		Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2			
Grupo	2401	Práctica	2	Fecha	26/04/2021
Alumno/a		Lerma, Martínez, Francisco			
Alumno/a		Morato, Martín, Jesús			

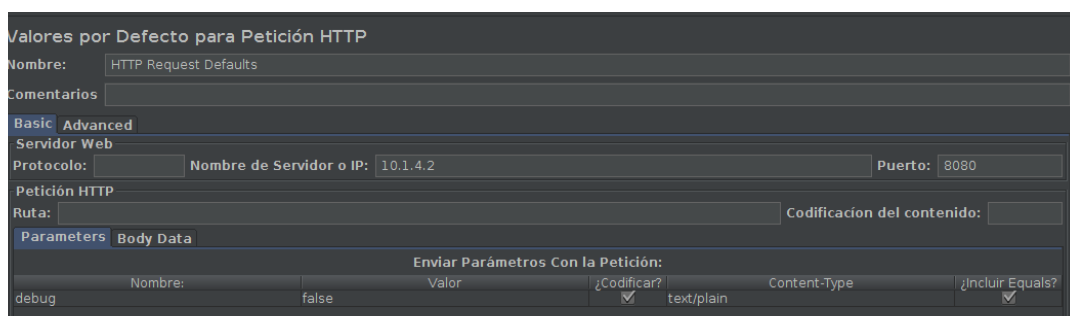
Objetivos de la práctica

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 de arquitectura puede 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.

Ejercicio 1

Para la realización de este ejercicio hemos seguido todos los pasos que se indican en el enunciado de la práctica, primero hemos procedido con la instalación del Jmeter. Posteriormente, hemos procedido a crear un plan de pruebas “P2 test” y añadir los parámetros por defecto en “HTTP Request Defaults”, con IP del servidor 10.1.4.2:



Después, 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.

User Defined Variables

Name:

Comments:

User Defined Variables			
	Name:	Value	Description
samples		1000	

Introducimos el thread group que simulara el conjunto de usuarios que accede a la aplicación:

Thread Group

Name:

Comments:

Action to be taken after a Sampler error:

☒ Continue
 ☐ Start Next Thread Loop
 ☐ Stop Thread
 ☐ Stop Test
 ☐ Stop Test Now

Thread Properties

Number of Threads (users):

Ramp-up period (seconds):

Loop Count: ☐ Infinite

☒ Same user on each iteration
☐ Delay Thread creation until needed
☐ Specify Thread lifetime

Duration (seconds):

Startup delay (seconds):

Con el fin de introducir elementos dinámicos en las peticiones que se traducirán en sentencias SQL distintas, vamos a utilizar el elemento de configuración que permitirá que el importe de cada pago tenga un valor comprendido entre 1 y 1000, relacionando este como la variable “importe” para uso en adelante:

Random Variable

Name:

Comments:

Output variable

Variable Name:

Output Format:

Configure the Random generator

Minimum Value:

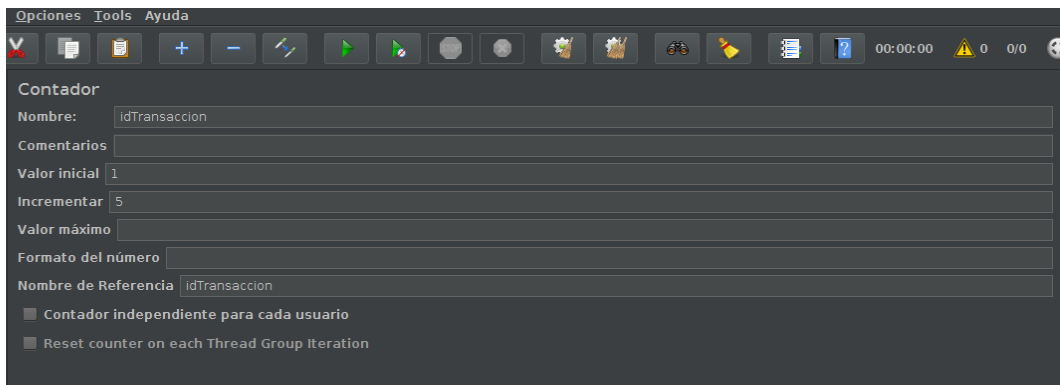
Maximum Value:

Seed for Random function:

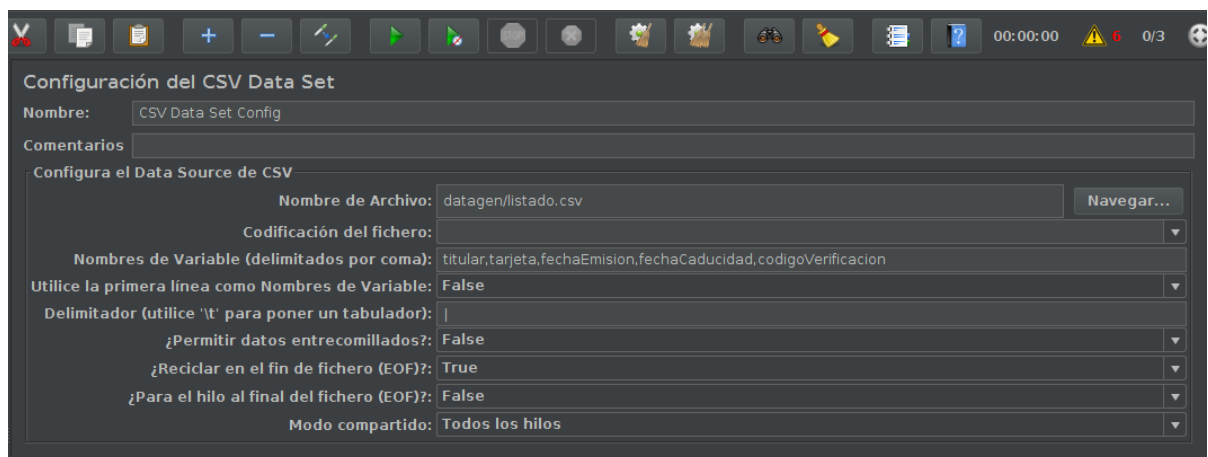
Options

Per Thread(User)?:

Además también tenemos que simular la generación del identificador de transacción puesto que este es un parámetro que no se puede repetir en cada pago de la tienda:



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, para ello introducimos la ruta relativa al fichero que se encontrará en “datagen/listado.csv”:



Añadiremos el generador de peticiones HTTP, además de definir todos los parámetros necesarios para que se puedan llevar a cabo las peticiones:

HTTP Request

Name: P1-base

Comments:

Basic **Advanced**

Web Server

Protocol [http]: Server Name or IP: Port Number:

HTTP Request

Method: POST Path: P1-base/procesapago Content encoding:

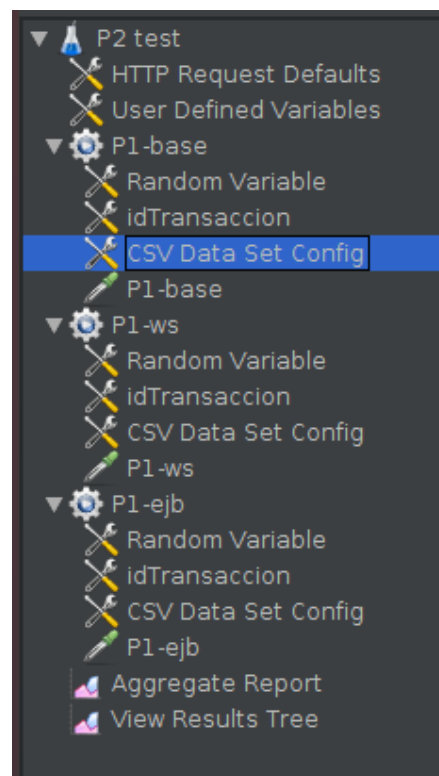
☒ Redirect Automatically ☐ Follow Redirects ☒ Use KeepAlive ☐ Use multipart/form-data ☐ Browser-compatible headers

Parameters **Body Data** **Files Upload**

Send Parameters With the Request:

Name:	Value	URL Encode?	Content-Type	Include Equals?
Name	P1-base	<input type="checkbox"/>	text/plain	<input checked="" type="checkbox"/>
Path	P1-base/procesapago	<input type="checkbox"/>	text/plain	<input checked="" type="checkbox"/>
numero	\${tarjeta}	<input checked="" type="checkbox"/>	text/plain	<input checked="" type="checkbox"/>
titular	\${titular}	<input checked="" type="checkbox"/>	text/plain	<input checked="" type="checkbox"/>
fechaEmision	\${fechaEmision}	<input checked="" type="checkbox"/>	text/plain	<input checked="" type="checkbox"/>
fechaCaducidad	\${fechaCaducidad}	<input checked="" type="checkbox"/>	text/plain	<input checked="" type="checkbox"/>
codigoVerificacion	\${codigoVerificacion}	<input checked="" type="checkbox"/>	text/plain	<input checked="" type="checkbox"/>
importe	\${importe}	<input checked="" type="checkbox"/>	text/plain	<input checked="" type="checkbox"/>
directConnection	false	<input checked="" type="checkbox"/>	text/plain	<input checked="" type="checkbox"/>
usePrepared	true	<input checked="" type="checkbox"/>	text/plain	<input checked="" type="checkbox"/>
idTransaccion	\${1idTransaccion}	<input checked="" type="checkbox"/>	text/plain	<input checked="" type="checkbox"/>
idComercio	1	<input checked="" type="checkbox"/>	text/plain	<input checked="" type="checkbox"/>

Por último y tras añadir los listener tanto “Aggregate Report” cómo “View Result Tree” adaptamos el plan de pruebas para que pueda inspeccionar cada uno de los proyectos que hicimos en la primera práctica P1-base, P1-ws y P1-ejb (habrá que meter los nuevos path a las aplicaciones y cambiar el inicio de los id de transacción y comercio para que no haya colisiones de claves) de tal forma que finalmente nos queda el siguiente archivo .jmx:



Ejercicio 2

Para este ejercicio hemos preparado el PC y las máquinas virtuales tal como se nos indica en la figura 22 del enunciado para el diagrama de despliegue: En la maquina virtual 1 (PC1VM) hemos iniciado el servidor de Glassfish junto con Postgresql y en la maquina virtual 2 (PC2VM) hemos detenido PostgreSQL con el comando `sudo service postgresql-8.4 stop`. Posteriormente hemos replegado todas las aplicaciones que teníamos desplegadas en los servidores, hemos cambiado los ficheros `build.properties` y `postgres.properties` de tal forma que sigan la estructura que se nos pide en el ejercicio. Se adjuntan capturas de los cambios:

P1-base:

```
as.host=10.1.4.2
```

```
db.port=5432
db.host=10.1.4.1
# Recursos y pools asociados
db.pool.name=VisaPool
db.jdbc.resource.name=jdbc/VisaDB
db.url=jdbc:postgresql://${db.host}
db.client.host=10.1.4.1
db.client.port=4848
```

P1-ws:

```
as.host.client=10.1.4.2
as.host.server=10.1.4.1
```

```
db.port=5432
db.host=10.1.4.1
# Recursos y pools asociados
db.pool.name=VisaPool
db.jdbc.resource.name=jdbc/VisaDB
db.url=jdbc:postgresql://${db.host}
db.client.host=10.1.4.2
db.client.port=4848
```

P1-ejb-cliente-remoto:

```
as.host=10.1.4.2
```

```
db.port=5432
db.host=10.1.4.1
# Recursos y pools asociados
db.pool.name=VisaPool
db.jdbc.resource.name=jdbc/VisaDB
db.url=jdbc:postgresql://${db.host}
db.client.host=10.1.4.1
db.client.port=4848
```

P1-ejb-servidor-remoto:

```
as.host.client=10.1.4.1
as.host.server=10.1.4.1
```

```
db.port=5432
db.host=10.1.4.1
# Recursos y pools asociados
db.pool.name=VisaPool
db.jdbc.resource.name=jdbc/VisaDB
db.url=jdbc:postgresql://${db.host}
db.client.host=10.1.4.1
db.client.port=4848
```

Una vez hemos cambiado estas direcciones de las máquinas, hemos procedido a modificar el fichero glassfish-web.xml para indicar la IP del EJB remoto que usa P1-ejb-cliente:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE glassfish-web-app PUBLIC "-//GlassFish.org//DTD GL
"http://glassfish.org/dtds/glassfish-web-app_3_0-1.dtd">
<glassfish-web-app>
  <ejb-ref>
    <ejb-ref-name>VisaDAOBean</ejb-ref-name>
    <jndi-name>corbaname:iiop:10.1.4.1:3700#java:global/P1-e
  </ejb-ref>
</glassfish-web-app>
```

Para redespregar todas las aplicaciones hemos creado el script "inicio.sh" que se encarga de redespregar las anteriores aplicaciones y limpiar las bases de datos.

Tras esto hemos desplegado todas las aplicaciones (en la máquina virtual 2 nos aseguramos antes de que hemos parado el proceso postgresql) y hemos ido anotando la salida de los comandos "free" y "nmon --> m" de tal forma que lo recogemos en las siguientes capturas de pantalla.

El comando free nos muestra 6 columnas que nos dan la siguiente información:

- total: memoria total del sistema, tanto la instalada como la asignada a swap.
- usado: memoria total en uso.
- libre: memoria total libre o que no está en uso.
- compartido: memoria compartida usada normalmente por los ficheros del sistema o tmpfs.
- búfer/caché: suma total de la memoria caché y del buffer.
- disponible: memoria total disponible para iniciar nuevas aplicaciones.

Por otro lado, el comando nmon nos muestra la memoria total, la memoria para swap, y la high memory que es la memoria de usuario y la low memory que es la memoria del kernel.

PC-Host:

- Free:

```
jesus99@jesus99-TM1701: ~/Escritorio/SI2-buena/SI2/P2
Archivo Editar Ver Buscar Terminal Ayuda
jesus99@jesus99-TM1701:~/Escritorio/SI2-buena/SI2/P2$ free
Memoria:      total        usado        libre    compartido búfer/caché    disponible
Swap:         2097148      423864      1673284
jesus99@jesus99-TM1701:~/Escritorio/SI2-buena/SI2/P2$
```

- Nmon:

```
nmon-16g——[H for help]——Hostname=jesus99-TM170Refresh= 2secs ——14:03.55
Memory and Swap
PageSize:4KB  RAM-Memory  Swap-Space      High-Memory      Low-Memory
Total (MB)    7845.1      2048.0      - not in use     - not in use
Free (MB)     428.9      1634.3
Free Percent  5.5%      79.8%
Linux Kernel Internal Memory (MB)
                Cached=    3377.2      Active=    5018.6
Buffers=       59.2 Swapcached=    9.8 Inactive =   1944.7
Dirty  =        0.2 Writeback =    0.0 Mapped  =   2247.9
Slab   =       216.3 Commit_AS =  14590.6 PageTables=    79.7
```

MV1:

- Free:

```
si2@si2srv01:~$ free
Mem:      total        used        free      shared    buffers    cached
-/+ buffers/cache:  351400    415768
Swap:     153592          0      153592
```

- Nmon:

```
jesus99@jesus99-TM1701: ~
Archivo Editar Ver Buscar Terminal Ayuda
nmon-12f——Hostname=si2srv01——Refresh= 2secs ——05:06.17
Memory Stats
RAM      High      Low      Swap
Total MB  749.2    0.0     749.2    150.0
Free MB   182.9    0.0     182.9    150.0
Free Percent  24.4%    0.0%    24.4%    100.0%
MB                                MB                                MB
                Cached=    179.6      Active=    459.5
Buffers=       42.4 Swapcached=    0.0 Inactive =    83.1
Dirty  =        0.0 Writeback =    0.0 Mapped  =    25.7
Slab   =       14.5 Commit_AS =  1039.9 PageTables=    1.7
```

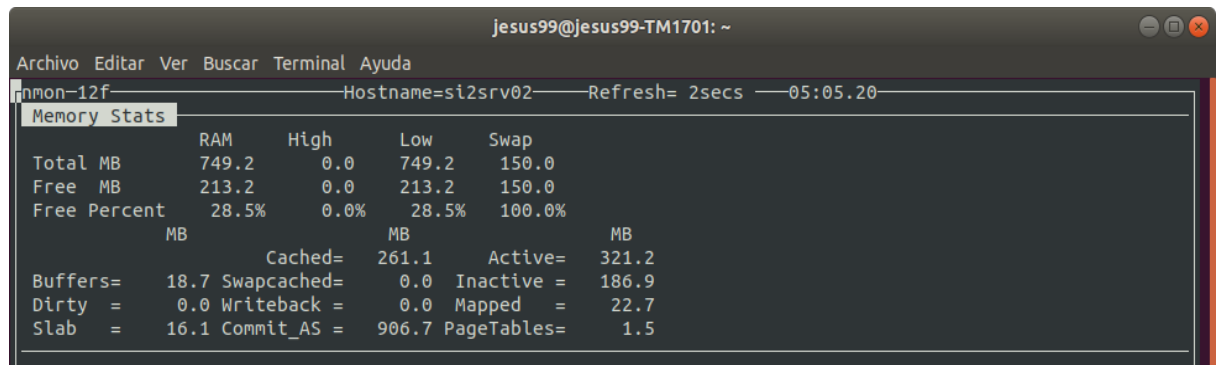
MV2:

- Free:

```
si2@si2srv02:~$ free
```

	total	used	free	shared	buffers	cached
Mem:	767168	547008	220160	0	19096	267264
-/+ buffers/cache:		260648	506520			
Swap:	153592	0	153592			

- Nmon:



The screenshot shows the nmon terminal window with the title 'jesus99@jesus99-TM1701: ~'. The menu bar includes 'Archivo', 'Editar', 'Ver', 'Buscar', 'Terminal', and 'Ayuda'. The status bar shows 'nmon-12f', 'Hostname=si2srv02', 'Refresh= 2secs', and '05:05.20'. The 'Memory Stats' section is highlighted, displaying the following data:

	RAM	High	Low	Swap
Total MB	749.2	0.0	749.2	150.0
Free MB	213.2	0.0	213.2	150.0
Free Percent	28.5%	0.0%	28.5%	100.0%

	MB		MB		MB
Buffers=	18.7	Cached=	261.1	Active=	321.2
Dirty =	0.0	Swapcached=	0.0	Inactive =	186.9
Slab =	16.1	Writeback =	0.0	Mapped =	22.7
		Commit_AS =	906.7	PageTables=	1.5

Tras esto, hemos procedido a realizar manualmente los pagos de calentamiento que se nos indican en enunciado para comprobar que todas las aplicaciones funcionan:

P1-base:

Pago con tarjeta

Pago realizado con éxito. A continuación se muestra el comprobante del mismo:

idTransaccion: 5
idComercio: 3
importe: 45.0
codRespuesta: 000
idAutorizacion: 1

[Volver al comercio](#)

P1-ws:

Pago con tarjeta

Pago realizado con éxito. A continuación se muestra el comprobante del mismo:

idTransaccion: 56
idComercio: 21
importe: 12.0
codRespuesta:
idAutorizacion:

[Volver al comercio](#)

P1-ejb:

Pago con tarjeta

Pago realizado con éxito. A continuación se muestra el comprobante del mismo:

idTransaccion: 345
idComercio: 2
importe: 125.0
codRespuesta:
idAutorizacion:

- -

Ejercicio 3

Para este ejercicio hemos creado primero un directorio como raíz de este proyecto, después hemos copiado el fichero P2.jmx generado en el apartado anterior, además de copiar la carpeta datagen que se nos entregaba en la práctica y las diferentes carpetas de las aplicaciones anteriores P1-base/P1-ws/P1-ejb-servidor-remoto/P1-ejb-cliente-remoto. Hemos incorporado el insert sql en todos los proyectos para que tengan la misma base datos y hemos creado un script para automatizar el proceso de repliegue y despliegue de la práctica:

```
#!/bin/bash
for i in P1-base P1-ws P1-ejb-servidor-remoto P1-ejb-cliente-remoto; do
    cd $i
    ant replegar; ant delete-pool-local
    cd -
done
cd P1-base
ant delete-db
cd -
for i in P1-base P1-ejb-servidor-remoto P1-ejb-cliente-remoto P1-ws; do
    cd $i
    ant limpiar-todo todo
    cd -
done
```

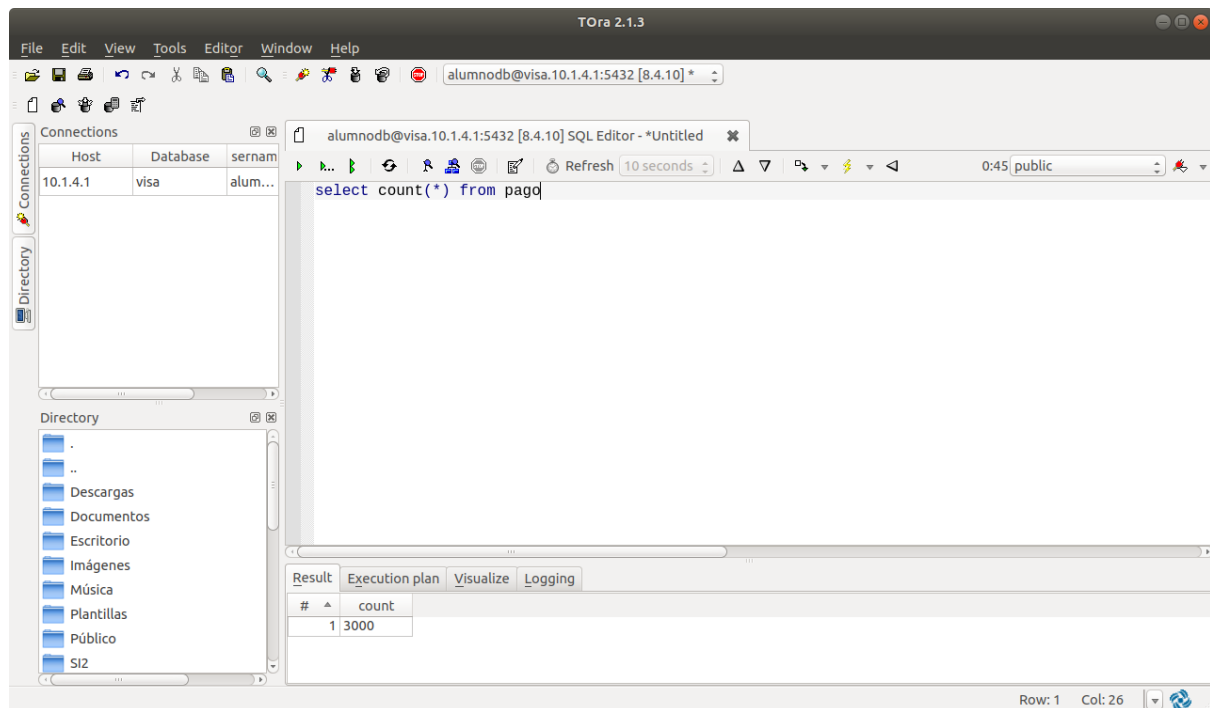
Para que funcionase con todas las aplicaciones por igual, como la aplicacion P1-ws no tenia el comando replegar, hemos creado uno en el build.xml que llama a replegar-servicio y replegar-cliente:

```
<target name="replegar">
    <antcall target="replegar-cliente" />
    <antcall target="replegar-servicio" />
</target>
```

Posteriormente, hemos abierto el fichero .jmx y hemos lanzado varias veces con el objetivo de eliminar ruido relacionado con procesos periódicos del sistema operativo, para observar el tiempo de las peticiones, el resultado en el “Aggregated Report” y el resultado ha sido el siguiente:

Etiqueta	# Muestras	Media	Mediana	90% Line	95% Line	99% Line	Min	Máx	% Error	Rendimiento	Kb/sec	Sent KB/sec
P1-base	1000	5	5	7	7	12	4	162	0,00%	103,3/sec	132,44	0,00
P1-ws	1000	39	37	48	52	61	31	375	0,00%	12,9/sec	16,61	0,00
P1-ejb	1000	11	10	14	15	22	7	286	0,00%	44,7/sec	58,35	0,00
Total	3000	18	10	41	44	56	4	375	0,00%	27,4/sec	35,35	0,00

Para comprobar que se ha realizado todo con éxito, hemos hecho dos comprobaciones, la primera es la utilización del comando nmon y la segunda es la consulta a través de Tora para ver que efectivamente se habían realizado 3000 pagos, además como vemos en la anterior imagen todas las columnas de error están al 0.00%:



¿Cuál de los resultados le parece el mejor? ¿Por qué? ¿Qué columna o columnas elegiría para decidir este resultado?

Tras realizar estas pruebas hemos observado que el rendimiento de la P1-base es el mejor de las tres aplicaciones, seguido posteriormente de P1-ejb y finalmente P1-ws. Estos resultados se reflejan también en la columna denominada “media”, que indica el tiempo medio de respuesta, por lo que a menor tiempo medio de respuesta mayor rendimiento y la clasificación anterior se mantiene. También debido a la columna “Throughput” que tiene un valor más alto que mide la productividad del servidor.

Posteriormente, realizamos las pruebas para el EJB local las cuales nos dan los siguientes resultados:



Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received K...	Sent KB/sec
P1-ejb	1000	4	4	5	5	13	2	639	0,00%	201,5/sec	261,24	0,00
Total	1000	4	4	5	5	13	2	639	0,00%	201,5/sec	261,24	0,00

Vemos que los resultados son mejores que los de P1-base, aunque en general la mayoría de columnas son bastante similares a esta última. Esto es debido a que estamos ejecutando las máquinas virtuales en el mismo ordenador, además está la base de datos en el mismo servidor.

En conclusión, vemos que P1-base y sobre todo P1-ejb-local son claramente superiores en rendimiento a las otras dos aplicaciones.

Por último se adjunta el server.log en la carpeta raíz de la entrega.

Ejercicio 4

Accedemos a través del navegador a la administración del sistema en 10.1.4.2:4848 y cambiamos los siguientes parámetros que nos piden en la práctica:

- Eliminamos la opción “-client” en “Configurations -> server-config -> JVM Settings -> Pestaña JVM Options”.
- Añadimos la opción “-server”.
- Comprobamos que está la opción “Xmx512m” y añadimos “Xms512m”.

<input type="checkbox"/>	-server
<input type="checkbox"/>	-Xmx512m
<input type="checkbox"/>	-Xms512m

- En la pestaña “Domain -> Pestaña Applications Configuration” desactivamos la opción de reload y auto deploy, y activamos el precompile.

Applications Configuration [Save]

Enable reloading so that changes to deployed applications are detected and the modified classes reloaded. Also enable and configure automatic deployment of applications. Click Add Property to specify additional settings.

[Load Defaults]

Reload: ☒ **Enabled**
Enables dynamic reloading of applications.

Reload Poll Interval: **Seconds**
Frequency for checking reload requests.

Admin Session Timeout: **Minutes**
A value of 0 means the session never times out.

Auto Deploy Settings

Auto Deploy: ☐ **Enabled**
Automatically deploys applications in the autodeploy directory.

Precompile:

☒ **Enabled**

Precompiles JSPs, deploys only resulting class files.

- Por último, activamos en HIGH los siguientes recursos en “Configurations -> server-config -> Monitoring”, dejando en OFF los otros:

Component Level Settings (16)		
<input checked="" type="checkbox"/>	Level: ▼	Change Level
Select	Module	Monitoring Level
<input type="checkbox"/>	Jvm	HIGH ▼
<input type="checkbox"/>	Transaction Service	OFF ▼
<input type="checkbox"/>	Connector Service	OFF ▼
<input type="checkbox"/>	Jms Service	OFF ▼
<input type="checkbox"/>	Security	OFF ▼
<input type="checkbox"/>	Web Container	HIGH ▼
<input type="checkbox"/>	Jersey(RESTful Web Services)	OFF ▼
<input type="checkbox"/>	Web Services Container	OFF ▼
<input type="checkbox"/>	Java Persistence	OFF ▼
<input type="checkbox"/>	Jdbc Connection Pool	HIGH ▼
<input type="checkbox"/>	Thread Pool	HIGH ▼
<input type="checkbox"/>	Ejb Container	OFF ▼
<input type="checkbox"/>	ORB (Object Request Broker)	OFF ▼
<input type="checkbox"/>	Connector Connection Pool	OFF ▼
<input type="checkbox"/>	Deployment	OFF ▼
<input type="checkbox"/>	Http Service	HIGH ▼

Lo siguiente es detener el servidor en la máquina virtual de 10.1.4.2 para obtener el fichero domain.xml, el cual se adjunta en la entrega, a través del siguiente comando en la máquina host:

```
scp si2@10.1.4.2:/opt/glassfish4/glassfish/domains/domain1/config/domain.xml ./
```

Lo siguiente ha sido revisar el fichero si2-monitor.sh para averiguar los siguientes parámetros.

1. Max Queue Size del Servicio HTTP:

```
francisco@francisco:~/SI2/P2$ asadmin --host 10.1.4.1 --user admin --passwordfile ./passwordfile get server.thread-pools.thread-pool.http-thread-pool.max-queue-size
server.thread-pools.thread-pool.http-thread-pool.max-queue-size=4096
Command get executed successfully.
```

Podemos comprobar que el comando se ejecuta correctamente y vemos que nos devuelve un tamaño de 4096 de Max Queue Size.

2. Maximum Pool Size del Pool de conexiones a nuestra DB:

```
francisco@francisco:~/SI2/P2$ asadmin --host 10.1.4.1 --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.
```

Podemos comprobar que el comando se ejecuta correctamente y vemos que el Maximum Pool Size devuelto es de 32.

3. El número de errores en las peticiones al servidor web:

```
francisco@francisco:~/SI2/P2$ asadmin --host 10.1.4.2 --user admin --passwordfile ./passwordfile monitor --type httplistener server
ec  mt  pt  rc
0   0   0.00 0
0   0   0.00 0
```

En este caso no utilizamos get y utilizamos monitor para conseguir las estadísticas de monitorización más comunes sobre el servidor 10.1.4.2. En este caso nos muestran 4

columnas, siendo la primera los errores y por tanto la que nos interesa. Si quisiéramos saber los elementos monitorizados tendríamos que usar el parámetro list en vez de monitor o get.

Ejercicio 5

Parámetro	Valor
Valor máximo del Heap de memoria en la JVM	512 Mb
Valor mínimo del Heap de memoria en la JVM	512 Mb
Máximo número de conexiones a procesar simultáneamente por el servidor web	5 conexiones
Tamaño máximo de la cola de conexiones pendientes de servicio	4096 conexiones
Máximo número de sesiones en el contenedor Web.	-1 (ilimitadas)
Máximo número de conexiones en pools JDBC	32 conexiones

Ejercicio 6

Lo primero fue replegar las aplicaciones P1-ws, P1-ejb-servidor-remoto y P1-ejb-cliente-remoto. Lo siguiente fue detener el servidor de aplicaciones de la MV1 y reiniciamos el dominio en la MV2 debido a los cambios realizados y en la monitorización. Debido a la lentitud del script de la monitorización de los recursos del servidor, decidimos ejecutar el script desde la MV2 y tuvimos que descomentar la línea del passwordfile en el script.

A continuación se muestran las imágenes correspondientes del nmon para el PC Host y la MV2 en los picos de utilización.

PC HOST:

```

mmon-16g [H for help] —Hostname=francisco—Refresh= 2secs —03:33.48
CPU Utilisation
+-----+
CPU User% Sys% Wait% Idle|0 |25 |50 |75 |100|
1 13.1 29.1 0.5 57.3|UUUUUUSSSSSSSSSSSS>
2 15.6 41.0 0.0 43.4|UUUUUUSSSSSSSSSSSSSS>
3 15.3 30.2 0.0 54.5|UUUUUUSSSSSSSSSSSS>
4 2.0 93.0 0.0 5.0|SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS>
+-----+
Avg 11.5 48.5 0.1 39.9|UUUUUSSSSSSSSSSSSSSSSSSSS>
+-----+
Memory and Swap
PageSize:4KB RAM-Memory Swap-Space High-Memory Low-Memory
Total (MB) 7804.5 1766.0 - not in use - not in use
Free (MB) 1565.0 1766.0
Free Percent 20.1% 100.0%
Linux Kernel Internal Memory (MB)
Cached= 4316.8 Active= 2385.5
Buffers= 197.4 Swpcached= 0.0 Inactive = 3153.7
Dirty = 2.1 Writeback = 0.0 Mapped = 1757.9
Slab = 268.3 Commit_AS = 8152.4 PageTables= 54.7
Network I/O
I/F Name Recv=KB/s Trans=KB/s packin packetout insize outsize Peak->Recv Trans
lo 0.0 0.0 0.0 0.0 0.0 0.0 2.0 2.0
vmnet1 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
vmnet8 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
wlp3s0 113.5 321.1 552.3 1663.8 210.5 197.7 125.6 330.7
enp2s0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Top Processes Procs=296-mode=3-1Base 3=Perf 4=Size 5=I/O[RootOnly] u=Args
PID %CPU Size Res Res Res Res Shared Faults Faults Command
Used KB Set Text Data Lib KB Min Maj
3762 102.8 2219692 873840 15836 119756 0 857448 393 0 vmware-vmx
3641 33.9 2145956 517468 15836 111384 0 501180 1 0 vmware-vmx
4543 26.4 5359272 367640 4 529252 0 29668 17 0 java
1946 10.0 3813660 238384 16 288748 0 124464 732 0 gnome-shell
1730 2.0 1935196 122428 2308 205448 0 84100 9 0 Xorg
1959 1.0 2125188 18552 84 28672 0 14136 85 0 pulseaudio
Warning: Some Statistics may not shown

```

MV2:

```

hmon-12f          Hostname=si2srv02      Refresh= 2secs      18:34.17
CPU Utilisation
+-----+
CPU  User%  Sys%  Wait%  Idle|0      |25      |50      |75      |100|
1   40.6    10.9   0.0    48.6|-----|-----|-----|-----|
2   64.2    4.2    0.0    31.6|-----|-----|-----|-----|
Avg 53.4    7.5    0.0    39.2|-----|-----|-----|-----|
+-----+

Memory Stats
+-----+
          RAM      High      Low      Swap
Total MB      749.2      0.0      749.2    150.0
Free MB        77.4      0.0      77.4    150.0
Free Percent   10.3%     0.0%     10.3%   100.0%
+-----+
          MB
          Cached=    179.7      Active=    519.8
Buffers=   16.4 Swapcached=    0.0 Inactive =   128.7
Dirty  =    0.2 Writeback =    0.0 Mapped   =   28.4
Slab    =   13.8 Commit_AS = 1323.3 PageTables=    1.8
+-----+

Network I/O
+-----+
I/F Name RecvKB/s TransKB/s packin packout insize outsize Peak->Recv Trans
lo      0.0      0.0      0.0      0.0      0.0      0.0      1.3      1.3
eth0     0.0      0.0      0.0      0.0      0.0      0.0      0.0      0.0
eth1  193.3    213.6    1221.0    792.9    162.1    275.9    205.3    227.9
+-----+

Top Processes Procs=94 mode=3 (1=Basic, 3=Perf 4=Size 5=I/O)
+-----+
PID      %CPU    Size      Res      Res      Res      Res      Shared      Faults      Command
      Used      KB      Set      Text      Data      Lib      KB      Min      Maj
1459  123.2    991256    469620    4      0    967268    18852    3741      0      0 java
4      0.5      0      0      0      0      0      0      0      0      0 ksoftirqd/0
1      0.0    2660    1592    104      0    384    1192      0      0      0 init
2      0.0      0      0      0      0      0      0      0      0      0 kthreadd
3      0.0      0      0      0      0      0      0      0      0      0 migration/0
5      0.0      0      0      0      0      0      0      0      0      0 watchdog/0
6      0.0      0      0      0      0      0      0      0      0      0 migration/1
7      0.0      0      0      0      0      0      0      0      0      0 ksoftirqd/1
8      0.0      0      0      0      0      0      0      0      0      0 watchdog/1
9      0.0      0      0      0      0      0      0      0      0      0 events/0
+-----+
Warning: Some Statistics may not shown

```

A continuación se muestran los resultados para la monitorización, se ve el número de muestras, el número de conexiones JDBC , el número de conexiones HTTP y HTTPQ.

```

si2@si2srv02:~$ bash si2-monitor.sh localhost
#Muestra numJDBCCount numHTTPCount numHTTPQ
0 0 0 0
1 0 0 0
2 0 1 0
3 1 1 0
4 1 1 0
5 0 1 0
6 0 1 0
7 1 1 0
8 0 0 0
9 1 1 0
10 0 1 0
11 0 1 0
12 0 1 0
13 0 1 0
14 0 0 0
15 0 1 0
16 1 1 0
17 0 1 0
18 0 1 0
19 1 0 0
20 0 0 0
^C
TOT.MUESTRAS 21 MEDIA: 0.285714 0.714286 0

```

Se aprecia que nunca se supera el máximo de una conexión tanto HTTP como en JDBC ya que solo hay un cliente y hace las peticiones de manera continua y secuencial.

¿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?

El elemento que más costoso parece si nos fijamos en las capturas en nmon, es tanto para el servidor como para el cliente la CPU. En cambio el uso de disco es bajo, y tampoco hay una gran carga de red al ser local y bastante rápida. El uso de memoria es continuo, pero no es muy elevado ya que al final solo tenemos un cliente simultáneo (como hemos podido ver en la monitorización).

¿Le parece una situación realista la simulada en este ejercicio? ¿Por qué?

No es una situación realista, ya que en el plan de pruebas solo hay un cliente (hilo) que realiza peticiones de manera secuencial. En una situación real y práctica, habrá un mayor número de clientes concurrentes y también las peticiones llegarán más desordenadas y con diferentes picos con mayor carga en ciertas situaciones y mayor descanso en otras. Además la red es local en el plan.

Teniendo en cuenta cuál ha sido el elemento más saturado, proponga otro esquema de despliegue que resuelva esa situación.

Una opción podría ser separar el acceso a la base de datos de este servidor, como en el caso de los EJB lo que reduciría la carga de trabajo. Otra opción sería la escalabilidad horizontal del servidor, aumentando el número de CPUs y cores para aumentar el rendimiento y reducir la saturación en cada procesador. Otra opción, sería no virtualizar la máquina en la que se ejecuta el servidor y dotarlo de todos los recursos físicos.

Ejercicio 7

En este ejercicio hemos preparado el script del Jmeter para su ejecución en el entorno de pruebas:

Server Name or IP: 10.1.4.2

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received K...	Sent KB/sec
/P1-base/c...	100	4	5	6	7	8	1	13	0,00%	2,1/sec	4,06	0,00
/P1-base/p...	100	12	13	15	16	34	4	75	0,00%	2,1/sec	2,58	0,00
Total	200	8	6	15	15	17	1	75	0,00%	4,0/sec	6,29	0,00

Vemos que el ejemplo funciona perfectamente y además se muestra una una captura del árbol de resultados:

The screenshot shows the JMeter interface with the 'Response data' tab selected. The left sidebar lists several samplers, with '/P1-base/procesapago' highlighted. The main panel displays the 'Response Body' of the selected sampler, which is an HTML document. The HTML content includes a DOCTYPE declaration, a title 'Sistema de Pago con tarjeta', and a body containing the text 'Pago con tarjeta'.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<html>
<head>
<title>Sistema de Pago con tarjeta</title>
<style type="text/css">
td.error {color:red;font-size:0.65em;}
</style>
</head>
<body>
<h1>Pago con tarjeta</h1>
```

Ejercicio 8

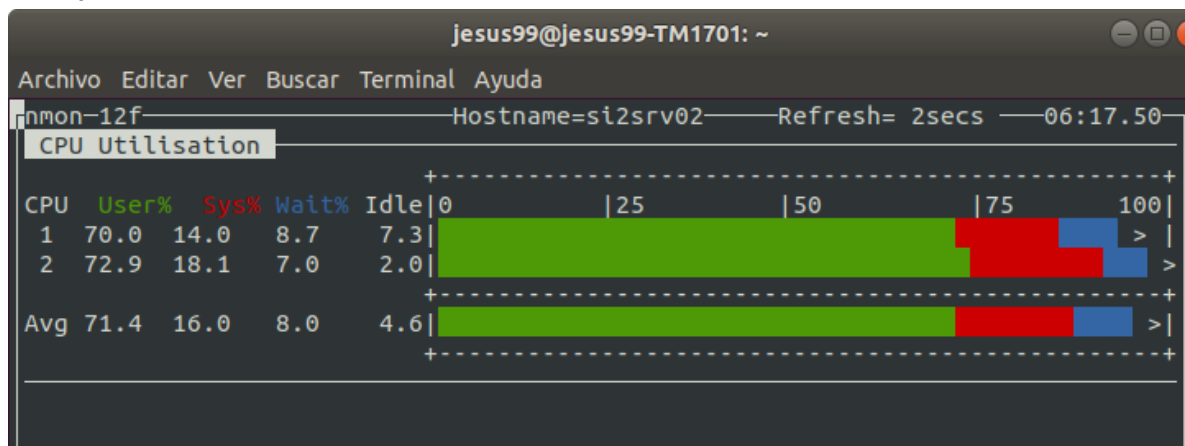
Para realizar el ejercicio hemos seguido los siguientes pasos:

- Hemos abierto el fichero P2-curvaProductividad.jmx en Apache Jmeter.
- Para cada una de las iteraciones hemos ido cambiando en 250 el número de hilos del grupo de hilos "Consultas".
- Hemos ejecutado el script de prueba.
- Antes de las iteraciones hemos lanzado tanto en el PC Host como en la máquina virtual 2 el comando "nmon -f -c 2 -s 2000 para después poder comprobar el uso de la CPU.
- Hemos ejecutado en cada una de las iteraciones el script si2-monitor.sh cuando han pasado 10 segundos desde el inicio de la prueba.
- Hemos guardado las **salidas** en la carpeta de datos del ejercicio 8 tanto de **Jmeter** como del **script de monitorización**.
- Hemos registrado los datos en la hoja de cálculo SI2-P2-curvaProductividad
- Durante la monitorización hemos ido revisando el uso de la CPU para detectar la saturación del sistema.

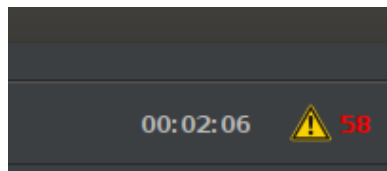
Se han rellenado los datos en el documento de cálculo para las medidas 1, 250, 500, 750, 1000, 1250, 1500, 1750, 2000, 2250, 2500 y 2750. Para justificar cada fila anterior, se ha adjuntado una carpeta llamada /data-ejer8. Dentro se puede encontrar el fichero nmon que justifica el uso de cpu, y también se encuentra una carpeta por cada toma anterior (1, 250, 500, ...). Dentro de dichas carpetas se encuentra el .csv del jmeter correspondiente a la prueba y un pantallazo del proceso de monitorización para cada número de usuarios.

Cuando llegamos a los 1750 hilos de usuarios observamos que el nivel de la CPU utilizada llega a más del 90% que era el límite que nos habíamos marcado para realización de las pruebas:

Se adjunta foto:

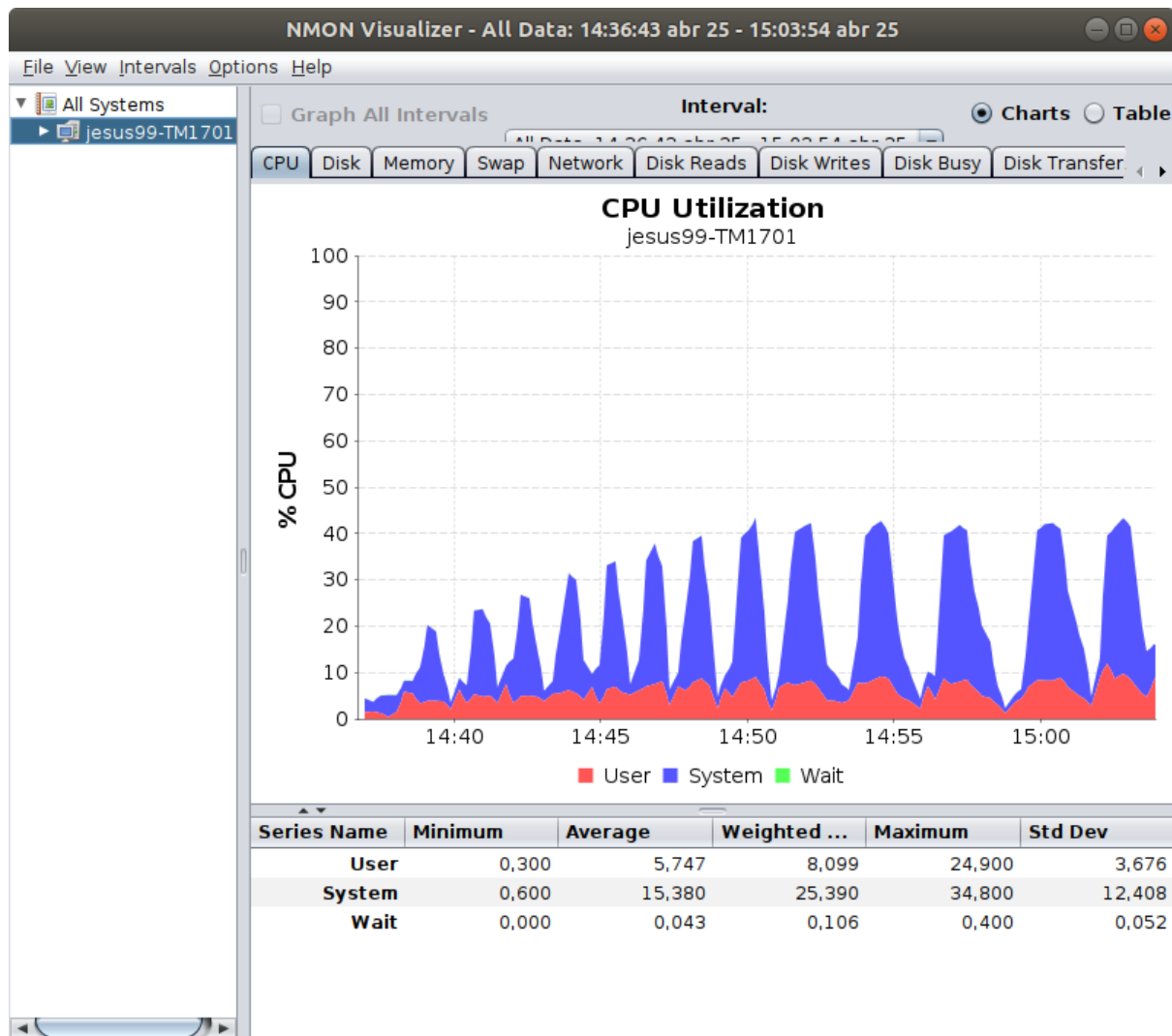


A partir de esta prueba observamos que se empiezan a dar errores de conexiones:

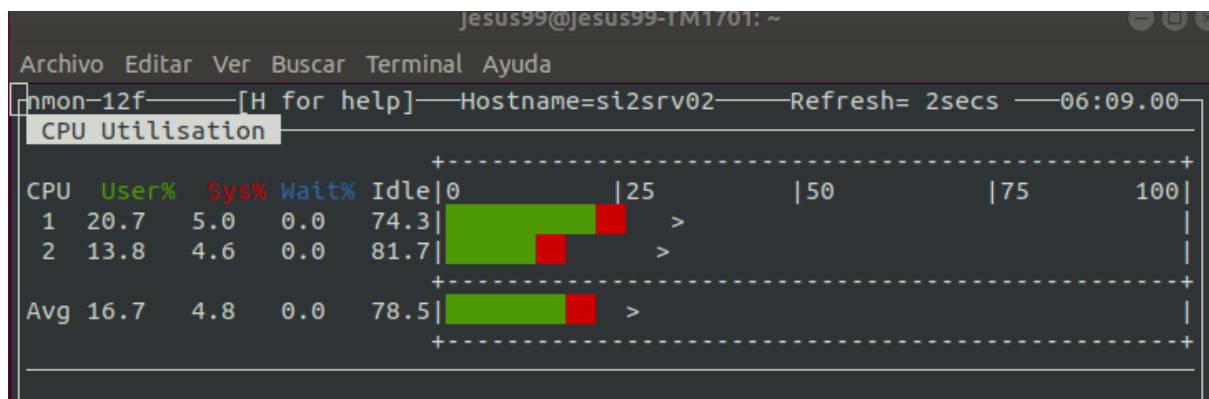


Además se adjuntan los resultados del Nmon "Data-collect" de ambas maquinas.

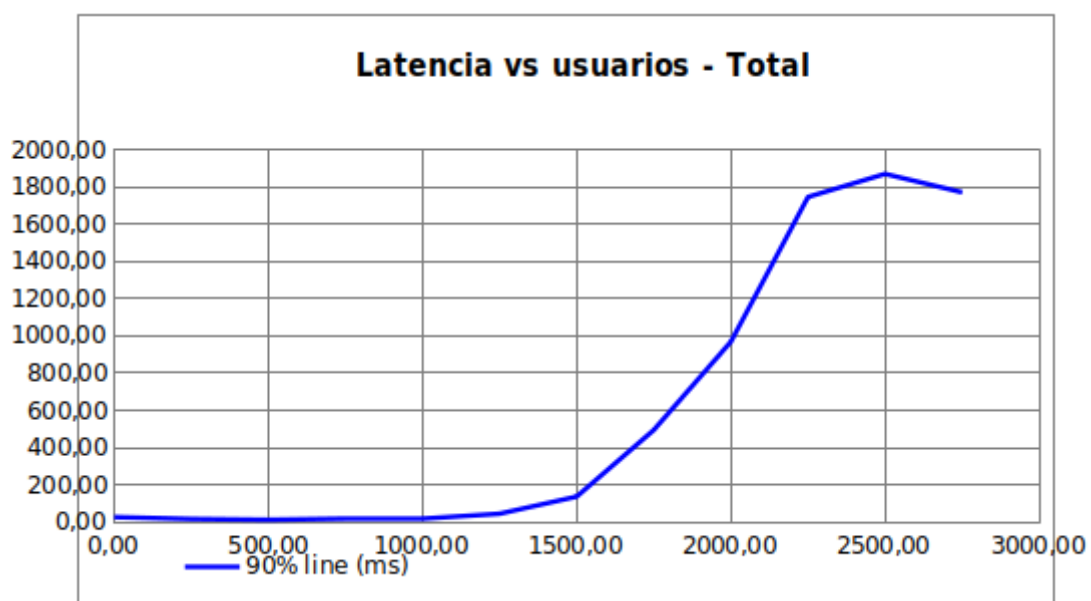
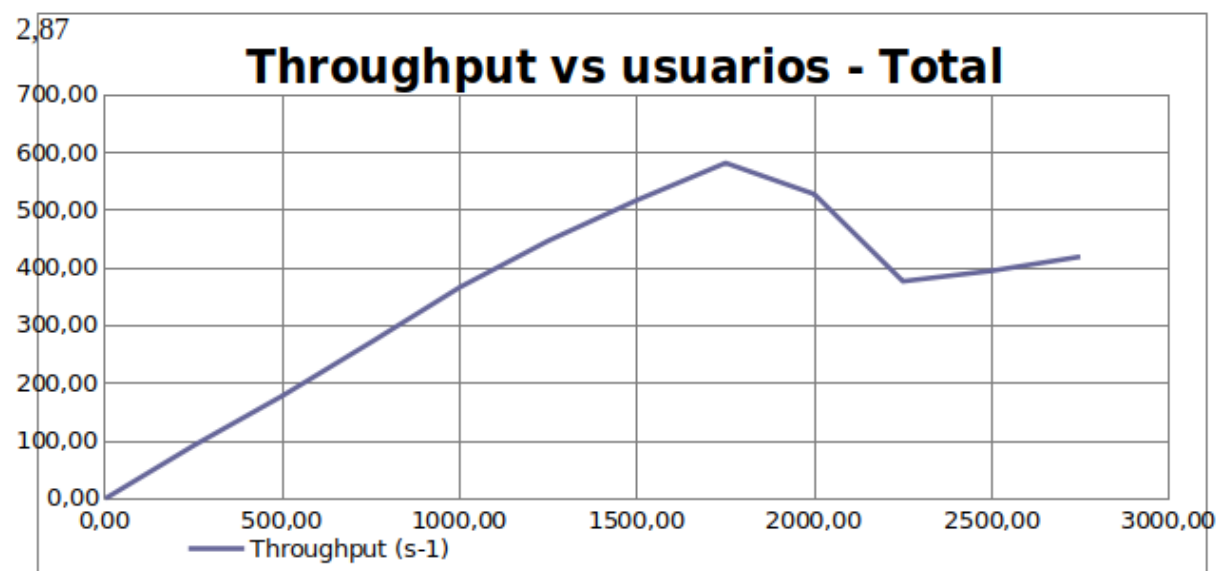
PC-HOST:



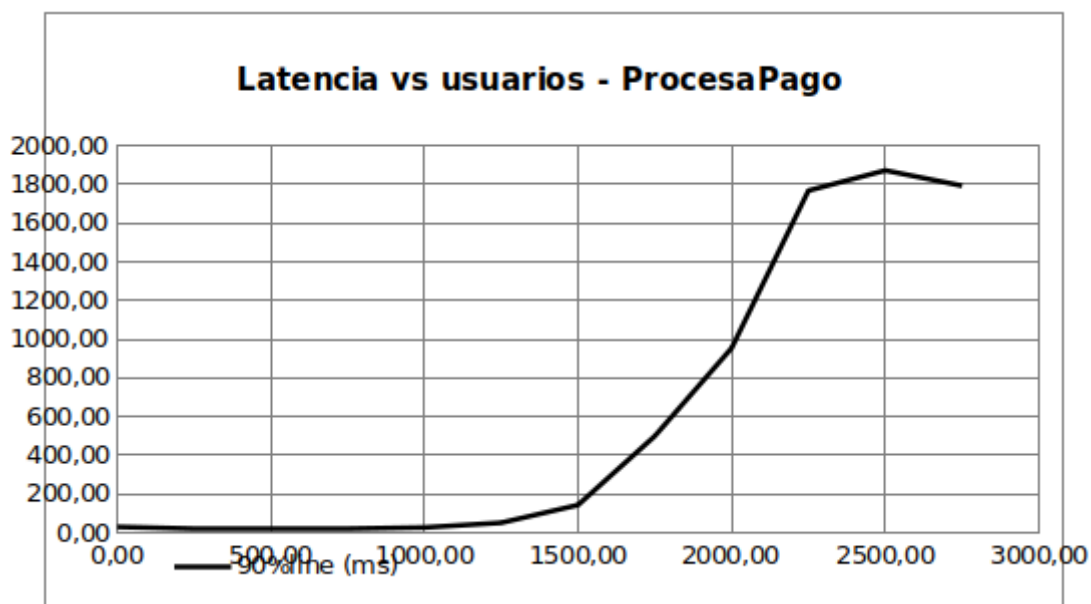
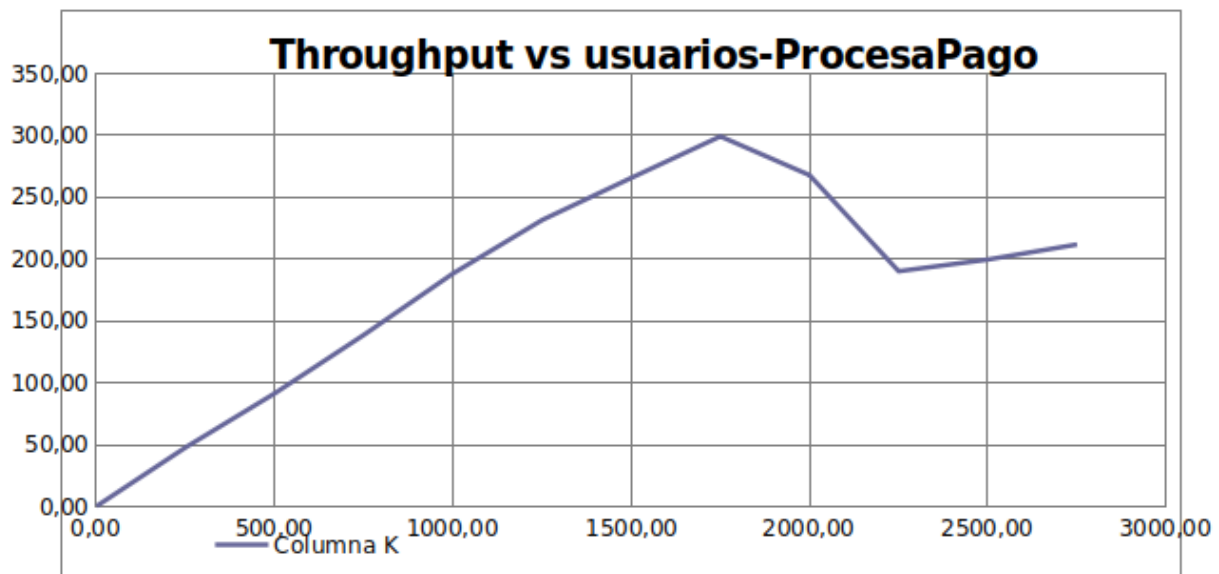
MV2:



Una vez hemos completado todos los datos en la hoja de cálculo obtenemos las siguientes curvas de productividad y latencia para el total:



Y las siguientes curvas para el procesaPago:



Ejercicio 9

Los datos que se han usado para este ejercicio son los datos recopilados en el anterior ejercicio. Los datos se encuentran en la carpeta /data-ejer8/<numero_de_usuario>.

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.

Se puede observar en las gráficas del ejercicio anterior que la saturación se produce en los 1750 usuarios, ya que es cuando se produce el descenso en el throughput del servidor y es

donde se produce el punto de inflexión en la curvatura de la latencia. El throughput que se alcanza en ese punto es el más alto que hemos obtenido con un total de 582. El throughput máximo que hemos obtenido es el anterior que es cuando empieza la zona de saturación, pero se observa en la curva que se empieza a estabilizar hasta obtener un total de 400.

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.

Como ha ocurrido en pruebas anteriores se puede ver que el recurso más usado es la CPU. Por lo tanto, si aumentamos el número de cores o procesadores, o la eficiencia de estos aumentaría el punto de saturación. También hay que tener en cuenta que al estar haciendo pruebas en el host, y virtualizando 2 máquinas, es normal que la CPU sea lo más usado.

Pero como escalar horizontalmente el sistema no es una opción para esta práctica, se puede apreciar en la monitorización de los recursos que solo hay un tamaño de 5 peticiones HTTP concurrentes.

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.

Hemos aumentado el número de máximos de hilo para las peticiones HTTP de 5 a un máximo de 8 peticiones o hilos. Volvemos a ejecutar las pruebas de la curva para un total de 1750 clientes que es el punto de saturación. Una vez cambiado el parámetro, reiniciamos el servidor y volvemos a correr las pruebas obteniendo los siguientes datos:

Etiqueta	# Muestras	Media	Mediana	90% Line	95% Line	99% Line	Min	Máx	% Error	Rendimiento	Kb/sec	Sent KB/sec
/P1-base/c...	17500	121	110	270	314	356	1	394	0,00%	310,6/sec	603,51	0,00
/P1-base/p...	17500	133	124	283	325	368	3	430	0,00%	305,0/sec	241,20	0,00
Total	35000	127	118	277	320	362	1	430	0,00%	597,4/sec	816,44	0,00

Hemos obtenido un rendimiento superior al obtenido anteriormente si nos fijamos en el tiempo medio de respuesta y en el rendimiento que es de 598 que es superior a los 582 obtenidos anteriormente. Si subimos a un mayor número máximo de hilos, obtendremos mejor rendimiento. Pero debido a que tenemos poca memoria ram en la MV2 donde ejecutamos el server hemos decidido no subir demasiados.