UAM UNIVERSIDAD AUTONOMA DE MADRID		**Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2**			
Grupo	**2401**	**Práctica**	2	**Fecha**	17/04/2021
Alumno/a		De las Heras Moreno, Martín			
Alumno/a		Valderrábano Zamorano, Santiago Manuel			

Práctica 2: Rendimiento

Ejercicio número 1:

Para realizar este ejercicio, hemos instalado JMeter, copiado las aplicaciones P1-base, P1-ws y P1-ejb (servidor remoto) y las hemos desplegado.
Después, hemos ido siguiendo los pasos que se indican en el enunciado para construir los planes de pruebas indicados. Todo esto se ha guardado en el fichero ***P2.jmx*** el cual se adjunta en el fichero de entrega.
Añadido a esto comentar que, para los siguientes ejercicios, como se recomienda en el enunciado, hemos añadido un árbol de resultados que desactivaremos tras comprobar que todo funciona correctamente.

Ejercicio número 2:

Al realizar este ejercicio, debido a la situación actual y que no tenemos acceso a los ordenadores de los laboratorios, hemos utilizado un único PC en el que desplegamos ambas máquinas virtuales.

Antes de desplegar las aplicaciones que se van a utilizar, hemos revisado y modificado los siguientes parámetros:

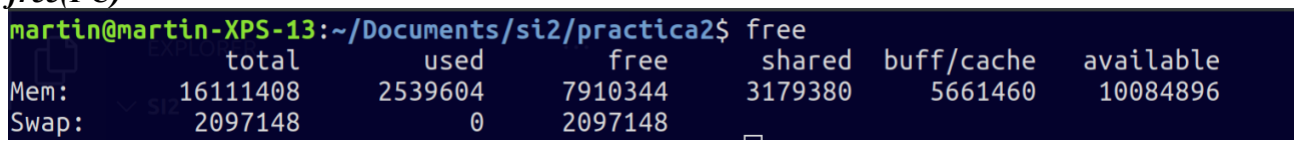
- P1-base:
  - build.properties:
    - as.host=**10.1.7.2**
  - postgresql.properties:
    - db.host=**10.1.7.1**
    - db.client.host=**10.1.7.2**
- P1-ws:
  - build.properties:
    - as.host.client=**10.1.7.2**
    - as.host.server=**10.1.7.1**
  - postgresql.properties:
    - db.host=**10.1.7.1**
    - db.client.host=**10.1.7.1**
- P1-ejb-servidor-remoto:
  - build.properties:
    - as.host.client=**10.1.7.1**
    - as.host.server=**10.1.7.1**
  - postgresql.properties:
    - db.host=**10.1.7.1**
    - db.client.host=**10.1.7.1**

- P1-ejb-cliente-remoto:
  - build.properties:
    - as.host.client=**10.1.7.2**
  - postgresql.properties:
    - db.host=**10.1.7.1**
    - db.client.host=**10.1.7.1**
  - glassfish-web.xml
    - **10.1.7.1**

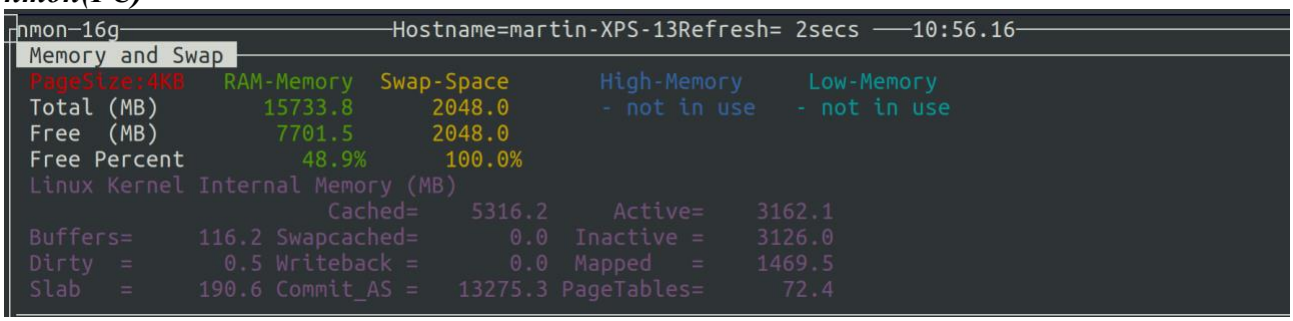
Tras realizar estos cambios, desplegamos las aplicaciones y hemos hecho un pago de prueba en cada una para probar que funcionan correctamente.

A continuación, se nos pide que mostremos las salidas de los comandos ***free*** y ***nmon*** (tras pulsar la tecla "M") ejecutados tanto en el PC como en las máquinas virtuales.

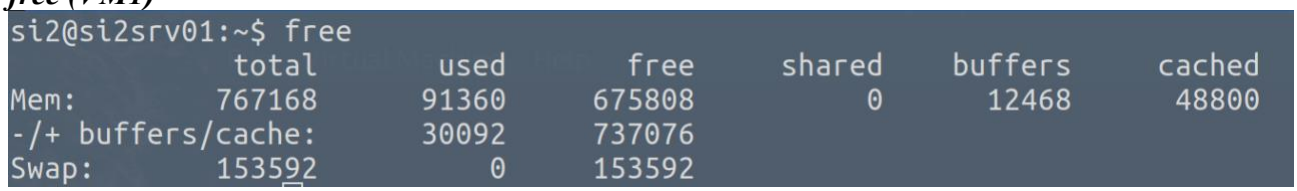
free(PC)



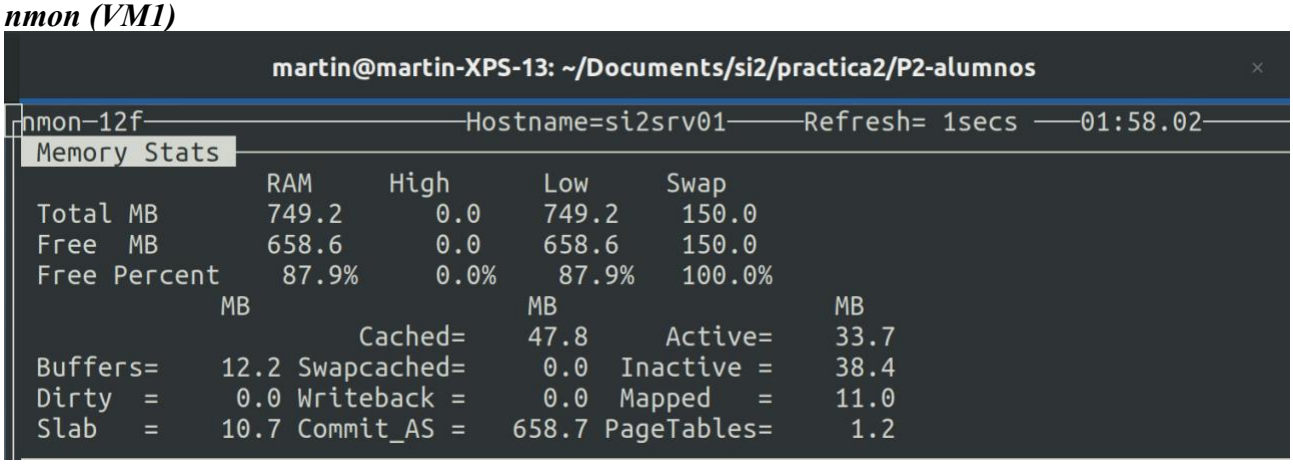
nmon(PC)



free (VM1)



nmon (VM1)



free (VM2)

```
si2@si2srv02:~$ free
             total       used       free     shared    buffers     cached
Mem:        767168      88556     678612          0      13048      47108
-/+ buffers/cache:      28400     738768
Swap:       153592          0     153592
```

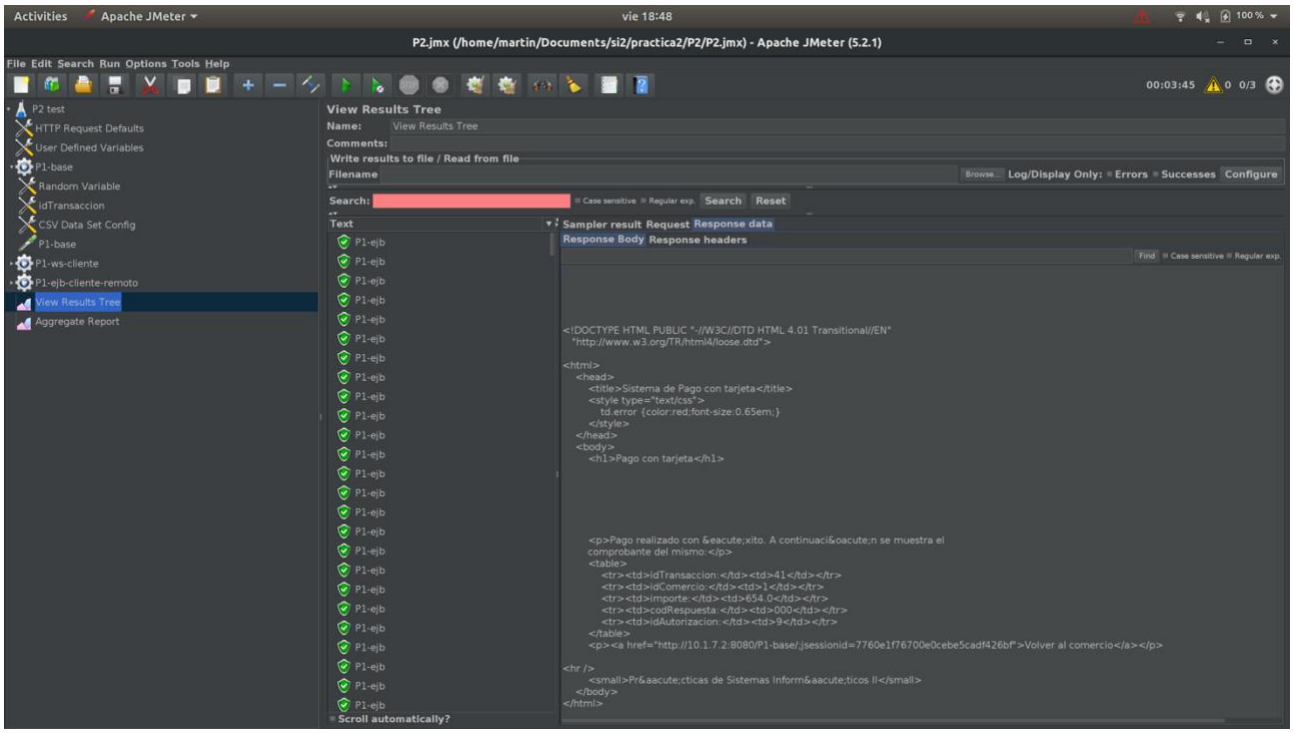
nmon (VM2)

```
nmon-12f                    Hostname=si2srv02       Refresh= 1secs ---01:58.42
 Memory Stats
                RAM      High       Low      Swap
 Total MB       749.2       0.0     749.2     150.0
 Free  MB       661.3       0.0     661.3     150.0
 Free Percent    88.3%      0.0%     88.3%    100.0%
              MB                        MB                      MB
                      Cached=      46.1       Active=      32.5
 Buffers=     12.7 Swapcached=       0.0   Inactive =      36.9
 Dirty  =      0.0 Writeback =       0.0   Mapped   =       8.0
 Slab   =     10.7 Commit_AS =     620.2 PageTables=       1.0
```

Ejercicio número 3:

En este ejercicio ejecutaremos el plan de pruebas preparado anteriormente sobre las 3 versiones que hemos preparado de la práctica haciendo uso de JMeter.
Para comprobar que se han realizado correctamente todos los pagos, hemos observado tres elementos distintos:

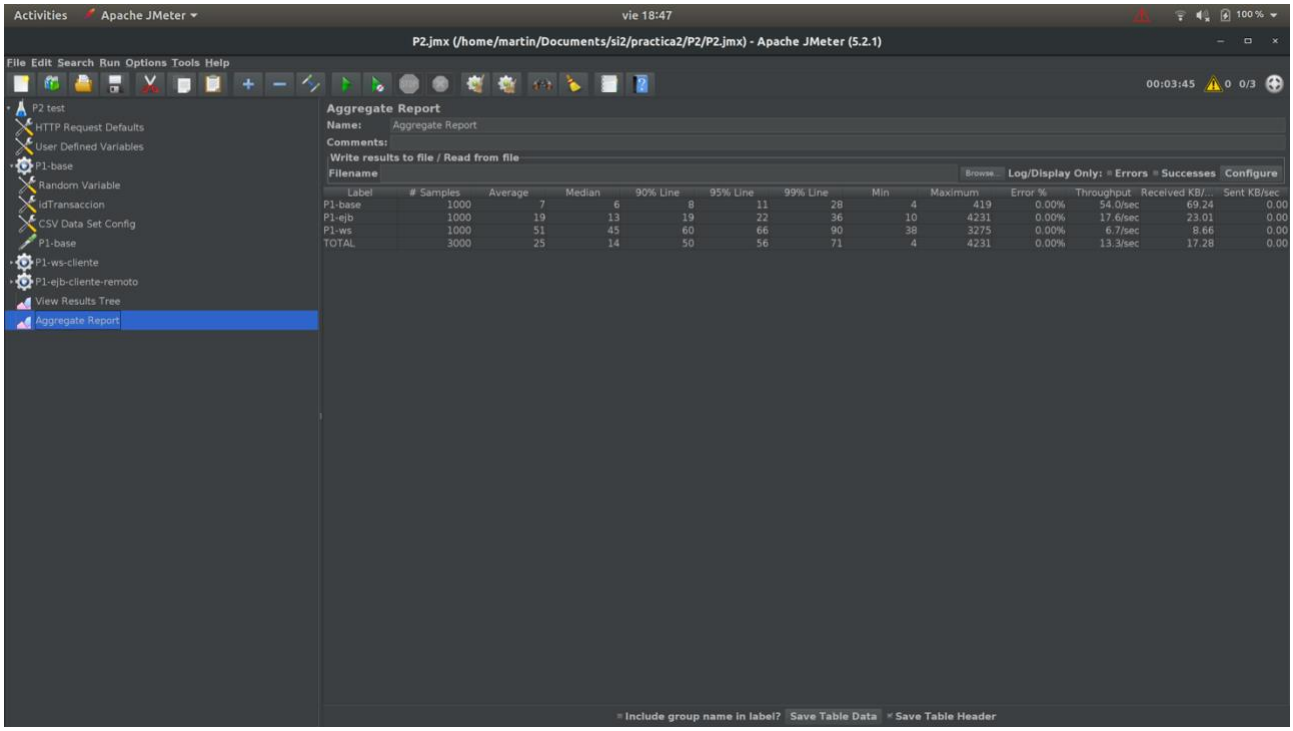
- ***Result Tree*** (JMeter)



- ***Base de Datos Visa*** (VM1)

```
visa=# select count(*) from pago;
 count
-------
  3000
(1 row)
```

- ***Aggregate Report*** (JMeter)



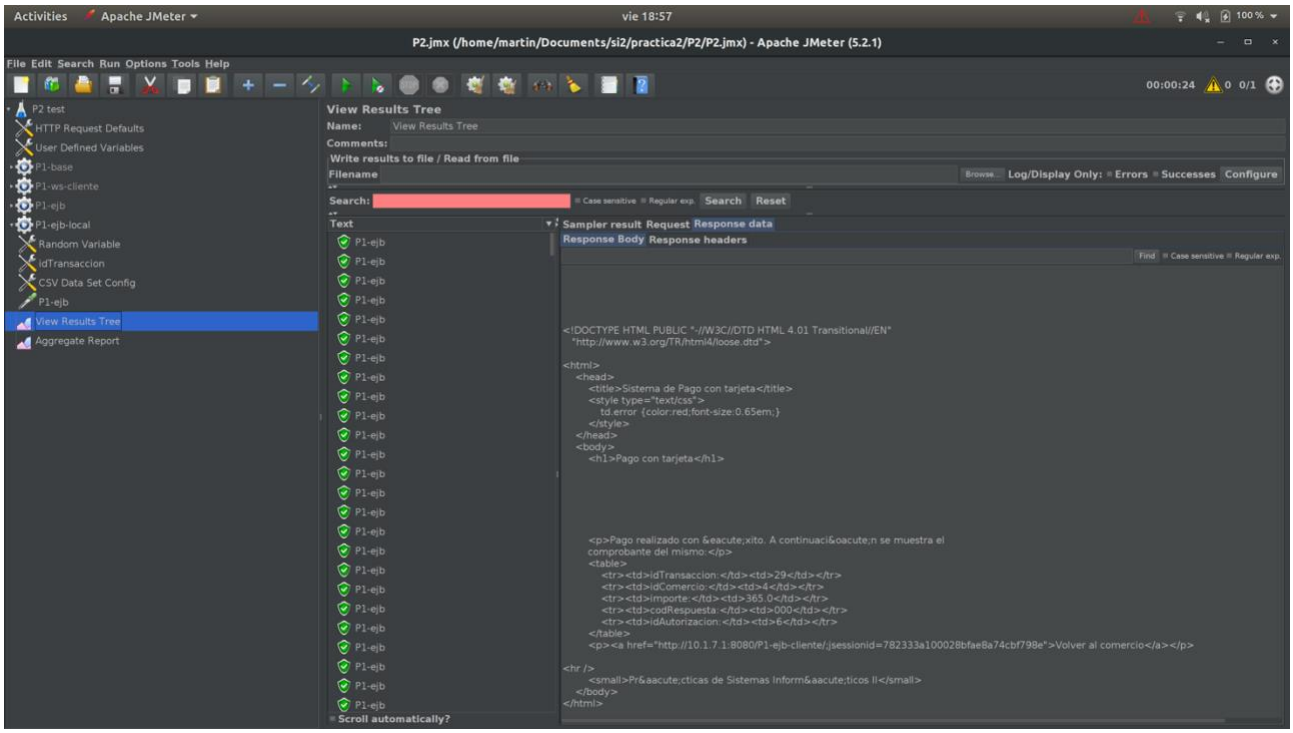
Se puede observar en la última captura que el Error es del 0% por lo que se han ejecutado correctamente.
A continuación, analizamos los resultados. Para entender el comportamiento tenemos que mirar las siguientes columnas:

- ***Throughpt*** → Es el Rendimiento. Califica la calidad de la aplicación ejecutada.
- ***Average*** → Indica el tiempo medio de respuesta en milisegundos.

Si observamos estos parámetros, se observa claramente que la aplicación que mejor rendimiento da es la P1-base (tanto throughput como average son mayores que las otras 2), después observamos que la segunda mejor es P1-ejb y por último P1-ws. (Conclusiones al final del ejercicio junto con ejb local)

Después, se nos pide que realicemos la misma batería de pruebas, pero únicamente con la aplicación P1-ejb-cliente (local). A continuación, se muestran los resultados obtenidos:

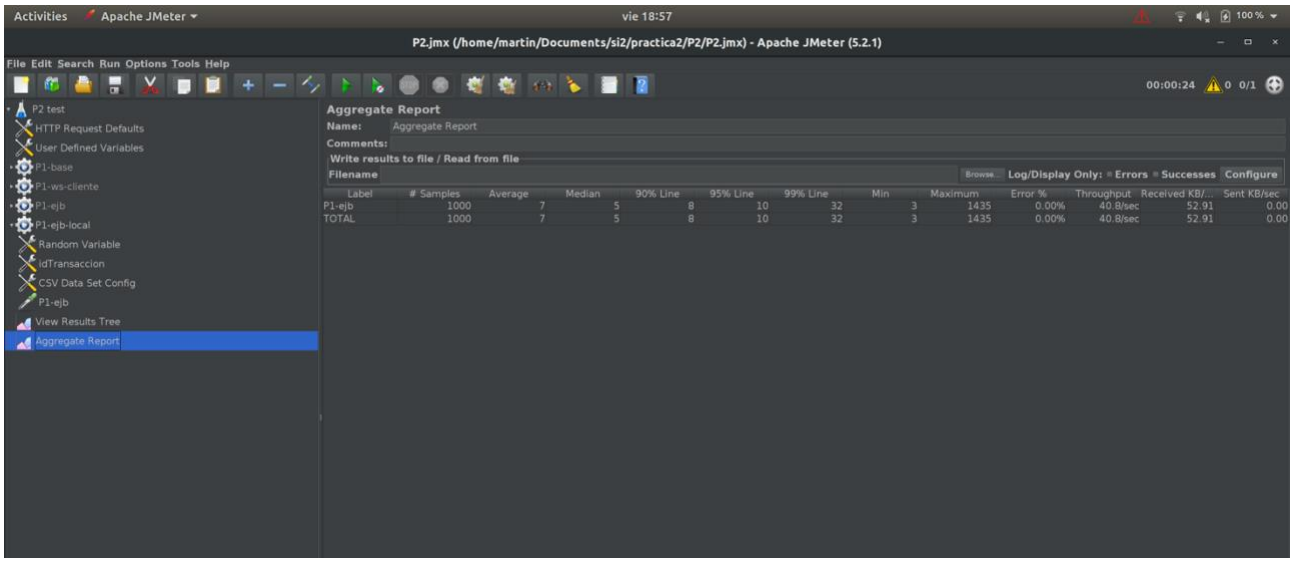
- ***Result Tree*** (JMeter)



- ***Base de Datos Visa*** (VM1)

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

- ***Aggregate Report*** (JMeter)



Volvemos a observar los parámetros ***throughput*** y ***average*** del *Aggregate Report*.
Se observa que los resultados son muy similares a los de P1-base. De hecho, la media (average) es exactamente la misma y el rendimiento (throughput) es ligeramente inferior.

Para terminar, analizando la diferencia de rendimientos nos surge la duda de cuál es razón por la que P1-base P1-ejb (local) tienen un rendimiento tan superior al resto. Esto se puede deber a diferentes factores como pueden ser el ordenador en el que se ejecutan las pruebas o el tipo de arquitectura. Creemos que esta última puede ser uno de los factores principales ya que al ser ambas de tipo "local" no dependen de un intermediario para acceder a las funciones del servidor, sino que lo hacen directamente a través del navegador.

Ejercicio número 4:

Para este ejercicio, hemos adaptado la configuración del servidor como se indica en el enunciado. Después, hemos obtenido el archivo que contiene esta configuración en la maquina virtual *($opt/glassfish4/glassfish/domains/domain1/config/domain.xml*) y lo hemos copiado en nuestro PC haciendo uso del comando ***scp.***

A continuación, revisamos el fichero ***si2-monitor.sh*** (que se nos provee como material de la práctica), en busca de los mandatos ***asadmin*** que debemos ejecutar en la VM1 para conocer los siguientes parámetros:

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

```
martin@martin-XPS-13:~/Documents/si2/practica2/P2$ asadmin --host 10.1.7.1 --passwordfile passwordfile get configs.config.server-config.thread-pools.thread-pool.http-thread-poo
l.max-queue-size
configs.config.server-config.thread-pools.thread-pool.http-thread-pool.max-queue-size=4096
Command get executed successfully.
martin@martin-XPS-13:~/Documents/si2/practica2/P2$
martin@martin-XPS-13:~/Documents/si2/practica2/P2$ asadmin --host 10.1.7.1 --passwordfile passwordfile get resources.jdbc-connection-pool.VisaPool.max-pool-size
resources.jdbc-connection-pool.VisaPool.max-pool-size=32
Command get executed successfully.
```

Monitorizar el número de errores en las peticiones al servidor:

```
martin@martin-XPS-13:~/Documents/si2/practica2/P2$ asadmin --host 10.1.7.2 --user admin --passwordfile passwordfile monitor --type httplistener
ec   mt   pt     rc
0    1617996556812 974109931.00 1661
0    1617996556812 527550251.00 3067
0    1617996556812 424559603.00 3811
```

Donde cada columna representa lo siguiente:

- **ec:** Contador de errores
- **mt:** Tiempo máximo de respuesta
- **pt:** Tiempo acumulado requerido para responder cada solicitud
- **rc:** Número de solicitudes.

Ejercicio número 5:

Para este ejercicio hemos accedido a la consola de administración de Glassfish y buscamos los parámetros requeridos según se indica en el enunciado. Después introducimos los valores en la tabla indicada de la hoja de cálculo SI2-P2-curvaProductividad.ods.

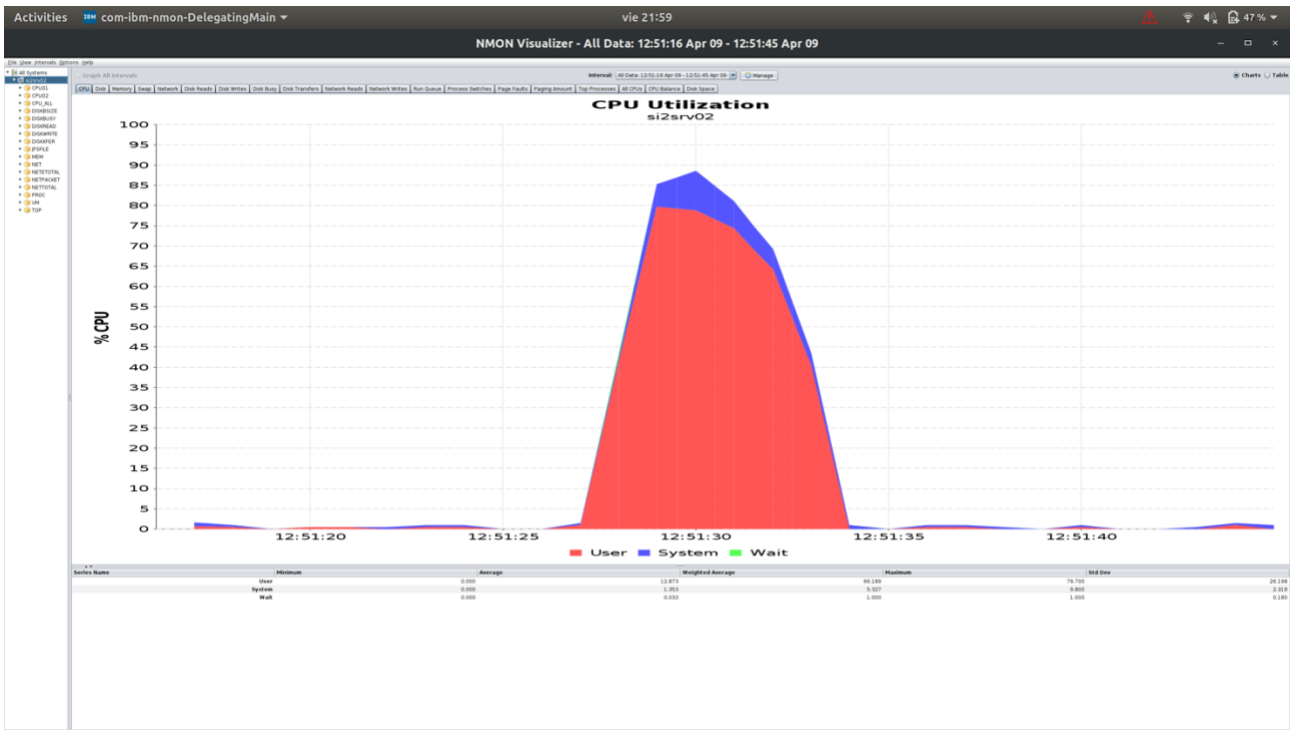
Parámetros de configuración			
Elemento	**Parámetro**		**Valor**
JVM Settings	Heap Máx. (MB)		512m
JVM Settings	Heap Mín. (MB)		512m
HTTP Service	Max.Thread Count		5
HTTP Service	Queue size		4096
Web Container	Max.Sessions		-1
Visa Pool	Max.Pool Size		32

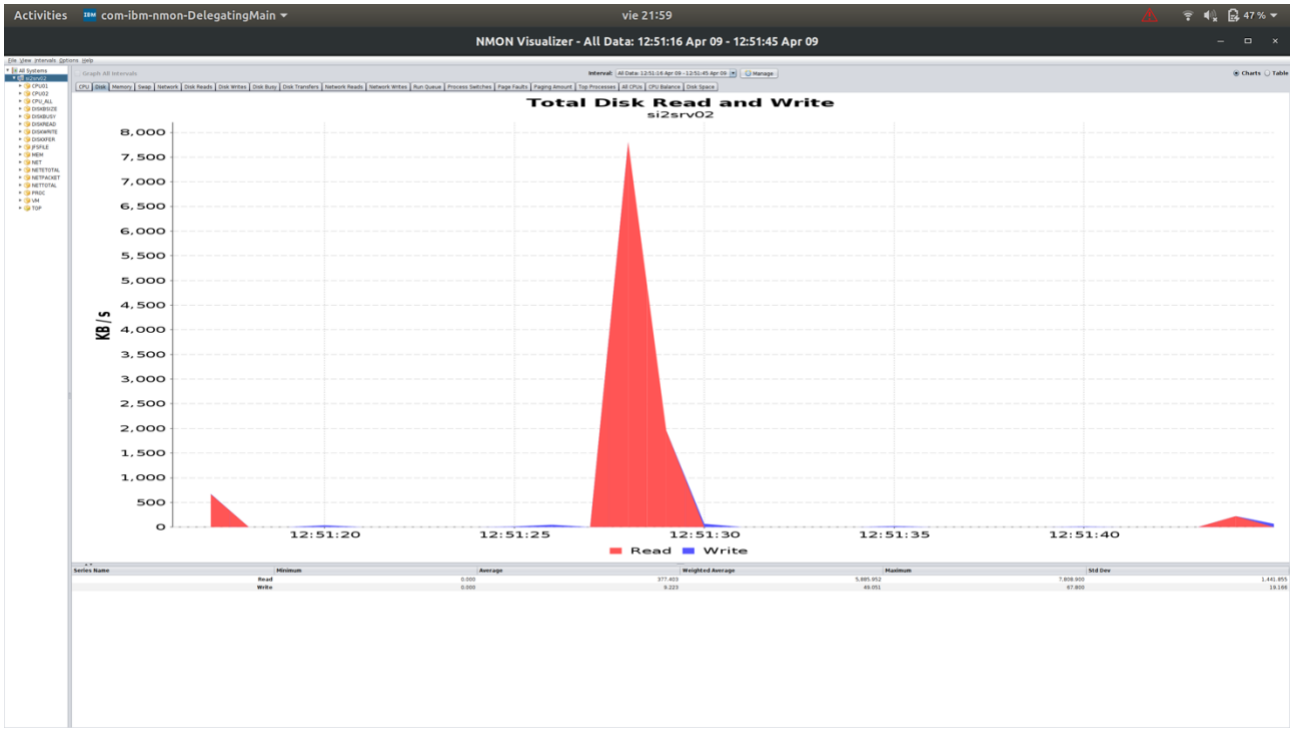
Ejercicio número 6:

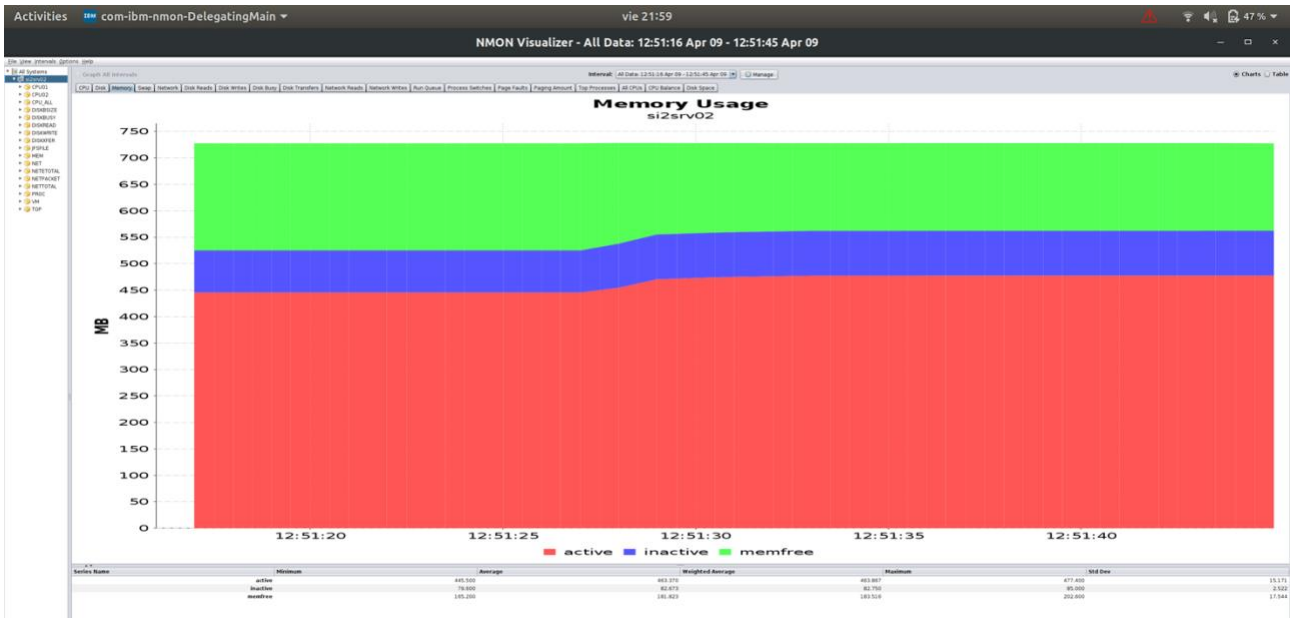
Para realizar este ejercicio, ejecutamos el siguiente comando para activar la monitorización en el servidor:

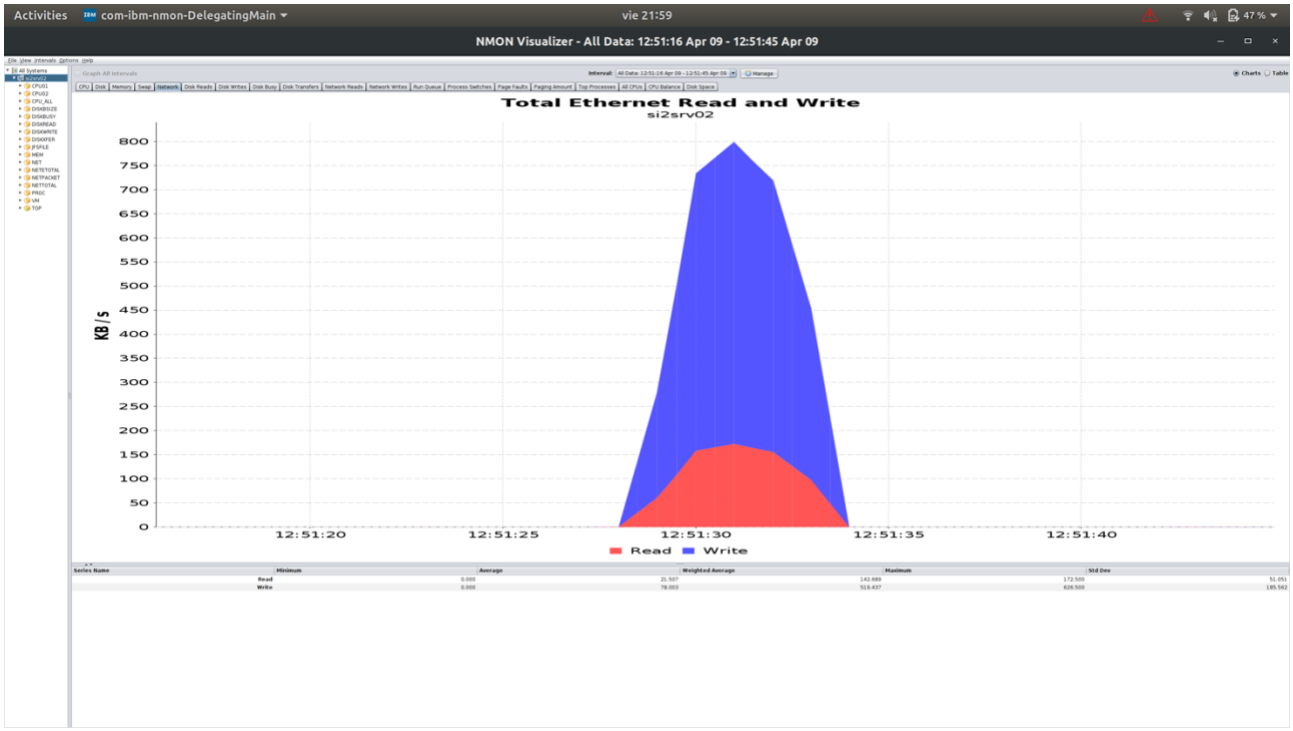
$nmon -f -t -s 1 -c 30

A continuación, ejecutamos de nuevo el plan de pruebas y observamos el fichero .nmon generado haciendo uso de la herramienta NMON Visualizer:




Activities
com-ibm-nmon-DelegatingMain
vie 21:59
47 %
NMON Visualizer - All Data: 12:51:16 Apr 09 - 12:51:45 Apr 09
Total Disk Read and Write
si2srv02
KB/s
8,000
7,500
7,000
6,500
6,000
5,500
5,000
4,500
4,000
3,500
3,000
2,500
2,000
1,500
1,000
500
0
12:51:20
12:51:25
12:51:30
12:51:35
12:51:40
Read
Write


Activities
com-ibm-nmon-DelegatingMain
vie 21:59
47 %
NMON Visualizer - All Data: 12:51:16 Apr 09 - 12:51:45 Apr 09
Memory Usage
si2srv02
MB
750
700
650
600
550
500
450
400
350
300
250
200
150
100
50
0
12:51:20
12:51:25
12:51:30
12:51:35
12:51:40
active
inactive
memfree



De los datos obtenidos en las capturas anteriores, podemos sacar varias conclusiones.

En primer lugar, se observa un notable aumento del uso de la CPU durante el tiempo de ejecución del programa de pruebas. Este tiempo no es muy prolongado, posiblemente debido a que la CPU del ordenador utilizado tiene un gran procesador. Creemos que se debe a que todo se ejecuta en una única máquina y cliente y servidor comparten recursos por el esquema que tiene P1-base.

Por otro lado, observamos también un aumento de lectura del disco en el mismo rango de tiempo en el que ocurre el aumento del uso de la CPU.

Además, del acceso a memoria podemos decir que se mantiene constante aunque con un ligero pico alrededor del rango de tiempo en el que se ejecuta el plan de pruebas.

Para terminar, observamos un enorme aumento de la lectura y escritura a través del Ethernet en el intervalo de tiempo de ejecución. Esto es lógico y se debe a que los accesos a base de datos para leer y escribir datos se hace a través de la red.

También se nos pide que comentemos la salida de la ejecución del script si-2monitor.sh:

```
martin@martin-XPS-13:~/Documents/si2/practica2/P2$ ./si2-monitor.sh 10.1.7.2
    #Muestra numJDBCCount numHTTPCount     numHTTPQ
           0          -1             0            0
           1          -1             1            0
           2          -1             1            0
           3          -1             1            0
^C
TOT.MUESTRAS        MEDIA:
           4          -1          0.75            0
```

Podemos observar que falla la monitorización de numJDBCCount. Esto pensamos de inicio que era así pero en el ejercicio 8 que lo volvemos a utilizar nos dimos cuenta del error y lo volvimos a descargar y funcionó a la perfección por lo que entendemos que el error en este ejercicio se debe a

que no se descargó bien el script o modificamos algo sin darnos cuenta que produjo el fallo.

Por otro lado, para contestar la pregunta de si el esquema utilizado es realista, creemos que **NO** lo es. Esto se debe a varias razones. En primer lugar, como hemos dicho anteriormente, hemos realizado las pruebas en un único PC con dos máquinas virtuales, situación que no es nada común en la realidad. Además, la P1-base tiene integrados el proceso cliente y servidor en el mismo ordenador, esquema que tampoco es realista.

Por último, creemos que un esquema mas realista y adecuado sería el de P1-ejb ya que podríamos desplegar el cliente y el servidor en máquinas distintas lo cual reduciría el uso total de CPU ya que no tendrían que compartir recursos como en el caso que hemos simulado.

Ejercicio número 7:

Este ejercicio, como se indica en el enunciado, no requiere que se responda ni que se adjunten capturas.

Ejercicio número 8:

En este ejercicio, vamos a ejecutar la batería de pruebas proporcionada en el material de Moodle. Mientras se realiza la ejecución, vamos a monitorizar los recursos del servidor. Para ello vamos a usar 3 métodos:

1. Aggregate Report (JMeter)
2. si2-monitor.sh
3. *vmstat -n 1 | (trap '' INT; awk '{print; if(NR>2) cpu+=$13+$14;}END{print"MEDIA"; print "NR: ",NR,"CPU: ", cpu/(NR-2);}';)*

Los resultados obtenidos, los vamos anotando en el fichero SI2-P2-curvaProductividad.ods.

Además, para verificar que los datos del .ods son verdaderos incluimos un directorio llamado ej8 el cual contiene una carpeta por cada número de usuarios, “C”, para los que se ejecutan la batería de pruebas. Cada una de estas carpetas contiene los siguientes ficheros:

- monitor.txt → Salida de si2-monitor.sh
- cpu-vm2.txt → Salida de la ejecución del comando vmstat.
- aggregate-report.jtl → Salida de la ejecución del JMeter.
- aggregate-report.csv → Salida de la ejecución de JMeter en formato csv.

Los resultados obtenidos los comentamos en el siguiente ejercicio.

Prueba de rendimiento

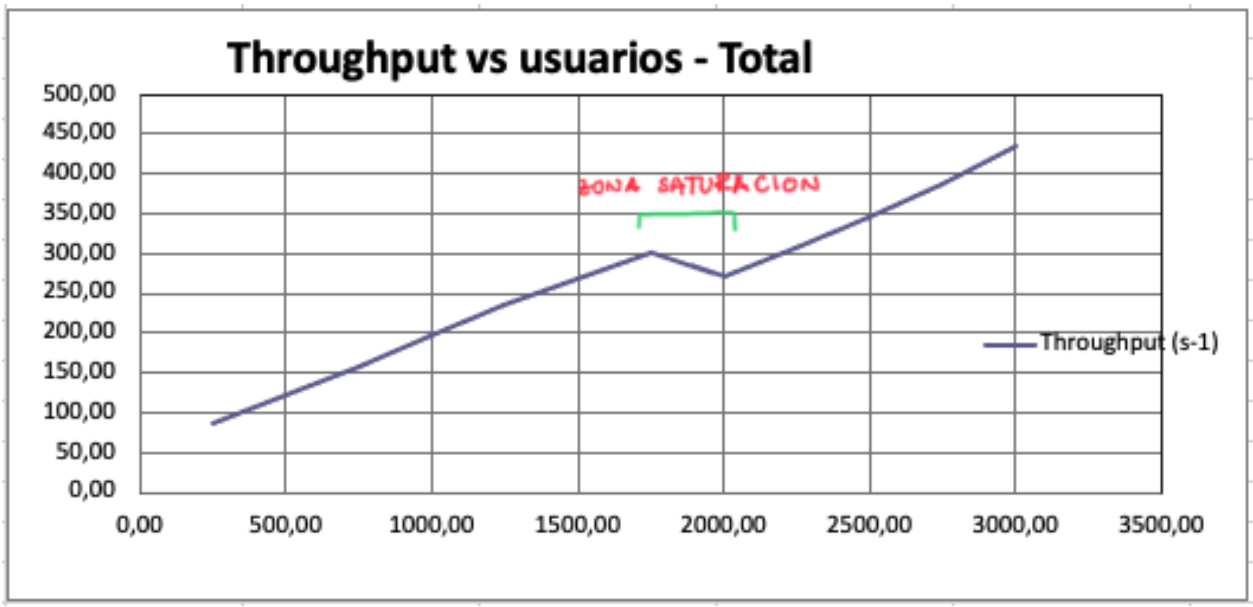
	Sistema	Monitores			Total			ProcesaPago		
Usuarios	CPU, average (%)	Visa Pool used Conns	HTTP Current Threads Busy	Conn queued, instant	Average (ms)	90% line (ms)	Throughput (s^{-1})	Average (ms)	90%line (ms)	Throughput (s^{-1})
250,00	31,26	0,06	0,24	0,02	10,00	22,00	85,54	14,00	27,00	44,43
500,00	42,24	0,43	0,86	0,11	9,00	21,00	121,25	13,00	27,00	61,71
750,00	46,80	0,83	2,03	0,75	10,00	25,00	158,87	14,00	31,00	80,36
1000,00	49,98	1,36	2,16	3,47	12,00	29,00	196,32	16,00	36,00	99,01
1250,00	56,80	1,36	3,47	17,78	22,00	48,00	234,06	27,00	55,00	117,83
1500,00	60,85	2,29	3,66	83,43	72,00	156,00	267,75	76,00	162,00	134,62
1750,00	67,80	2,69	4,06	161,38	135,00	329,00	300,58	140,00	332,00	150,99
2000,00	67,16	2,21	3,29	293,58	1210,00	1685,00	272,05	1207,00	1712,00	136,26
2250,00	63,78	1,99	2,89	264,50	1228,00	1797,00	307,04	1231,00	1825,00	153,78
2500,00	49,22	1,42	2,07	180,75	1278,00	1880,00	345,90	1286,00	1908,00	173,24
2750,00	54,45	1,47	2,14	174,39	1364,00	1964,00	388,65	1364,00	1988,00	194,66
3000,00	58,97	1,80	2,51	208,31	1478,00	2106,00	435,29	1474,00	2132,00	218,02

Ejercicio número 9:

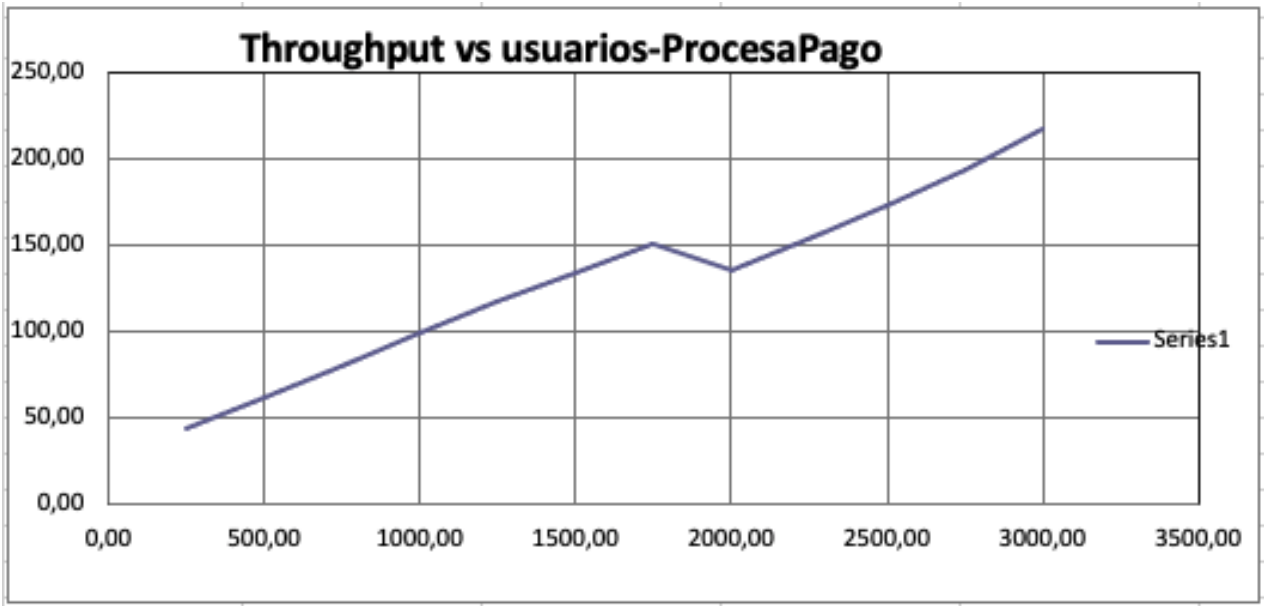
Respecto a la primera pregunta, observamos que el punto de saturación se produce entre 1750 y 2000 usuarios. Llegamos a esta conclusión por 2 motivos:

1) Se observa en la gráfica que hay una bajada del throughput entre los 2 valores mencionados, de 307,04 a 272,05.
2) Al observar los Aggregate Reports de ambos observamos que en el de 2000 usuarios es el primero en el que encontramos errores, un 0,128%.

Haciendo una estimación y en base al throughput que observamos durante la ejecución en JMeter, estimamos que el throughput máximo se encuentra alrededor de 320 (s^{-1}).

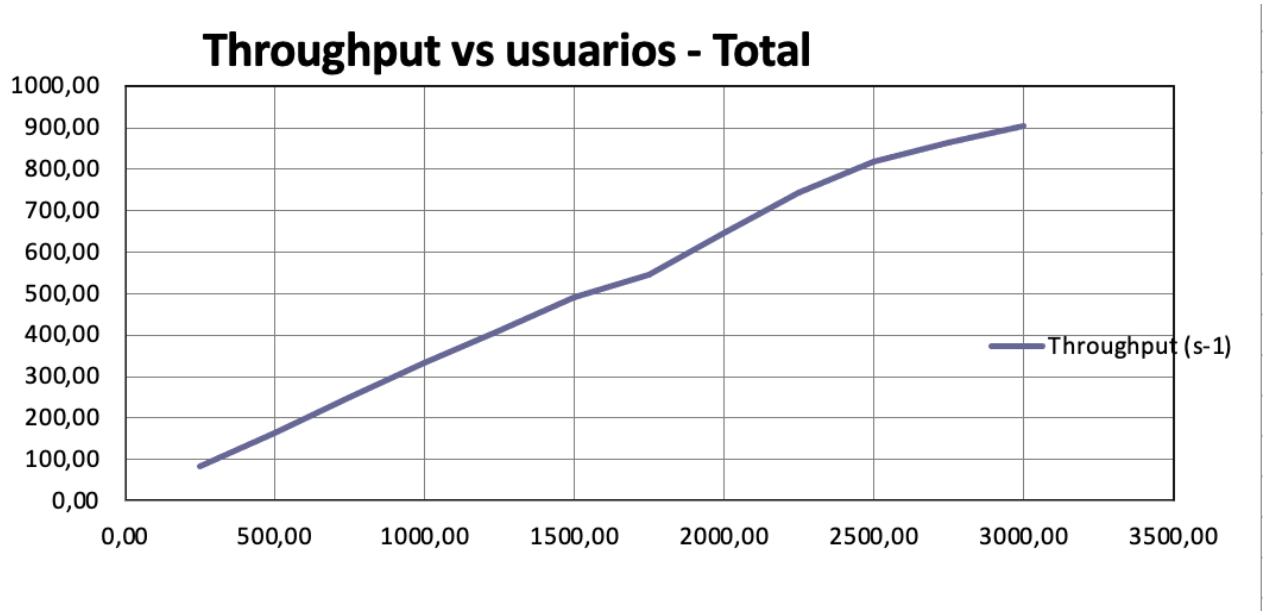


También podemos confirmar los resultados anteriores fijándonos en la gráfica del throughput de ProcesaPago ya que el punto de saturación ocurren en el mismo intervalo de usuarios.



Para mejorar el rendimiento y alcanzar la saturación con un número de usuarios mayor, creemos que debemos modificar el número de hilos de procesamiento del servidor. Por defecto para esta práctica, tanto el máximo como el mínimo es de 5 hilos.

En nuestro caso, decidimos aumentar estos valores a 10 el mínimo y 15 el máximo. Y tras ejecutar de nuevo la batería de pruebas, confirmamos que no solo el throughput es mayor sino que, para 3000 usuarios, que es el máximo que probamos, no encontramos error de conexión por lo que el punto de saturación ocurrirá con más de 3000 usuarios.



Concluimos por tanto, afirmando que el cambio realizado es muy efectivo.