

 UNIVERSIDAD AUTÓNOMA DE MADRID	

**Escuela Politécnica Superior**  
**Ingeniería Informática**  
**Prácticas de Sistemas Informáticos 2**

Grupo	2401	Práctica	2	Fecha	24/03/2020
Alumno/a	Pelayo Alvaredo, Victoria				
Alumno/a	Benjumeda Rubio, Jose Manuel				

## Práctica 2: Rendimiento

### Ejercicio 1:

Para la realización e este ejercicio hemos seguido todos los pasos indicados en el enunciado de la práctica.

No hemos encontrado ningún problema, con la realización del ejercicio. Para comprobar su correcto funcionamiento, a la hora de la simulación hemos añadido el *árbol de resultados*, tal y como se explica en una anotación escrita en el pdf.

Se adjunta el fichero creado, P2.jmx, en los archivos de la práctica.

### Ejercicio 2:

Sobre este ejercicio, y el resto de la práctica, nos gustaría destacar, que debido a la situación actual, consecuencia del coronavirus, hemos tenido que realizar los ejercicios en un solo ordenador, a pesar de que en el enunciado se especifica que se realice en dos. Por lo que tenemos en nuestro despliegue ambas máquinas virtuales en el mismo ordenador, el resto del diagrama de despliegue se correspondería al indicado en el enunciado, figura 22.

En este ejercicio hemos desplegado las aplicaciones tal y como se especifica en el enunciado.

Primero hemos modificado el fichero *glassfish-web.xml* para indicar la ip que utilizará P1-ejb-cliente (**10.1.8.1**)

Para ello hemos tenido que modificar los ficheros *build.properties* y *postgresql.properties*:

- P1-base
  - o En *build.properties* hemos modificado los siguientes parámetros:
    - **as.host=10.1.8.2**
  - o En *postgrsql.properties*:
    - **db.host=10.1.8.1**
    - **db.client.host=10.1.8.1**
- P1-ejb-cliente-remoto:
  - o En *build.properties*:
    - **as.host=10.1.8.2**
  - o En *postgresql.properties*:
    - **db.host=10.1.8.1**
    - **db.client.host=10.1.8.1**
- P1-ejb-servidor-remoto:
  - o En *build.properties*:
    - **as.host.client=10.1.8.1**
    - **as.host.server=10.1.8.1**
  - o En *postgresql.properties*:
    - **db.client.host=10.1.8.1**

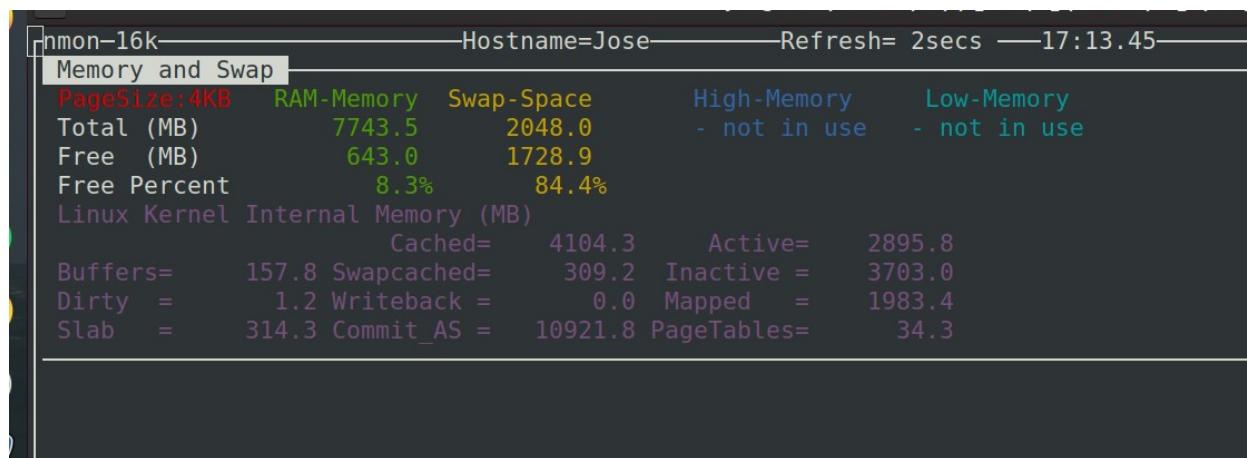
- **db.host=10.1.8.1**
- P1-ws
  - o En build.properties:
    - **as.host.client=10.1.8.2**
    - **as.host.server=10.1.8.1**
  - o En postgresql.properties:
    - **db.client.host=10.1.8.1**
    - **db.host=10.1.8.1**

En este ejercicio nos piden que adjuntemos capturas del comando *free* y *nmon*. Con el comando *free* obtenemos información cuanta memoria física y de intercambio(swap) tenemos libre y usada por el sistema, también por buffers y cachés. *Nmon* es una herramienta que nos permite monitorizar nuestro sistema operativo.

- Salida del comando free (PC)

```
Total time: 3 seconds
jose@Jose:~/Escritorio/Uni/4/2_cuatri/SI_II/Practicas/SI2_P2/P1-ejb-cliente-remoto$ free
              total usado libre compartido búfer/caché disponible
Memoria:    7929304     2746604     535812    1976512     4646888     2902176
Swap:        2097148     332676    1764472
jose@Jose:~/Escritorio/Uni/4/2_cuatri/SI_II/Practicas/SI2_P2/P1-ejb-cliente-remoto$
```

- Salida del comando nmon(PC)



- Salida del comando free (MV1)

```
si2@si2srv01:~$ free
              total      used      free      shared      buffers      cached
Mem:       767168     477588     289580          0      28880     172224
-/+ buffers/cache:  276484     490684
Swap:      153592          0     153592
si2@si2srv01:~$
```

- Salida del comando nmon(MV1)

```
jose@Jose: ~/Escritorio/Uni/4/2_cuatri/SI_II/Practicas/SI2_P2
nmon-12f——[H for help]——Hostname=si2srv01——Refresh= 2secs ——08:14.10
Memory Stats
      RAM     High     Low     Swap
Total MB    749.2     0.0    749.2    150.0
Free MB     281.4     0.0    281.4    150.0
Free Percent 37.6%   0.0%   37.6% 100.0%
      MB          MB          MB
      Cached= 165.0      Active= 329.1
Buffers= 38.9 Swapcached= 0.0 Inactive = 117.1
Dirty = 0.0 Writeback = 0.0 Mapped = 25.3
Slab = 13.4 Commit_AS = 926.3 PageTables= 1.5
```

- Salida del comando free(MV2)

```
si2@si2srv02:~$ free
      total        used        free      shared  buffers   cached
Mem: 767168      721932      45236          0  27912  176080
-/+ buffers/cache: 517940  249228
Swap: 153592          0  153592
si2@si2srv02:~$ █
```

- Salida del comando nmon(MV2)

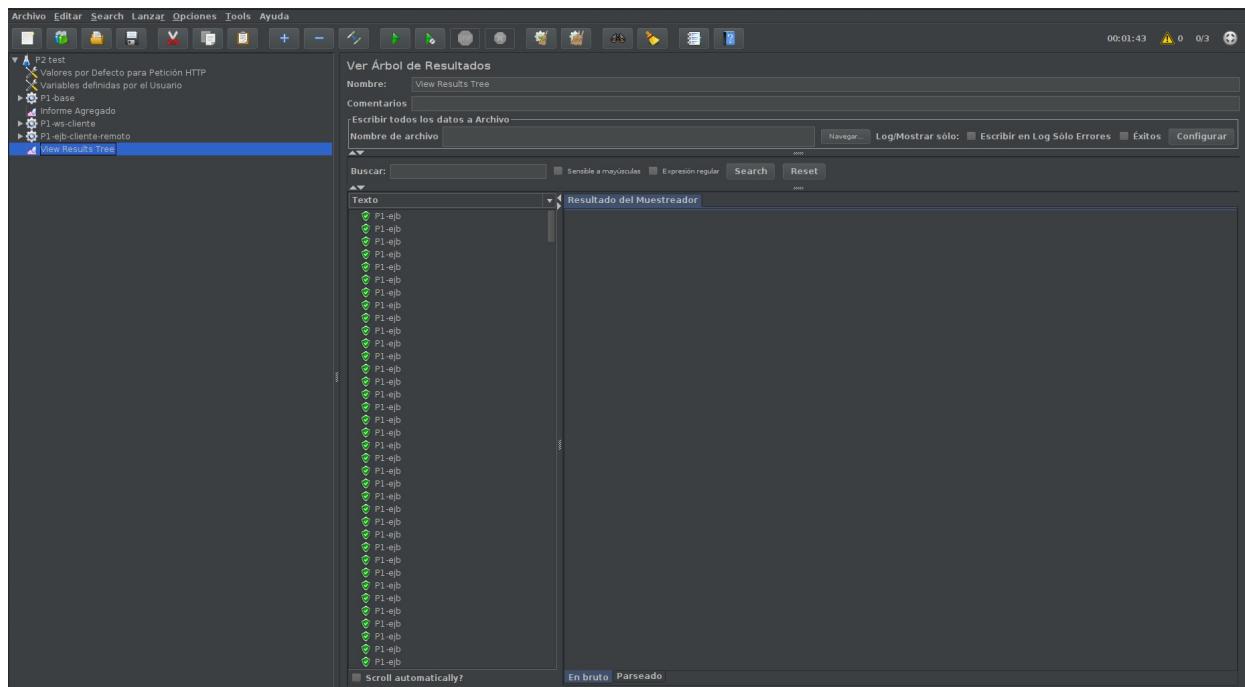
```
jose@Jose: ~/Escritorio/Uni/4/2_cuatri/SI_II/Practicas/SI2_P2
nmon-12f——[H for help]——Hostname=si2srv02——Refresh= 1secs ——08:14.27
Memory Stats
      RAM     High     Low     Swap
Total MB    749.2     0.0    749.2    150.0
Free MB     314.9     0.0    314.9    150.0
Free Percent 42.0%   0.0%   42.0% 100.0%
      MB          MB          MB
      Cached= 147.1      Active= 304.6
Buffers= 23.7 Swapcached= 0.0 Inactive = 108.7
Dirty = 0.0 Writeback = 0.0 Mapped = 25.1
Slab = 12.7 Commit_AS = 963.5 PageTables= 1.4
```

Con este nuevo despliegue, hemos ejecutado un pago con cada aplicación, y todos han funcionado como siempre. No adjuntamos capturas porque son las mismas que en las prácticas anteriores, y nos parece que recarga la memoria sin motivo.

### Ejercicio 3:

En este ejercicio nos piden que ejecutemos el plan de pruebas que hemos realizado anteriormente, es decir, P2.jmx.

En el árbol de resultados vemos algunas peticiones que se han ejecutado correctamente:



Para estar seguros de que los pagos se han ejecutado, realizamos la siguiente consulta en la máquina virtual donde está la base de datos, es decir, comprobamos que se han realizado un total de 3000 pagos:

Para esto hemos accedido a la base de datos con “psql visa -U alumnodbd”, y hemos ejecutado la query que se ve en la captura:

```
psql: warning: Falling back to the standard locale ("C").  
psql (8.4.10)  
Type "help" for help.  
  
visa=# select count(*) from pago;  
count  
-----  
 3000  
(1 row)  
  
visa=#
```

Hemos guardado el fichero server.log en la carpeta de la práctica tras la ejecución del plan de pruebas.

Éste es el informe agregado de las pruebas realizadas, donde vemos 0% de error en todas ellas:

Nombre de archivo													Navegar...	Log/Mostrar sólo:	Escribir en Log Sólo Errores	Exitos	Configurar
Etiqueta	# Muestras	Media	Mediana	90% Line	95% Line	99% Line	Mín	Máx	% Error	Rendimiento	Kb/sec	Sent KB/sec					
P1-base	1000	9	6	9	11	17	4	2584	0,00%	103,9/sec	133,28	0,00					
P1-ws	1000	49	42	58	64	81	36	3741	0,00%	20,0/sec	25,88	0,00					
P1-ejb	1000	14	13	19	21	27	9	925	0,00%	66,1/sec	86,20	0,00					
Total	3000	24	13	47	55	70	4	3741	0,00%	40,1/sec	51,89	0,00					

Los mejores resultados son los obtenidos para P1-base. Suponemos que esto se debe a que esta aplicación no tiene un cliente que haga de intermediario entre nuestro navegador y el servidor, sino que se accede directamente a las funciones del servidor con nuestro navegador actuando como cliente.

De esta manera no hay que gastar tiempo en enviar peticiones a un cliente y que luego ya éste las mande al servidor (y a la vuelta igual), sino que las peticiones de nuestro navegador llegan directamente al servidor.

Casi todas las columnas proporcionan información que indica que funciona mejor:

Columna “Media”: de media, las peticiones tardan menos en procesarse.

Columna “Mediana”: la mediana es un valor más bajo, lo que significa que la mitad de las peticiones están por debajo de un tiempo menor que la mitad de las peticiones de las otras aplicaciones.

Columna “90% Line”: el 90% de las peticiones tarda un tiempo por debajo de las 9 uds.

Columna “95% Line”: análogo.

Columna “99% Line”: análogo.

Columna “Mín”: esta columna en realidad da muy poca información, puesto que lo que le pase a una petición de entre todas las que se hacen no puede tomarse en consideración para sacar conclusiones generales. Sin embargo, cuadra que la que menos tarda haya tardado menos que la que menos tarda en las otras aplicaciones.

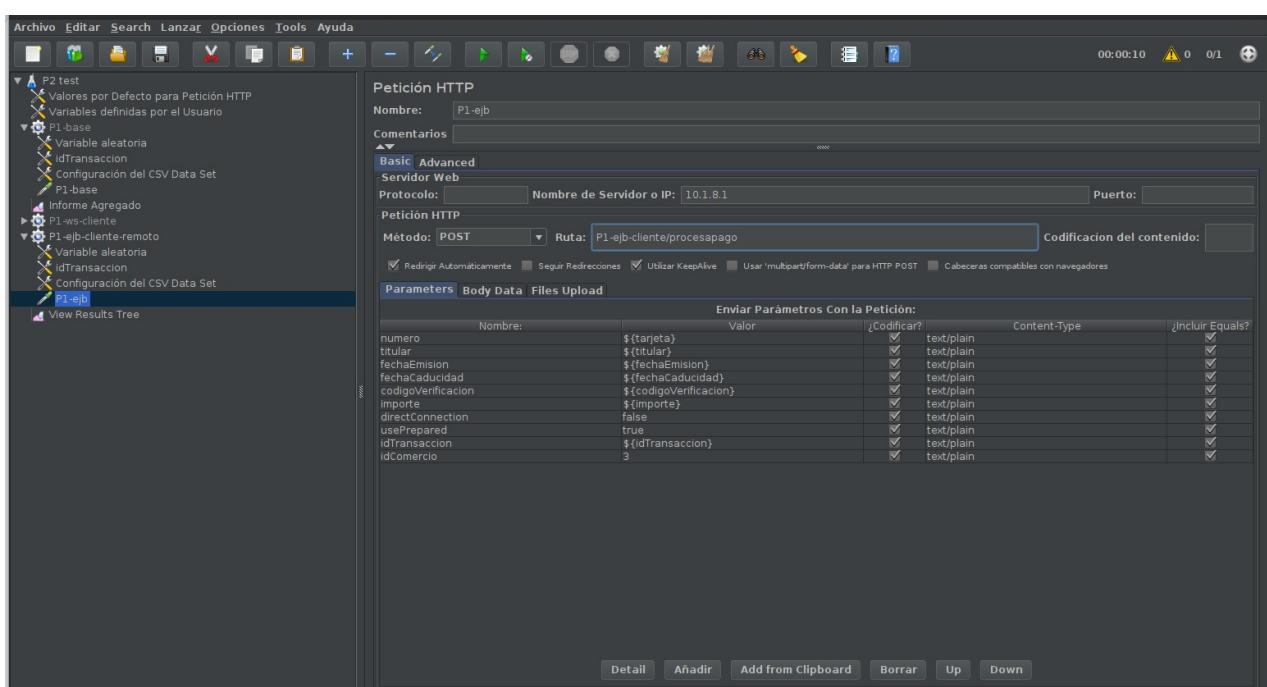
Columna “Max”: Como hemos dicho, lo que le pase a una petición no es muy representativo. En este caso, el mejor tiempo lo tiene la aplicación P1-ejb.

Columna “Rendimiento”: Nos indica la cantidad de peticiones procesadas por unidad de tiempo.

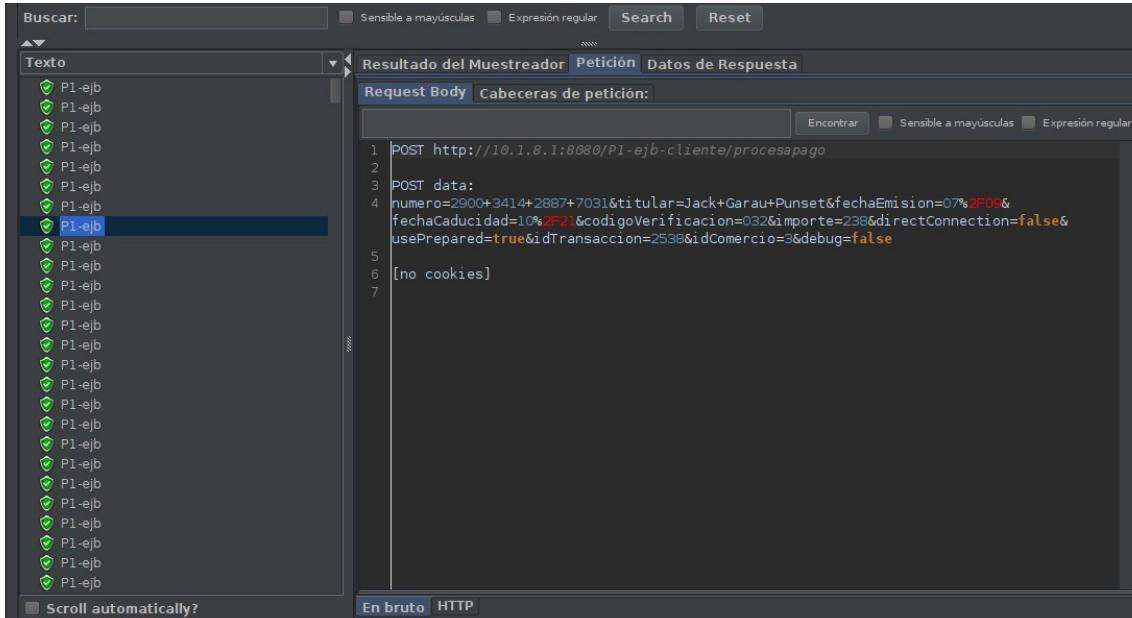
Ésta es una medida muy representativa. Con diferencia el rendimiento más alto es el de P1-base, lo que nos indica que es la aplicación que procesa mayor volumen de peticiones por unidad de tiempo.

Finalmente, en este ejercicio nos piden repetir la prueba de P1-ejb (inhabilitando P1-base y P1-ws) con el EJB local incluido en P1-ejb-servidor-remoto, en lugar de con el EJB remoto incluido en P1-ejb-cliente-remoto.

Cambiamos la ip para que ya no coja la que hay en valores por defecto, que es la del cliente, y cambiamos también el nombre de la aplicación, que ya no es P1-ejb-cliente-remoto, que es el cliente que creamos para acceder a la interfaz remota del ejb, sino P1-ejb-cliente, que es el cliente que creamos para el servidor utilizando la interfaz local.



Ejecutamos una vez de prueba, con el árbol de resultados activado, para comprobar que las peticiones se hacen correctamente, y vemos que efectivamente la petición se hace a la url <http://10.1.8.1:8080/P1-ejb-cliente/procesapago>:



Realizamos la query para comprobar que todos los pagos se han hecho:

```

psql (8.4.10)
Type "help" for help.

visa=# select count(*) from pago;
 count
-----
 1000
(1 row)

visa=#

```

A continuación, borramos los pagos de la base de datos con “delete from pago;” y volvemos a ejecutar con el árbol de resultados desactivado, obteniendo este informe agregado:

Nombre de archivo		Log/Mostrar sólo: <input type="checkbox"/> Escribir en Log Sólo Errores <input type="checkbox"/> Éxitos <input type="checkbox"/> Configurar											
Etiqueta	# Muestras	Media	Mediana	90% Line	95% Line	99% Line	Mín	Máx	% Error	Rendimiento	Kb/sec	Sent KB/sec	
P1-ejb	1000	8	5	9	11	17	3	2844	0,00%	108,8/sec	141,05	0,00	
Total	1000	8	5	9	11	17	3	2844	0,00%	108,8/sec	141,05	0,00	

Los resultados son mejores porque ahora las funciones de visadao se utilizan desde el sitio en el que están, es decir, desde el servidor, mediante una interfaz local, en lugar de desde un cliente con una interfaz remota. Esto hace que el proceso sea más rápido básicamente por la misma razón que en P1-base: porque se elimina el cliente P1-ejb-cliente-remoto, y lo que antes se hacía en dos pasos, contactando desde nuestro navegador con el cliente (actuando nuestro navegador de cliente y el cliente de servidor) y luego el cliente con el servidor, ahora se hace en un solo paso, contactando nuestro navegador con el servidor directamente. Así se crean la mitad de peticiones, y desaparece el tiempo que tardaba la otra mitad en enviarse.

## Ejercicio 4:

En este ejercicio primero hemos adaptado la configuración de nuestro servidor, tal y como se especifica en el enunciado. En esta parte de la práctica no hemos encontrado problemas, ya que estaba todo bastante explicado paso por paso, en el pdf.

Después, nos piden que revisemos el script *si2-monitor.sh* e indicar los mandatos asadmin para averiguar los siguientes valores:

- Max Queue Size del servicio HTTP

**asadmin --host 10.1.8.1 --passwordfile passwordfile get configs.config.server-config.thread-pools.thread-pool.http-thread-pool.max-queue-size**

```
si2@si2srv02:~$ asadmin --host 10.1.8.1 --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
```

- Maximum Pool Size del Pool de conexiones a nuestra BD

**asadmin --host 10.1.8.1 --passwordfile passwordfile get resources.jdbc-connection-pool.VisaPool.max-pool-size**

```
si2@si2srv02:~$ asadmin --host 10.1.8.1 --passwordfile passwordfile get resources.jdbc-connection-pool.VisaPool.max-pool-size
resources.jdbc-connection-pool.VisaPool.max-pool-size=32
Command get executed successfully.
```

Mandato para monitorizar el número de errores en las peticiones al servidor web:

**asadmin --host 10.1.8.1 --user admin --passwordfile passwordfile monitor --type httplistener**

```
si2@si2srv02:~$ asadmin --host localhost --user admin --passwordfile passwordfile monitor
--type httplistener
ec  mt  pt    rc
4   6742 27.00 4693
4   6742 24.00 5713
4   6742 27.00 6093
^Csi2@si2srv02:~$
```

## Ejercicio 5:

En este ejercicio nos piden simplemente que registremos en la hoja de cálculo los siguientes valores, que encontramos en la consola de administrador en los lugares indicados en el enunciado:

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

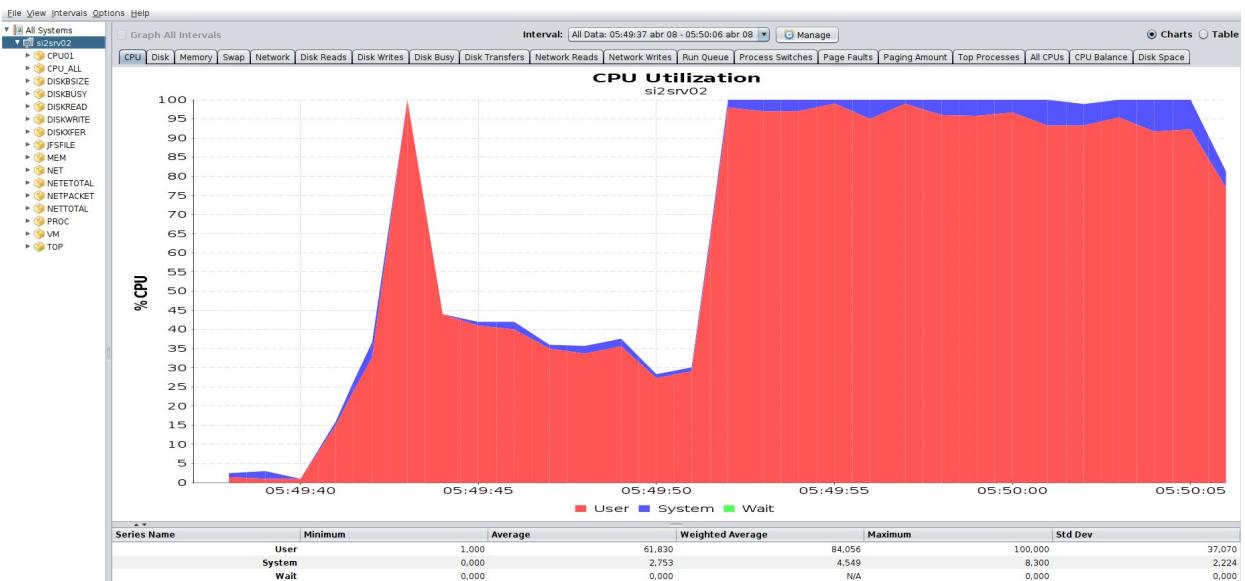
## Ejercicio 6:

En este punto de la práctica, para poder ejecutar si2-monitor.sh en el servidor, puesto que en el ordenador host daba problemas, tuvimos que instalar openssh-server, para poder utilizar el comando scp.

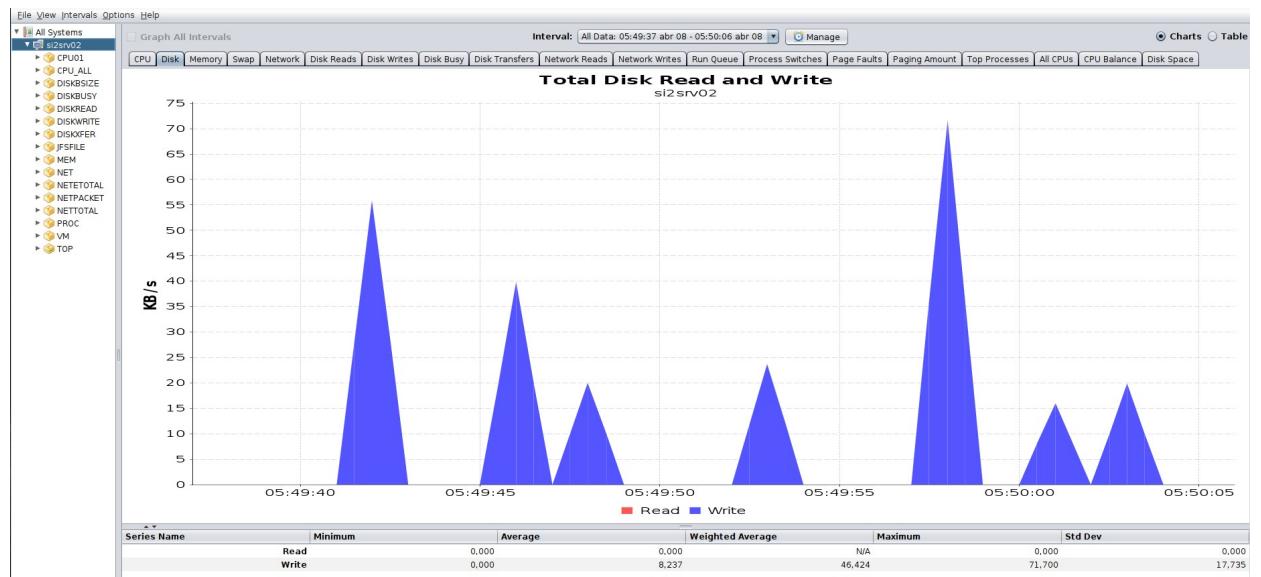
Ejecutamos el comando nmon -f -t -s 1 -c 30 en la máquina virtual del servidor, para ejecutar 30 capturas con 1 segundo de intervalo entre cada una de ellas, volcando el resultado a fichero y con la opción -t para incluir datos de los procesos que más consumen.

Aunque incluimos el fichero en la memoria, estas son algunas capturas del fichero abierto con el programa NMONVisualizer:

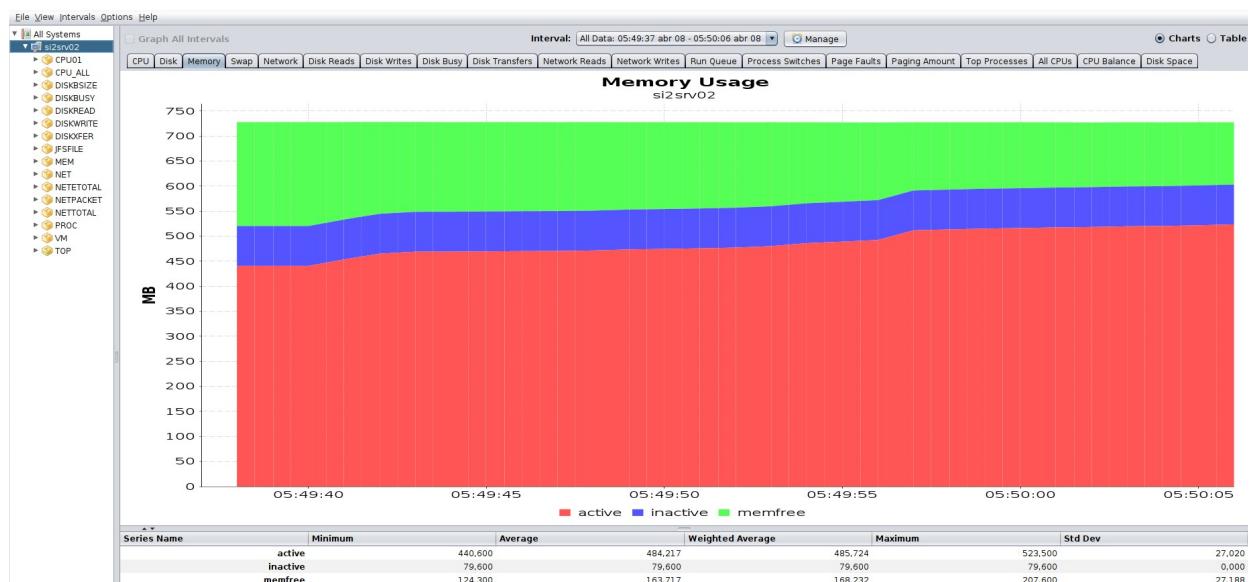
Porcentaje de uso de la CPU:



Accesos a disco en kilobytes por segundo:



## Cantidad de megabytes utilizados de ram:



Dado que las tres gráficas tienen unidades completamente distintas, puesto que la cpu se mide en porcentaje de uso respecto al total, los accesos a disco se mide en kilobytes/segundo y el uso de memoria ram se mide en megabytes, no nos parece que se puedan comparar entre ellos, al menos con los datos facilitados por NMON.

Sin embargo, podemos decir que la CPU alcanza en varios puntos, y en el segundo de ellos de manera prolongada, valores cercanos al 100%.

Del acceso a disco vemos que se hace a intervalos aislados, y, como los pagos no se guardan en memoria, sino en la base de datos, suponemos que no es muy representativo, que simplemente serán procesos internos del ordenador que están guardando datos en memoria que no se relacionan de manera directa con nuestras pruebas.

Del acceso a memoria vemos que es el más constante de los 3 comportamientos, pues aunque el servidor no guarda datos en disco, sí que mueve datos de un sitio a otro continuamente y para eso está utilizando la memoria ram, durante todo el tiempo que está funcionando.

Captura de la monitorización con si2-monitor.sh desde vm2:

```
si2@si2srv02:~/Documents$ ./si2-monitor.sh localhost
#Muestra numJDBCCount numHTTPCount      numHTTPQ
      0           1           0           0
      1           0           0           0
^C
TOT.MUESTRAS      MEDIA:
      2          0.5          0          0
```

No son valores representativos porque la prueba es demasiado rápida, pero más adelante, en el ejercicio 8 se ve su correcto funcionamiento.

Respecto a si esta situación es realista o no, nos parece que son datos útiles pero que hay que tener cuidado interpretándolo, ya que una cantidad de peticiones semejante no tiene mucho que ver con 1000 usuarios. Es decir, sabemos como se comporta nuestro sistema al recibir ese volumen de peticiones, que podría llegar a suceder en la realidad, pero quizás sucedería para 10000 usuarios, o incluso más, ya que hay que tener en cuenta que las peticiones están lanzándose a mucha velocidad

una tras otra, y esto no es lo que hace una persona utilizando ninguna pasarela de pagos.

Como se propone más adelante en el enunciado, se podría introducir un retardo entre una petición y otra, para intentar representar el tiempo que un usuario pasa pensando o haciendo cualquier cosa entre un pago y otro. Nos parece que esto puede hacerlo un poco más realista, sin embargo habría que pensar en qué situación un usuario tiene que hacer varios pagos seguidos (aquí si tiene sentido) o si por el contrario va a pasar mucho tiempo entre un pago y el siguiente, en cuyo caso habría que considerar las peticiones como peticiones aisladas, asumiendo que son de distintos usuarios, y en este caso ni siquiera el think time proporcionaría mayor semejanza con el comportamiento real. Este segundo caso es al que nos referimos cuando decimos que, si hubiese 1000 usuarios conectados a la aplicación y todos quisiesen hacer pagos, no sería muy probable que los ejecutaran uno detrás de otro a la velocidad de un ordenador, y llegasen al servidor 1000 pagos en el intervalo de unos segundos, si no que quizás en lo que 100 usuarios realizan su pago, los otros estarían pensando, de manera que solo habría que procesar 100 pagos en cada intervalo de tiempo.

En conclusión, los datos conseguidos son útiles pero hay que darles un contexto y tener cuidado al interpretarlos.

Finalmente, respecto a mejorar el funcionamiento de la aplicación en base a dar mayor capacidad al recurso más utilizado, lo primero que intentaríamos sería dar una mayor capacidad a la CPU, pues esta llega al máximo de su capacidad durante gran parte del funcionamiento del servidor. Es decir, en el entorno en el que estamos, crearíamos otra máquina virtual con 2 CPU's para que se puedan repartir el procesamiento de los pagos.

### **Ejercicio 7:**

En el enunciado se dice explícitamente que este ejercicio no genera información para la memoria de la práctica.

### **Ejercicio 8:**

En este ejercicio ya tenemos el script de prueba preparado, y seguimos como siempre todos los pasos del enunciado.

Como herramientas de monitorización, utilizamos el script si2-monitor.sh, desde la máquina virtual 2 (en el ordenador host no obteníamos ningún dato), y el comando vmstat, para obtener de ahí el porcentaje de uso de CPU en lugar de con nmon.

El comando exacto para utilizar vmstat es:

```
vmstat -n 1 | (trap " INT; awk '{print; if(NR>2) cpu+=$13+$14;}END{print"MEDIA";print "NR:",NR,"CPU: ",cpu/(NR-2);}';)
```

Respecto a si2-monitor.sh, lo hemos enviado a vm2 con el comando:

scp si2-monitor.sh [si2@10.1.8.1](mailto:si2@10.1.8.1):

El proceso que seguimos para la monitorización es dejar preparados los comandos en ambas terminales (abrimos dos terminales conectadas a vm2 con ssh [si2@10.1.8.1](mailto:si2@10.1.8.1), y ejecutamos un comando en cada una), y lanzar el banco de pruebas en jmeter.

Una vez lanzado, nos fijamos en el número de hilos creados (que representan usuarios) en la esquina superior derecha, y una vez termina el periodo “Ramp-Up”, es decir, cuando se llega al número de usuarios indicado, ejecutamos los comandos.

Mientras se está ejecutando la prueba, permanecemos atentos al número de hilos, y cuando empieza a descender, detenemos ambos comandos con ctrl+c.

Tras realizar cada prueba, anotamos en la hoja de cálculo, para cada número de usuarios (hilos):

- Del comando vmstat, el porcentaje de uso de CPU.
- De la ejecución de si2-monitor, las conexiones de visa pool utilizadas (máximo de 32), el porcentaje de hilos ocupados (máximo de 5) y las peticiones que hay en la cola (máximo 4096)
- Del informe agregado, tiempo de media, línea del 90% y rendimiento.

Los resultados obtenidos están en el documento SI2-P2-curvaProductividad.ods, y se comentan en el siguiente ejercicio.

## **Ejercicio 9:**

### Cálculo del punto de saturación

Calculamos el punto de saturación de la gráfica throughput-total, puesto que la petición se compone de la comprobación de la tarjeta y del proceso del pago, con lo cual, nos parece que lo más representativo es estudiar ambos.

Lo primero que queremos comentar de los resultados obtenidos, es que para 250 usuarios, el comportamiento no era normal. Debido probablemente a que eran una carga de trabajo excesivamente pequeña, y no sabemos a qué factores más, el rendimiento que obteníamos era muy superior a lo esperado, y de ahí que la gráfica tenga esa forma, en la que empieza en un valor alto, luego disminuye hasta el mínimo, y luego ya se comporta de forma normal. Tras varias mediciones, seguimos obteniendo valores alrededor de  $80\text{-}90 \text{ s}^{-1}$ . No hemos considerado este valor para calcular el punto de saturación, ni lo hemos tenido en cuenta, pero dado que el enunciado pedía que empezásemos la monitorización con 250 usuarios, lo hemos dejado en la memoria.

A partir de las gráficas obtenidas, podemos calcular el punto de saturación. Para ello, tenemos que dar una aproximación lineal del comportamiento de la gráfica en zona lineal, y esto lo hacemos calculando la recta de regresión. Lo hemos hecho con un calculador de recta de regresión online, y hemos cogido los valores para 500, 750, 1000 y 1250 usuarios, que es donde la gráfica es más similar a una línea recta.

La ecuación que obtenemos es  $y = -68.8 + 0.151 * x$ .

Debido a que los resultados experimentales suelen variar respecto a los teóricos, a que el entorno de pruebas no era para nada el idóneo (una máquina virtual dentro de un ordenador con recursos bastante limitados, que además está ejecutando los programas de monitorización) y que, para eliminar ruido, las medidas deberían tomarse muchas más veces de lo que hemos podido hacer, no hemos obtenido una zona de transición clara. Sin embargo, el punto de saturación sigue considerándose el punto de corte entre la recta que representa el comportamiento lineal y la recta que representa el comportamiento constante.

El comportamiento más parecido a una constante se obtiene de los 1250 usuarios a los 2000. En este caso no vamos a calcular la recta de regresión, pues tenemos la condición de que la pendiente de la recta tiene que ser 0. Lo que vamos a hacer es una recta horizontal en el valor de media, que es el 117,8. Por tanto, la recta será  $y = 117,8$ .

El punto de corte de ambas rectas es el  $(1235,76, 117,8)$ , y por tanto, se alcanza a los 1235 usuarios, y tiene un throughput de  $117,8 \text{ s}^{-1}$ .

Por último, el throughput máximo en zona de saturación, vemos en la gráfica que es el de los 1250 usuarios que es  $124,1 \text{ s}^{-1}$ .

## Análisis de los valores obtenidos:

Vamos a buscar recursos que estén siendo desaprovechados, porque haya más de los que se utilizan, y de los que por lo tanto podemos reducir la cantidad.

Por otro lado, también queremos ver qué recursos están siendo utilizados al máximo de su capacidad, y de los que por lo tanto vendría bien tener un poco más.

Fijándonos en los resultados de si2-monitor.sh, vemos que el número de conexiones visa pool, ha estado siempre por debajo de las 5 conexiones, mientras que el tamaño que hemos reservado para este recurso es de 32 conexiones. De aquí sería posible recortar.

El siguiente parámetro, los hilos que procesan peticiones, están siendo utilizados al máximo de su capacidad a partir de los 1750 usuarios, y puede ser que hagan falta más para las cantidades más altas de usuarios.

Por último, el tamaño de la cola, que es de 4096, creemos que es el recurso más desaprovechado, ya que en los momentos de mayor saturación, solo estamos viendo, como mucho, unas 600 peticiones en la cola.

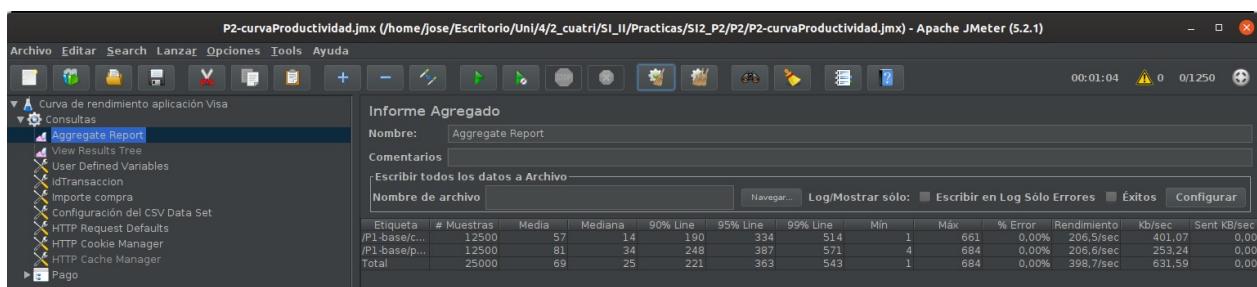
En una primera prueba, probamos simplemente aumentando el número de hilos del servidor de 5 a 10, y obtuvimos resultados muy malos, de la mitad del rendimiento que obteníamos para 5 hilos. De aquí ni siquiera cogimos datos, pues era obvio que no era el camino correcto.

Otro día distinto, además de los 10 hilos, redujimos el tamaño de la cola a 1024 peticiones, y, con el servidor reiniciado, la aplicación desplegada desde 0 y habiendo hecho las pruebas de calentamiento, volvimos a realizar la prueba.

Como buscamos que el punto de saturación aumente, nos vale con tomar muestras inmediatamente posteriores al punto de saturación anterior, para ver si siguen teniendo los valores de la recta constante o si han seguido aumentando como la recta de la sección de crecimiento lineal.

Tomamos muestras de 1250 y 1500 usuarios, y obtenemos los mejores rendimientos hasta el momento. Ambas pruebas, tanto para 1250 como para 1500 usuarios, se han repetido varias veces, y hemos tomado los valores en los que oscilaban la mayoría de ellas (para alguna obteníamos valores raros, excesivamente bajos o altos, y esas las hemos descartado).

Para 1250 usuarios, un rendimiento de caso 400 peticiones por segundo.



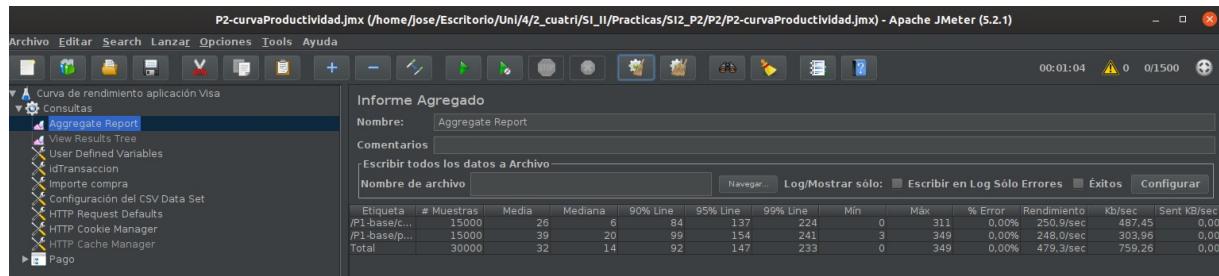
Valores de si2-monitor:

- Conexiones visa pool: 4,64
- Hilos en el servidor: 10
- Peticiones en cola: 175,64

Porcentaje de uso de CPU: 98,2.

Vemos que se utilizan los 10 hilos del servidor, así que hemos hecho muy bien aumentándolos.

Para 1500 usuarios, un rendimiento cercano a las 500 peticiones por segundo.



Valores de si2-monitor:

- Conexiones visa pool: 1,96
- Hilos en el servidor: 4,81
- Peticiones en cola: 4,22

Porcentaje de uso de CPU: 97,54.

Somos conscientes de que surgen muchas incoherencias en los valores, como por ejemplo que para 1500 usuarios trabajen menos hilos del servidor que con 1250, y que se hayan encontrado tan pocas peticiones en la cola en 1500 y tantas en 1250, pero son los valores que hemos obtenido haciendo la monitorización perfectamente, y esperamos que no se tengan en cuenta puesto que los conocimientos teóricos los hemos aplicado lo mejor posible, y, repetimos, el entorno de pruebas no era idóneo.