UAM UNIVERSIDAD AUTONOMA DE MADRID		**Escuela Politécnica Superior Ingeniería Informática Prácticas de Sistemas Informáticos 2**			
Grupo	**2402**	**Práctica**	2	**Fecha**	29/03/2019
Alumno/a		Gómez, Borzdynski, Óscar			
Alumno/a		Polanía, Bernárdez, Pablo Alejo			

Práctica 2: Rendimiento

Ejercicio número 1:

Nada que mencionar en la memoria. La configuración de *JMeter* ha sido llevada a cabo correctamente.

Ejercicio número 2:

Para este ejercicio, y el resto de la práctica, se ha decidido correr las máquinas virtuales en un único PC. Por tanto las direcciones que se asocian a los diferentes elementos del sistema son:

- PC físico: 10.10.0.157.
- PC1VM (máquina virtual 1): 10.2.5.1.
- PC2VM (máquina virtual 2): 10.2.5.2.

Siguiendo el esquema y las instrucciones del enunciado y las IP's elegidas se han modificado los ficheros *build.properties* de los proyectos *P1-base, P1-ws* y *P1-ejb-<cliente/servidor>-remoto*. Los campos modificados son los mismos que se han modificado en otras prácticas: *as.host.client, as.host.server* y *db.host*. Por último en el directorio *P1-ejb-cliente-remoto* se ha modificado la ip que aparece en el fichero *glassfish-web.xml* dejándola con el valor 10.2.5.1.

Tras el despliegue de estos proyectos se ha comprobado que los pagos funcionaban de la manera habitual.

Una vez comprobado se han ejecutado los comandos *free y nmon* en las distintas máquinas (física y virtuales). Los resultados se encuentran en la ruta *./evidencias/ej2* (los nombres de cada captura son intuitivos).

Tras ejecutar el comando *free* se muestra por terminal una tabla con 2 filas y 6 columnas. Las filas hacen referencia a memoria física y a memoria virtual. Las columnas (en orden) hacen referencia al total de memoria, a la usada, a la libre, a la compartida (usada por ficheros de sistema), a la usada por *buffers* del *kernel* y una estimación de la memoria disponible (sin tener en cuenta la virtual) para el inicio de nuevas aplicaciones o procesos. Las tres últimas columnas no hacen referencia a la memoria virtual. Todos estos datos vienen dados en kB. En las maquinas virtuales sale una fila más que representa la memoria usada y libre sin tener en cuenta la memoria usada por *kernel buffers* o por la *cache*.

Por otro lado, si ejecutamos el comando *nmon* (resultados en el mismo directorio) y accedemos a la sección de memoria (pulsando *m*) podemos ver que muestra una información muy parecida a la mostrada por el comando anterior. Además de mostrar la información de *free* también muestra el porcentaje de uso, el tamaño de página, el número de tablas de páginas, memoria mapeada… Además muestra la *high memory* y la *low memory* que representan la memoria de usuario y la memoria del *kernel* respectivamente. Como se ha mencionado en el enunciado y teniendo en cuenta la información que suministra, se ve que esta herramienta se centra en el análisis del uso de *RAM* de la máquina.

Con todo, estas dos herramientas sirven para estudiar y controlar el uso de memoria de cada máquina.

Ejercicio número 3:

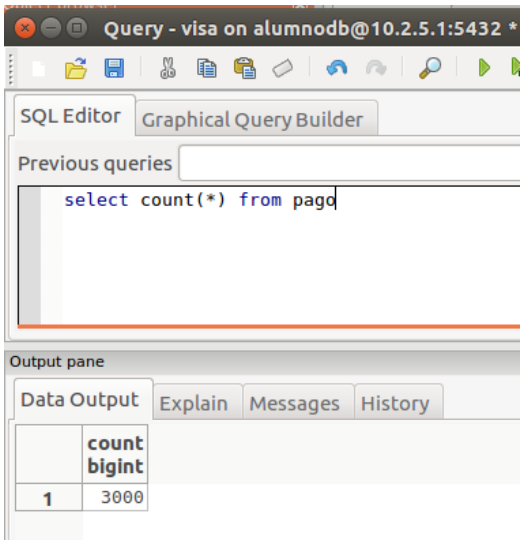
Tras preparar el directorio *P2,* ejecutar *JMeter,* borrar todos los pagos de la base de datos y asegurarnos de que el uso de *CPU* y el acceso a disco son bajos ejecutamos la prueba. El resultado que se recibe es:

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughp...	Received ...	Sent KB/s...
P1-ws	1000	111	103	138	155	206	89	814	0.00%	8.9/sec	11.58	0.00
P1-ejb	1000	30	28	42	47	63	19	77	0.00%	32.1/sec	42.10	0.00
P1-base	1000	13	12	16	18	69	9	161	0.00%	70.1/sec	89.95	0.00
TOTAL	3000	52	28	111	128	163	9	814	0.00%	19.1/sec	24.73	0.00

Se puede apreciar que no hay errores en las peticiones en ninguno de los proyectos (columna *Error %*). Esta tabla consta de 4 filas y (una por cada *Thread Group* y el *TOTAL*) de 12 columnas. Estas columnas han sido explicadas en el enunciado y las que más atención van a recibir en el desarrollo de esta práctica son:

- *Samples* y *Error %:* Para asegurarse que todas las peticiones se han llevado a cabo.
- *Average:* Indica el promedio del tiempo que se ha tardado en administrar cada petición. Este resultado se expresa en milisegundos (*ms*).
- *90% Line:* Esta columna indica el valor en milisegundos bajo el cual el 90% de las peticiones han sido administradas.
- *Throughput:* Indica la carga o productividad del servidor. Se mide en peticiones por segundo administradas.

Para asegurarnos de que los pagos se han realizado con éxito también podemos llevar a cabo una petición a la base de datos que nos muestre el número total de pagos guardados.



Se guarda el fichero *server.log* y el fichero *server-cliente.log* en la ruta *./evidencias/ej3.*

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

Según los resultados mostrados por *JMeter* se podría decir que el mejor servidor es el de *P1-base* que tiene la media (*Average*) más baja y la producción (*Throughput*) más alto. Estas dos columnas indican la velocidad media a la que se administra una sola petición y el número de peticiones atendidas en un segundo. También se podría tener en cuenta la columna *90% Line* ya que es importante asegurarse que no solo unas cuantas peticiones se administran rápido sino la mayoría de ellas. Si se tuviese que elegir una única columna para tener en cuenta sería la columna *Throughput* que representa el rendimiento del servidor.

Por último se desactiva la opción de ejecutar los *Thread Groups* de manera secuencial y se ejecuta el *P1-ejb* de manera local:

Label	# Sam...	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throug...	Receive...	Sent K...
P1-ejb	1000	22	12	27	37	84	7	5687	0.00%	44.0/sec	57.07	0.00
TOTAL	1000	22	12	27	37	84	7	5687	0.00%	44.0/sec	57.07	0.00

Se puede ver que al correrlo de manera local mejora notablemente respecto a la remota. Pasa de un *Throughput* de 32,1 peticiones por segundo a uno de 44. Esta mejora se refleja en la media también. Todas las capturas y ficheros relacionados con el ejercicio (excepto P2.jmx) se encuentran en la ruta *./evidencias/ej3.*

Ejercicio número 4:

Tras la configuración de la máquina virtual 2 se ha guardado el fichero *domain.xml* en la ruta *./evidencias/ej4*. Todas las capturas asociadas a este ejercicio se encuentran en el mismo directorio.

Revisar el script *si2-monitor.sh* e indicar los mandatos *asadmin5* que debemos ejecutar en el *Host PC1* para averiguar los valores siguientes, mencionados en el Apéndice 1, del servidor *PC1VM1*:

(Se ha ejecutado el comando en la máquina virtual 2 ya que, aunque el enunciado pida que se ejecute en la 1 creemos haber escuchado que se debía ejecutar para la 2. En todo caso los resultados son iguales para ambas máquinas)

Para los dos primeros comandos se usa la estructura de árbol de los recursos solicitados. Primero hay que fijarse en que el comando consta de dos partes. En la primera se especifica la IP a la que se quiere acceder y las credenciales para poder acceder a ella. La segunda parte indica la acción que se quiere llevar a cabo sobre el servidor al que se ha accedido (en nuestro caso coger el valor de una variable: *get*). Para indicarle la variable que queremos coger se sigue la estructura de árbol de la aplicación yendo desde la raíz hasta las hojas (o recursos). P. ej.: *server.grupo-de-elementos.sss.aaaa.nombre-del-recurso*.

1. Max Queue Size del Servicio HTTP:

```
oscar@oscar:~/SI2/P2$ asadmin --host 10.2.5.2 --user admin --passwordfile ./pass
wordfile 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.
```

Se aprecia que devuelve el valor esperado del tamaño máximo de cola: 4096.

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

```
oscar@oscar:~/SI2/P2$ asadmin --host 10.2.5.2 --user admin --passwordfile ./pass
wordfile get resources.jdbc-connection-pool.VisaPool.max-pool-size
resources.jdbc-connection-pool.VisaPool.max-pool-size=32
Command get executed successfully.
```

Se aprecia que devuelve el valor esperado del número máximo de hilos para conectarse a la DB: 32.

Para cambiar los anteriores valores a través de comandos sería cambiar el *get* por un *set* e indicar el valor que se le quiere dar.

3. Número de errores en las peticiones al servidor web:

```
oscar@oscar:~/SI2/P2$ asadmin --host 10.2.5.2 --user admin --passwordfile ./pass
wordfile monitor --type httplistener server
ec    mt    pt      rc
0     1553710744176 2517524.00 617159
```

Este último comando comparte la primera parte con los anteriores pero en la segunda, en vez de *get,* se pone *monitor* para conseguir las estadísticas de monitorización más comunes del servidor. Muestra una tabla con cuatro columnas de la cual nos interesa la primera *ec* (*error count*). Se puede apreciar que marca 0 errores.
En caso de querer saber los elementos que tienen activada la monitorización se puede hacer uso del comando *list* sustituyendo *get* o *monitor* en los anteriores comandos.

Ejercicio número 5:

Se ha accedido al servidor *glassfish* en modo administrador y se ha comprobado el valor de los elementos listados en el enunciado. Nada más que añadir.

Ejercicio número 6:

Se adjuntan los ficheros *.nmon* locales y de la máquina virtual 2 en la ruta *./evidencias/ej6*. En esa misma carpeta se encuentran capturas para corroborar las respuestas dadas en el ejercicio. Para la realización de este ejercicio se ha decidido utilizar la herramienta *NMONVisualizer*. Las capturas han sido tomadas desde esa herramienta. Se adjunta un fichero *monitor.txt* que representa los datos de la herramienta *si2-monitor.sh*

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 …).

En el caso de esta monitorización el recurso más utilizado ha sido el de la *CPU* como se puede comprobar comparando los distintos *.png* adjuntos. En el fichero *ej6 vm cpu.png* se puede apreciar que el uso de C*PU* entre el usuario y el sistema llega al 80% en su punto máximo. Por otro lado el acceso a disco es muy bajo y solo se producen ciertas escrituras mientras que el trafico de red tampoco es elevado. Esto tiene sentido teniendo en cuenta que la respuesta de la red en nuestra simulación es instantánea y no tiene ningún tipo de descanso. Esto implica que la *CPU* este continuamente en uso. El uso de memoria también es continuo debido a que las peticiones son secuenciales y no hay picos ni valles en el uso. Al ser un único cliente, el uso de este recurso es bajo. Se ve que es un único cliente con la herramienta *nmon.*

	#Muestra	numJDBCCount	numHTTPCount	numHTTPQ
2	0	0	0	0
3	1	0	0	0
4	2	0	1	0
5	3	0	1	0
6	4	1	1	0
7	5	1	1	0
8	6	0	2	0
9	7	0	1	0
10	8	0	1	0
11	9	0	1	0
12	10	0	1	0
13	11	0	1	0
14	12	0	1	0
15	13	0	1	0
16	14	0	0	0
17	15	0	1	0
18	16	0	1	0
19	17	0	1	0
20	18	0	1	0
21	19	1	1	0
22	20	0	0	0
23	^C			
24	TOT.MUESTRAS	MEDIA:		
25	21	0.142857	0.857143	0

Se ve que el número de hilos, el número de elementos en cola, y el número de conexiones a la base de datos no es superior a 1 o 2 en ningún caso. Esto se debe a que solo hay un cliente y las peticiones se hacen de manera secuencial con el mismo intervalo de tiempo.

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

No se trata de una situación realista ya que en esta simulación la respuesta del servidor y de la red es instantánea. Además en una situación realista habría más de un solo cliente y las peticiones no serían secuenciales, sino que dependerían del cliente. Estos dos factores generarían picos en el número de peticiones alternado con valles y periodos en los que no se hiciese ninguna petición. Por todo ello se considera que no es una situación realista.

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

Debido a que la causa de saturación en el sistema ha sido la *CPU* se piensa que se podría optar por un sistema de despliegue *EJB* en el que el servidor no se encarga del proceso entero de administración de las peticiones sino que está repartido con la base de datos. Otra opción sería suministrar más recursos *CPU* para que no hubiese una situación de saturación por esta característica.

Ejercicio número 7:

Se ha comprado el correcto cambio del *script* llevando a cabo una prueba. Captura en *./evidencias/ej7*. Nada más que añadir.

Ejercicio número 8:

Se han ejecutado las distintas pruebas para la recogida de datos con número de usuarios igual a: 1, 250, 500, 600, 675, 750, 825, 925, 1000, 1100, 1200, 1250, 1350 , 1425, 1500 y 2000. Para la recogida de datos se ha utilizado la herramienta *si2-monitor.sh* y la herramienta *vmstat*, además del *aggregate report* de *JMeter.* En la carpeta *./evidencias/ej8* se adjuntan todas las capturas pertinentes que respaldan los datos que se han introducido en las tablas de *SI2-P2-curvaProductividad.ods*. Este último fichero generará las tablas de *Throughput vs usuarios* tanto para *ProcesaPago* como para *Total*.

Ejercicio número 9:

Los archivos que respaldan las siguientes respuestas se encuentran en la carpeta *./evidencias/ej8/users-<numero_usuarios>* ya que fueron recopilados en el ejercicio anterior.

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.

Para calcular el punto de saturación se debe calcular el punto en el que se cruzan las rectas de la zona lineal y la línea de saturación. Mirando las gráficas generadas y tomando valores se ve que la zona lineal llega hasta los 600 usuarios. Tomando valores, la recta resultante es (con x siendo los usuarios e y el *throughput*):

- *Y = 0'33X*

En el caso de la línea de saturación se elige el valor más alto de *throughput* y se traza una recta paralela al eje x. Por tanto la recta de saturación es de al forma:

- *Y = 292'9*

Para el cálculo de ambas se han seguido la tabla y la gráfica de *Total*. Al intersecar ambas rectas se deduce el punto de saturación con valor (887'6, 292'9). Esto indica que el sistema debería implementarse para que no superase los 887 usuarios en ninguna situación.

Para diferenciar las zonas se han estudiado las gráficas. Sobretodo para este punto se ha mirado la de *Latencia vs usuarios* que muestra cundo se cambia de fase al ser más claro el cambio de la pendiente de la gráfica. Con todo las diferentes zonas quedan de la siguiente manera:

- Zona lineal: entre 0 y 600 usuarios.
- Zona de transición: entre los 600 y los 1100 usuarios.
- Zona de saturación: a partir de los 1100 usuarios. En cierto punto la productividad decrece debido a que ciertas características del sistema empieza a fallar. Esto se produce sobre los 1350 usuarios.

El máximo *throughput* alcanzado en la zona de saturación es el usado para calcular la recta de saturación, es decir, con 1350 usuarios y con un valor igual a 292'9 peticiones por segundo.

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 en pruebas anteriores se comprueba que se necesita más recursos de *CPU*. Se puede ver gracias a la herramienta *vmstat*. Las columnas correspondientes a *cpu* muestran que el sistema va aumentando el uso de está para administrar las peticiones pero llega un momento en el que no puede superar el valor 3000. Se puede apreciar que alcanza valores cercanos a este en la captura *vmstat-600.png*. A partir de ahi estos valores no aumentan probando así que no se pueden dedicar más recursos de este tipo a los procesos. Por tanto para aumentar los usuarios en el punto de saturación se deberían destinar más recursos a la *CPU* para administrar el sistema. Como apunte, hay que tener en cuenta que estas pruebas han sido ejecutadas en un único *PC* físico y dos virtuales en él. Esto puede explicar por qué el recurso de la *CPU* es el primero en saturarse.

Respecto a la configuración de *glassfish* (servidor de aplicaciones) nos hemos dado cuenta de que solo procesa 5 peticiones *HTTP* a la vez debido al tamaño máximo y mínimo del *Thread Pool*. Para el siguiente ejercicio se propone aumentar el número de hilos.

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.

Se propone igualar el campo *Thread Pool size* entre 5 y 8. La prueba se lleva cabo con 925 usuarios (punto de saturación tiene 887'6 usuarios). Se ve que el throughput no solo supera al valor anterior con 925 usuarios (271'2 peticiones por segundo) sino que se asemeja al mejor valor obtenido en la gráfica total (292'9). En esta prueba se ha obtenido un *throughput* de 292'5 peticiones por segundo. Esto prueba que al aumentar el tamaño del *pool* de hilos se administran las peticiones más rápido y la aplicación es más eficiente. Las evidencias se encuentran en la carpeta *./evidencias/ej9* con los nombres *tipoprueba-925-thread-pool-8.x* donde *tipoprueba* es *aggregate, vmstat* o *monitor* y *x* es *png* o *csv*. En todo caso aumentar el número de hilos demasiado sería contraproducente, ya que al cambiar la *cpu* entre hilos produce retrasos lo que conlleva a una disminución del *throughput*.