	123,00
1000,00	38,38	0,87	2,48	0,80	11,00	25,00	326,00	14,00	29,00	168,00
1250,00	41,08	1,91	2,84	4,47	18,00	29,00	404,00	21,00	33,00	209,60
1500,00	43,80	2,28	3,91	15,72	27,00	44,00	486,30	31,00	49,00	251,50
1750,00	49,10	2,96	4,20	34,27	51,00	128,00	561,00	55,00	132,00	291,40
2000,00	50,12	2,08	3,67	161,62	938,00	3039,00	414,00	898,00	2561,00	211,80
2250,00	60,61	0,26	3,60	238,47	14721,00	31569,00	86,10	14641,00	31595,00	43,20
2500,00	67,32	0,26	3,73	281,49	16823,00	31585,00	89,10	16645,00	31582,00	44,70

Con estos resultados podemos sacar, entre otras, las dos siguientes gráficas, que ratifican lo anteriormente comentado.



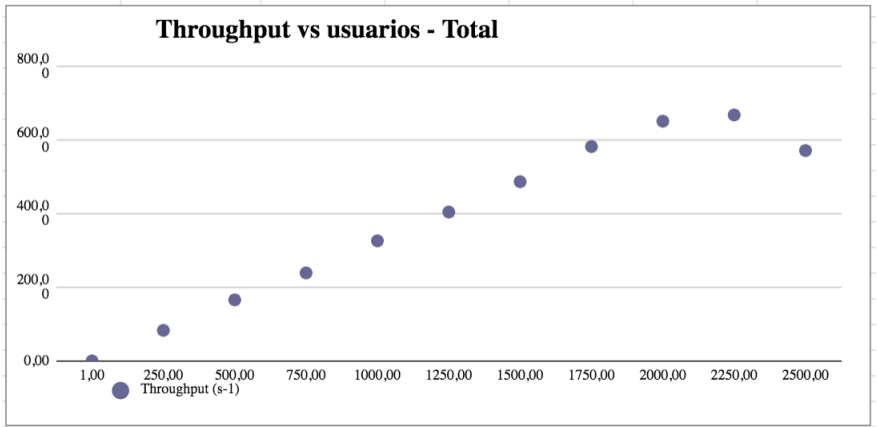
Se puede observar que a partir de los 2000 usuarios simultáneos el servidor colapsa (saturación) y comienzan a darse errores de conexión, hasta que la saturación es máxima a partir de donde no se pueden producir más conexiones debido a la caída del servicio del servidor.

En teoría, la gráfica resultante debería tener una parte lineal y otra parte de saturación. En las gráficas resultantes no es perceptible esta última zona, debido al aumento abrupto de errores de conexión.

A continuación se nos pregunta por el parámetro de configuración que deberíamos modificar si queremos alcanzar la saturación en un número mayor de usuarios. La respuesta es el **número de hilos de procesamiento del servidor** (tanto el número de hilos mínimo como el máximo). Actualmente estos valores los tenemos en 5 (tanto para el máximo como para el mínimo).

Modificando el valor de hilos mínimo a 10 y el valor de hilos máximo a 15, podemos probar a ejecutar las pruebas de nuevo y ver si mejora el rendimiento.

Efectivamente, el rendimiento mejora, ahora el punto de **saturación se alcanza sobre los 2100 usuarios simultáneos**, con un valor **aproximado de 650 s⁻¹** y un *throughput* **máximo de 667,28 s⁻¹** (alcanzado en un número de usuarios aproximado de 2250). Se muestra la imagen resultante de la gráfica total de la prueba tras las mejoras.



Es fácil ver que sobre los 2500 usuarios conectados empiezan a ocurrir fallos de conexión al servidor, indicando que se está entrando en colapso por sobrecarga.

III Conclusiones

El objetivo de esta práctica es aprender a medir el rendimiento de una aplicación JAVA EE, así como conocer las implicaciones que algunas decisiones de diseño o de arquitectura pueden tener en el rendimiento de la aplicación.

La entrega de los resultados de esta práctica se regirá por las normas expuestas durante la presentación de la asignatura. El incumplimiento de estas normas conllevará a considerar que la práctica no ha sido entregada en tiempo. Esto implica que se tendrá que volver a enviar correctamente y que se aplicará la penalización por retraso pertinente.

Contenido del fichero:

- El directorio P2, con el código resultante de todas las modificaciones sugeridas.
- Archivo JMX con el guion de pruebas creado por el alumno en la primera parte (P2.jmx)
- Archivo JMX con el guion de pruebas de productividad (P2-curvaProductividad.jmx).
- Archivo SI2-P2-curvaProductividad.ods con los datos recogidos y la curva de productividad.
- Archivo domain.xml(del dominio domain1) tras realizar las modificaciones sugeridas en la segunda parte de la práctica.
- Memoria de la ejecución de la práctica, en la que se comparen los valores de latencia (al percentil 90%) y throughput para las tres modalidades descritas en la primera parte, así como la respuesta a todas las cuestiones planteadas y los resultados experimentales de la segunda y tercera parte de la práctica.
- En caso de que no estén incluidos en la memoria (lo cual no se recomienda para que quede más limpia), los datos de los resultados experimentales de la prueba de determinación de la curva de productividad (pantallazos o logs de nmon, salidas de si2-monitor, Aggregate Report de JMeter) deberán enviarse dentro de la entrega, preferiblemente organizados en carpetas según el número de usuarios para el que se esté haciendo la prueba (valor de C).

IV Bibliografía

- Java EE 7 Tutorial
- Apache JMeter
- Oracle, GlassFish Enterprise Server v3.1 PerformanceTuning Guide
- Oracle, GlassFish Enterprise Server v3 Administration Guide
- Oracle, Java 6 Virtual Machine Garbage Collection Tuning
- Documentación P2 Moodle