

Práctica 2: Rendimiento

Índice

1	OBJETIVOS.....	3
2	ENTORNO	3
3	MATERIAL ENTREGADO	3
4	RENDIMIENTO EN APLICACIONES J2EE.....	4
4.1	INTRODUCCIÓN	4
4.2	DETALLES DE RENDIMIENTO DEL SERVIDOR DE APLICACIONES.....	4
4.3	ALGUNOS PARÁMETROS QUE AFECTAN AL RENDIMIENTO	6
4.3.1	<i>Tamaño del thread pool.....</i>	6
4.3.2	<i>Dimensionamiento del pool de conexiones JDBC.....</i>	6
5	ESTRATEGIA DE PRUEBAS	6
6	MEDICIÓN DE RENDIMIENTO CON JMETER	7
6.1	ASPECTOS GENERALES	7
6.2	INICIANDO JMETER	8
6.3	AÑADIENDO PARÁMETROS POR DEFECTO	8
6.4	AÑADIENDO UNA VARIABLE DE USUARIO	9
6.5	AÑADIENDO UN GRUPO DE HILOS	10
6.6	AÑADIENDO UNA VARIABLE ALEATORIA	11
6.7	AÑADIENDO UNA VARIABLE CONTADOR.....	12
6.8	AÑADIENDO UN CONJUNTO DE DATOS ALEATORIOS	12
6.9	GENERANDO PETICIONES HTTP.....	13
6.10	MIDIENDO RESULTADOS	15
6.11	COMPLETANDO LA DEFINICIÓN DEL PLAN DE PRUEBAS	16
7	PRIMERA PRUEBA: EJECUCIÓN ITERATIVA DE LAS TRES VERSIONES.....	17
7.1	DESPLIEGUE DE LAS APLICACIONES	17
7.2	EJECUCIÓN	19
7.3	DEPURANDO LA PRUEBA ANTERIOR.....	21
8	MONITORIZACIÓN	21
8.1	HERRAMIENTAS DE MONITORIZACIÓN	22
8.2	MONITORIZACIÓN DEL SERVIDOR	22
8.3	MONITORIZACIÓN JAVA	23
8.4	PARÁMETROS A MONITORIZAR EN LA PRÁCTICA	25
8.5	CONFIGURAR EL SERVIDOR DE APLICACIONES CON PARÁMETROS CORRECTOS DE RENDIMIENTO Y MONITORIZACIÓN	25
8.6	REGISTRAR LOS PARÁMETROS DE CONFIGURACIÓN DEL SERVIDOR.....	26
9	SEGUNDA PRUEBA: EJECUCIÓN ITERATIVA DE P1-BASE CON MONITORIZACIÓN.....	27
9.1	REINICIAR EL SERVIDOR DE APLICACIONES	27
9.2	INICIAR HERRAMIENTAS DE MONITORIZACIÓN DEL SERVIDOR	27
9.3	CAMBIOS EN EL JMX	27
10	SATURACIÓN DE UN SERVIDOR: LA CURVA DE PRODUCTIVIDAD.....	28
11	TERCERA PRUEBA: DETERMINACIÓN DE LA CURVA DE PRODUCTIVIDAD DE P1-BASE.....	29
11.1	PREPARAR EL CLIENTE PARA LA PRUEBA DE RENDIMIENTO	29

11.2	OBTENCIÓN DE LA CURVA DE PRODUCTIVIDAD	31
11.3	EJECUCIÓN DE LA PRUEBA	31
11.4	INTERPRETACIÓN DE LOS RESULTADOS	32
12	ENTREGA.....	33
13	BIBLIOGRAFÍA RECOMENDADA	33
	APÉNDICE 1: ASPECTOS GENERALES DEL RENDIMIENTO EN APLICACIONES J2EE.....	34
13.1	APLICACIÓN.....	34
13.2	JAVA VIRTUAL MACHINE (JVM)	34
13.3	CONFIGURACIÓN DEL HARDWARE Y SISTEMA OPERATIVO	34
13.4	SERVIDOR DE APLICACIONES.....	35
13.4.1	<i>Servicio HTTP.....</i>	35
13.4.2	<i>Web Container.....</i>	36
13.4.3	<i>EJB Container.....</i>	36
13.4.4	<i>Recursos JDBC</i>	36
14	APÉNDICE 2: OBTENCIÓN DE LA CURVA DE RENDIMIENTO EN ENTORNOS DE PRUEBAS	38

1 Objetivos

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

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

Para lograr estos objetivos, ampliaremos el servicio de pago VISA introducido en la práctica anterior para que trabaje con un *backend* PostgreSQL donde consultará un listado de tarjetas válidas y posteriormente almacenará la información de los pagos realizados.

2 Entorno

El entorno de realización de la práctica es el mismo que se ha empleado en las prácticas anteriores:

- Servidor de aplicaciones Glassfish V4.0
- Base de datos PostgreSQL

Así mismo, en esta práctica introduciremos el uso de:

- Apache JMeter como herramienta para la realización de las pruebas.

3 Material entregado

En esta práctica se entregará un fichero P2-alumnos.tgz con el siguiente contenido:

- Directorios **datagen/** y **sql/** que incluyen una nueva carga de datos (insert.sql) y fichero equivalente tabulado de tarjetas para test (listado.csv)
- Hoja de cálculo para anotar los parámetros del servidor de aplicaciones y mostrar la curva de productividad (**SI2-P2-curvaProductividad.ods**)
- Un script de pruebas para emplear en la última parte de la práctica (**P2-curvaProductividad.jmx**)
- Script de monitorización (**si2-monitor.sh**)

4 Rendimiento en aplicaciones J2EE

4.1 Introducción

El rendimiento de aplicaciones J2EE es un tema complejo, y crítico en muchos casos. La cadena de procesamiento de las peticiones a un sistema que sigue esta arquitectura es larga, recorre muchos elementos de procesamiento y cada uno de ellos posee numerosos parámetros de configuración, la mayoría de los cuales afectan al rendimiento.

Por otro lado, no se puede establecer una configuración *por defecto* que satisfaga las necesidades básicas de un gran número de aplicaciones o grupos de aplicaciones. Cada aplicación, con sus propias características de funcionamiento y requerimientos de usuario final, tendrá sus necesidades específicas que es preciso tener en cuenta a la hora de ponerla en producción. El argumento tópico de diseño que consiste en configurar el sistema *a lo grande*, con valores extremos, suele producir los efectos contrarios a los buscados: se invierten recursos del sistema donde no son necesarios, y no hay capacidad disponible allí donde realmente se requiere.

En este tema se va a realizar un análisis básico del rendimiento, presentando cómo estudiarlo sobre una aplicación en un entorno de pruebas, cómo medirlo, y cómo modificar parámetros del servidor de aplicaciones para analizar su efecto en el sistema global. Más que enseñar a los alumnos a resolver el problema del rendimiento, lo cual requeriría un tiempo y dedicación que queda totalmente fuera del ámbito de estas prácticas, se pretende mostrar a los alumnos el problema, para concienciarse de él, poder tener el conocimiento básico que les permita dialogar con un especialista en caso de que sea necesario abordar estos asuntos en el desarrollo de su vida profesional, o dar los primeros pasos para que el propio alumno se convierta en uno de estos especialistas.

Por último, este tema entrará sólo en el ámbito del rendimiento en el servidor de aplicaciones y los elementos que lo componen. El resto de los elementos de la arquitectura son vitales también para lograr este objetivo. Aunque en la bibliografía recomendada del tema se puede encontrar información detallada sobre las mejores prácticas para la mejora del rendimiento en la arquitectura J2EE, en el Apéndice 1 se ofrece, a modo de guía, algunos de los puntos más importantes que se deben considerar.

4.2 Detalles de rendimiento del servidor de aplicaciones

Los servidores de aplicaciones, como se ha explicado en las clases de teoría y como se ha ido viendo a lo largo de estas prácticas, constan de un conjunto de elementos que proporcionan el entorno adecuado para realizar aplicaciones J2EE. En el caso de Glassfish, la arquitectura concreta se presenta en la siguiente figura:

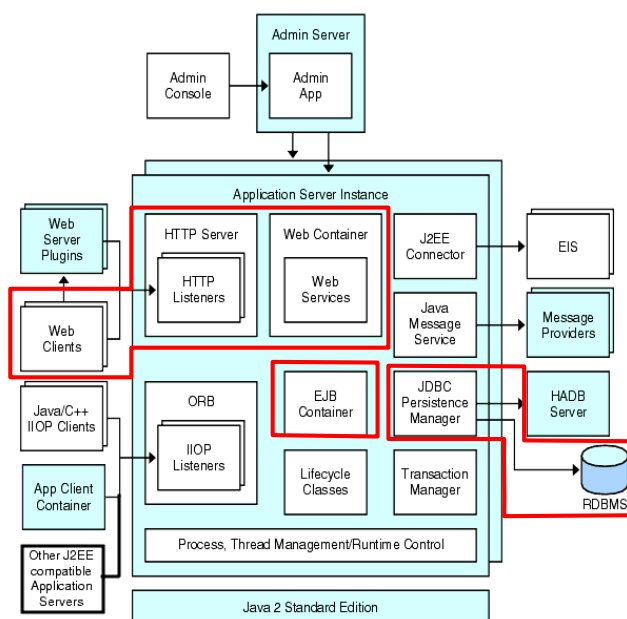


Figura 1: Elementos de arquitectura JAVA EE que se emplearán en esta práctica

Nos vamos a centrar en la cadena de procesamiento de las aplicaciones vistas durante las prácticas. En este caso, y de una manera muy simplificada, los elementos que entran en juego son los siguientes:

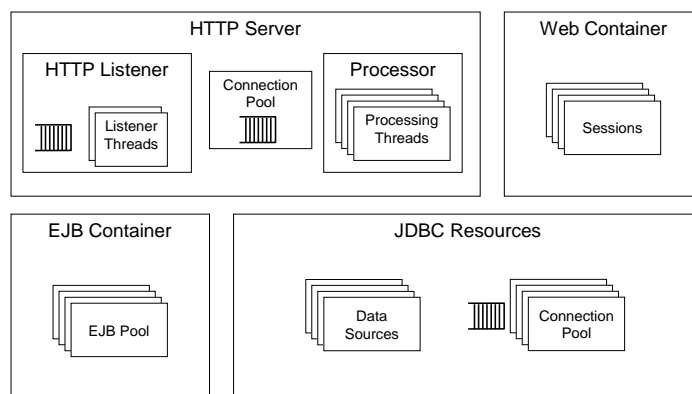


Figura 2: Elementos de la cadena de procesamiento que se emplearán en esta práctica

Los ejemplos de configuración que se presentan en este enunciado están en inglés. En esta lengua, los nombres de los parámetros de configuración que aparecen en la pantalla coinciden con los reales de los parámetros en los archivos de configuración. Por este motivo, se ha preferido evitar traducciones que en muchos casos sólo llevan a confusiones.

En una instalación como la de los laboratorios, con Glassfish en inglés, trabajando directamente sobre el servidor creado por defecto en la instalación, la consola de administración presenta la mayoría de los parámetros de configuración seleccionando en el panel de navegación: **Configurations -> server-config**.

4.3 Algunos parámetros que afectan al rendimiento

En el apéndice 1 se puede encontrar la lista de los principales parámetros de rendimiento que impactan en el servidor de aplicaciones. Algunos de los más importantes son los siguientes.

4.3.1 Tamaño del thread pool

La siguiente figura muestra un ejemplo de pantalla de configuración en la consola de administración de Glassfish. Concretamente, es la pantalla de configuración de los parámetros que regulan las unidades de proceso del pool de threads HTTP empleado para las aplicaciones de propósito general instaladas en el servidor (**http-thread-pool**). Permite definir el tamaño inicial del pool, incremento del mismo, *timeout* de las conexiones, etc.

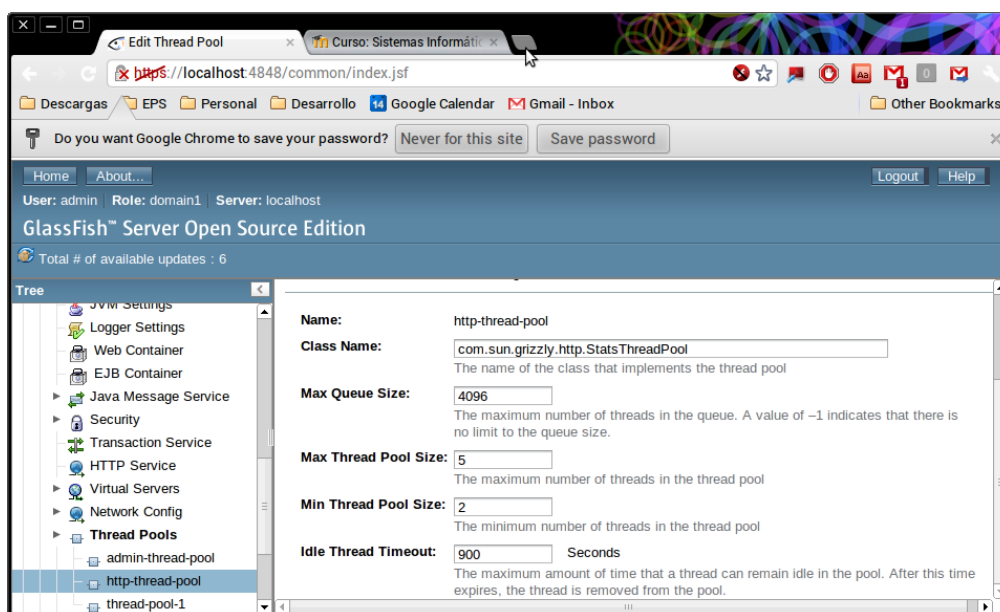


Figura 3: Propiedades del pool de threads

4.3.2 Dimensionamiento del pool de conexiones JDBC

Visto ya en la práctica anterior, estos parámetros permiten perfilar el volumen de conexiones esperado por nuestra aplicación, para ajustarlo en la medida de lo posible a la realidad. Se accede en *Resources->JDBC->JDBC Connection Pools->Nombre del pool*

5 Estrategia de pruebas

En los siguientes pasos procederemos a crear un plan de pruebas para nuestra aplicación, cuyo cometido a grandes rasgos será:

- Realizar 1000 peticiones iterativas **en un sólo hilo** al servidor con datos aleatorios de tarjetas sobre los que realizar pagos, de importes comprendidos entre 1 y 1000. Mediremos los **tiempos de respuesta** y nos quedaremos con el tiempo de corte en el 90% de las muestras, es decir, ignoraremos los tiempos de pico. También tomaremos una medición básica del rendimiento (*throughput*).
- Realizar 1000 peticiones en **varios hilos** al servidor, simulando cada hilo un usuario real de la aplicación. Con esta trataremos de visualizar la **curva de productividad** con el fin de identificar cuál es el **punto de saturación** de nuestra aplicación, cuál es el número máximo de usuarios concurrentes que no debe sobrepasar si no queremos saturar su funcionamiento.

6 Medición de rendimiento con JMeter

6.1 Aspectos generales

JMeter es una herramienta de test de carga, pruebas funcionales y medición de rendimiento de aplicaciones web, bases de datos y otros recursos.

En este apartado vamos a definir un plan de pruebas (*Test Plan*) que nos permita medir y comparar el rendimiento de las diferentes opciones de despliegue del sistema de pagos descritas hasta ahora.

Importante: No emplee todavía el plan de pruebas proporcionado P2-curvaProductividad.jmx, el cual se empleará en la última parte de la práctica: inicie un proyecto vacío. Todos los pasos de este apartado deberán completarse partiendo de cero.

Un plan de pruebas constará de varios elementos organizados en forma de árbol. El árbol se ejecuta de arriba abajo, en una secuencia de pruebas.

Los diferentes elementos disponibles (a los cuales nos referiremos por sus nombres en inglés de ahora en adelante) se pueden añadir de forma gráfica. De ellos utilizaremos únicamente el siguiente subconjunto:

- **Thread Group (grupo de hilos):** Simula un conjunto de usuarios concurrentes que accederán a nuestra aplicación. En el ejemplo de esta práctica únicamente usaremos *thread groups* con un único hilo (un usuario). Todos los *samplers* (muestreadores) que creemos deberán estar incluidos dentro de un *thread group* (ver más adelante).
- **Config Element (elemento de configuración):** Define un conjunto de parámetros que pueden ser reutilizados en diferentes grupos de hilos.
- **Sampler (muestreador):** Es un elemento cuyo cometido es generar una petición a un servidor. La tarea de recoger la respuesta queda relegada a un *listener* (escuchador) que veremos a continuación.
- **Listener (escuchador):** Recoge la respuesta de la petición anterior, y realiza algún tipo de cálculo o representación de la misma para su posterior análisis. Los *listeners* pueden ser locales al grupo de hilos o globales al plan de pruebas.

IMPORTANTE: Durante la creación del plan de pruebas es necesario **seguir las instrucciones en orden**. Así mismo deberemos grabar con frecuencia el proyecto. Se generará un fichero con extensión .jmx

A lo largo de la práctica se trabajará con las prácticas P1-base, P1-ejb (en su versión de EJB remoto) y P1-ws realizadas en la Práctica 1. La práctica P1-base por defecto despliega la aplicación con el nombre P1, en lugar de P1-base. Para cambiarlo deberá replegar la aplicación, si la tuviera desplegada, editar el fichero `build.properties` de P1-base y cambiar la línea `nombre=P1` por `nombre=P1-base`. A continuación deberá empaquetar y desplegar de nuevo la aplicación (ejecute `ant todo`).

6.2 Iniciando JMeter

El paquete de JMeter no está instalado en los laboratorios, pero se encuentra en la máquina virtual. Para comenzar a usarlo es necesario descargarlo (<http://ftp.cixug.es/apache/jmeter/binaries/apache-jmeter-4.0.tgz>) y descomprimirlo previamente. Ejecute desde el directorio personal:

```
tar -xzf apache-jmeter-4.0.tgz
```

Una vez descomprimido, se puede ejecutar así:

```
apache-jmeter-4.0/bin/jmeter.sh
```

6.3 Añadiendo parámetros por defecto

En lo que sigue, el plan de pruebas se ha renombrado como *P2 test* en lugar de *Test Plan*.

Con el fin de facilitar la migración del plan de pruebas a diferentes PCs, añadiremos un elemento de configuración denominado “*HTTP Request Defaults*”. Este elemento recoge los valores por defecto que se emplearán en todos los generadores de peticiones que veremos a continuación.

Deberemos pulsar sobre el conjunto de pruebas, botón derecho, *Add* (añadir) → *Config Element* (elemento de configuración) → *HTTP Request Defaults* (Valores por defecto de HTTP)

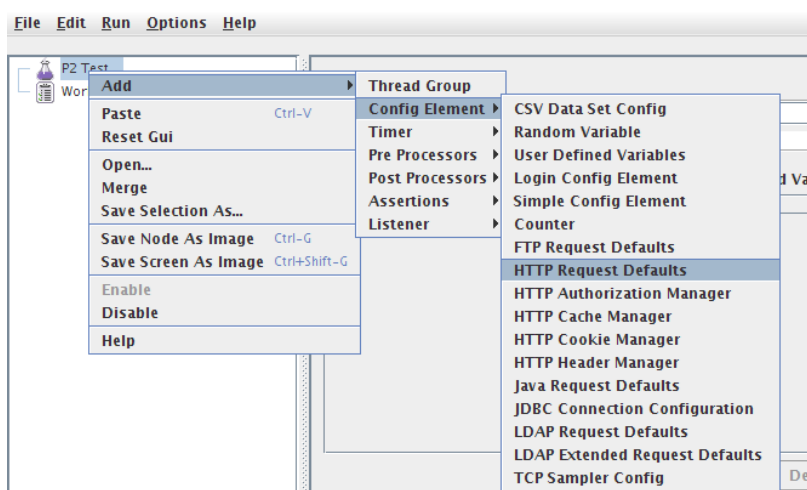
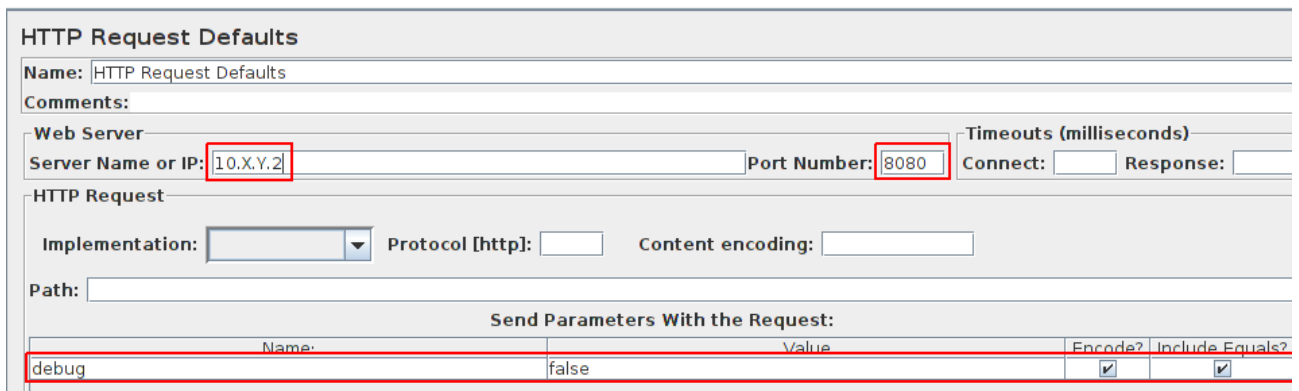


Figura 4: Añadiendo el elemento de configuración HTTP por defecto

Tras añadir este elemento, dispondremos de un cuadro de diálogo como el que se muestra a continuación, del cual únicamente definiremos la dirección IP de PC2VM (10.X.Y.2), el puerto del servidor (8080) y los parámetros que en principio no cambian entre pruebas, como por ejemplo *debug* (inicialmente a *false*, aunque luego podremos cambiarlo).



Name	Value	Encode?	Include Equals?
debug	false	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Figura 5: En los valores por defecto especificamos la IP y parámetros adicionales

6.4 Añadiendo una variable de usuario

Lo siguiente que añadiremos es un conjunto de variables de usuario. Éstas tienen una sintaxis similar a las propiedades de Java, (fichero .properties), y podremos asignarles un valor aquí reutilizándolo posteriormente como \${variable}.

En particular la variable que queremos añadir es una denominada “samples” que nos permitirá indicar el número de muestras que queremos tomar en todas las pruebas.

Para añadirlo, pulsaremos sobre el *Test Plan*, botón derecho *Add* → *Config Element* → *User Defined Variables*.

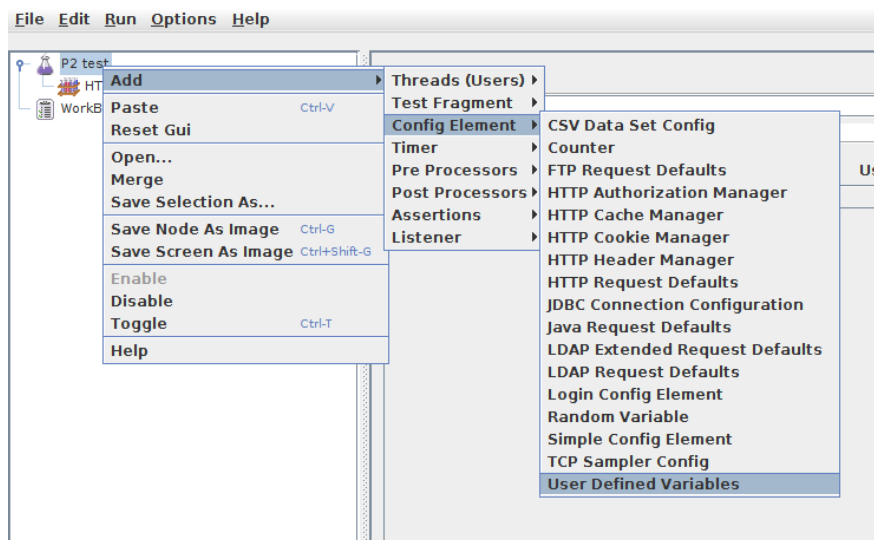


Figura 6: Añadiendo una variable de usuario

Posteriormente, en el diálogo de las variables a añadir, incluiremos la nueva “samples” con un valor de 1000, tal y como se indica en la figura.

User Defined Variables

Name: User Defined Variables

Comments:

User Defined Variables	
Name:	Value
samples	1000

Figura 7: La nueva variable “samples” define el número de repeticiones de la prueba a realizar

6.5 Añadiendo un grupo de hilos

A continuación introduciremos un *thread group* (grupo de hilos) que simulará el conjunto de usuarios que accede a la aplicación. En esta primera prueba, simularemos un único usuario, por lo que como veremos más adelante, nuestro “grupo de threads” en realidad tendrá un único thread.

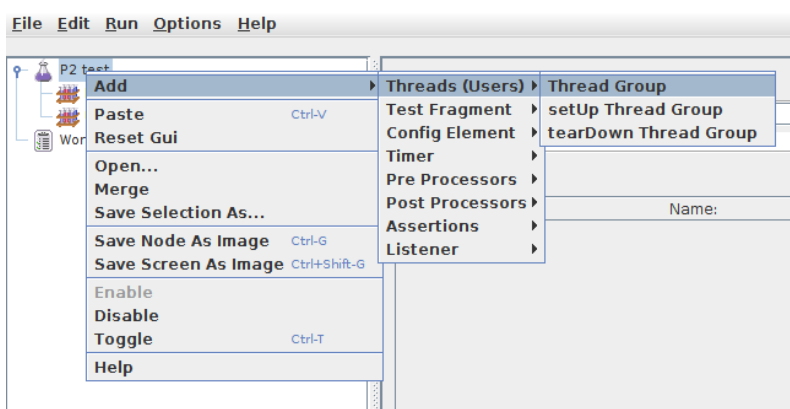
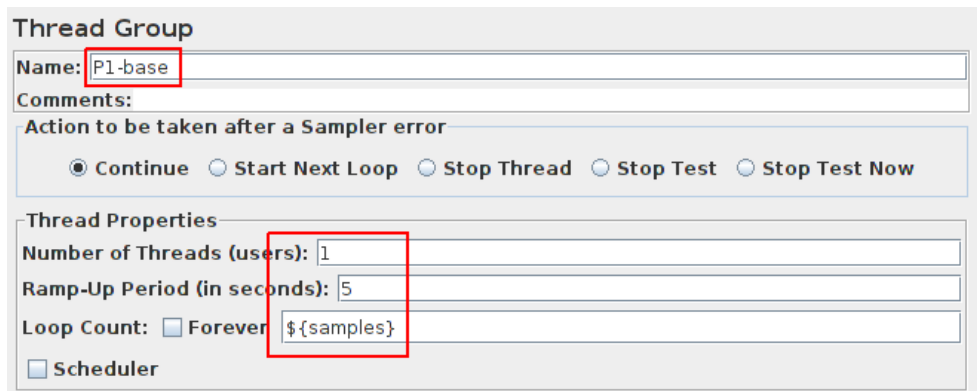


Figura 8: Añadiendo un grupo de hilos

Para añadirlo, pulsaremos sobre el *Test Plan*, botón derecho *Add* → *Threads (Users)* → *Thread Group*.

Especificaremos:

- *Name*: Introduciremos el nombre del conjunto para la primera versión de la aplicación a probar, P1-base
- *Number of threads*: Introduciremos el valor 1.
- *Ramp-up period (seconds)*: Indica el “tiempo de calentamiento”, que permitirá lanzar el número de *threads* de forma progresiva. Como sólo hemos definido un hilo, es irrelevante, pero en pruebas multihilo permite evitar la saturación inicial por el hecho de que lleguen todas las peticiones simultáneamente.
- *Loop count*: Indica el número de repeticiones de cada hilo. Introduciremos el valor de la variable anteriormente definida, \${samples}



Thread Group

Name: P1-base

Comments:

Action to be taken after a Sampler error

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

Thread Properties

Number of Threads (users): 1

Ramp-Up Period (in seconds): 5

Loop Count: ☐ Forever ☐ Sample Count: 5

☐ Scheduler

Figura 9: Especificamos los parámetros del grupo de threads

6.6 Añadiendo una variable aleatoria

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 éste como la variable "importe" para uso en adelante. Para ello, sobre el grupo de hilos P1-base, botón derecho *Add* → *Config Element* → *Random Variable*, y configurar los parámetros de acuerdo a la Figura 11.

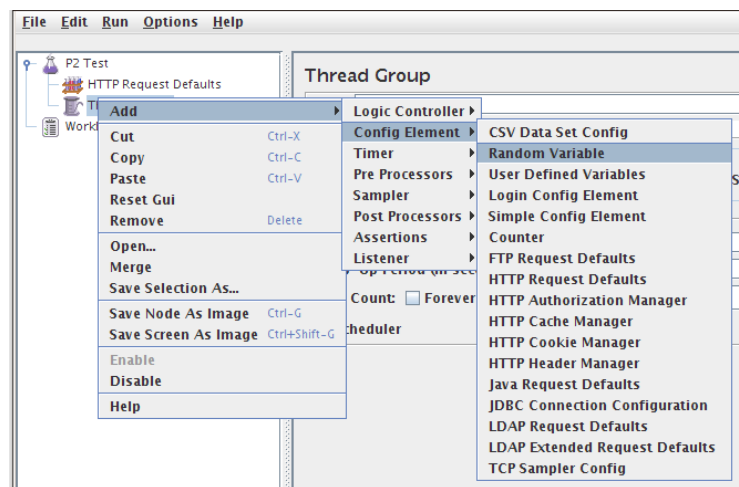
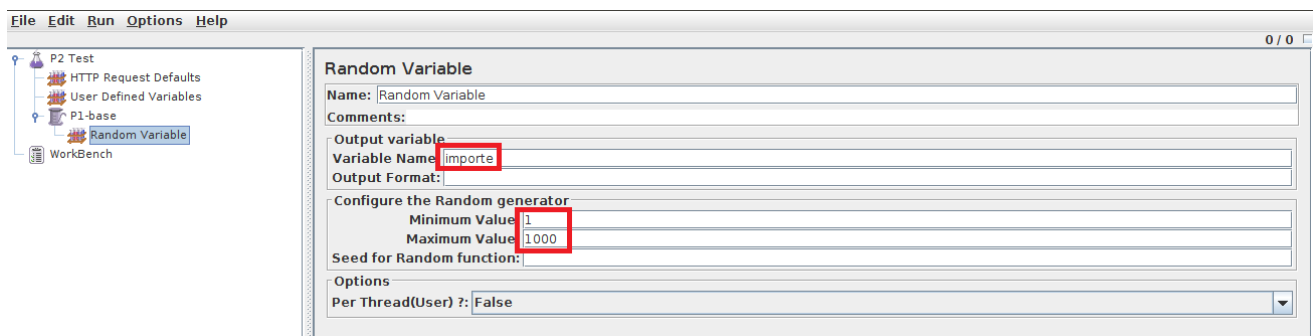


Figura 10: Elemento de configuración para variable aleatoria



Random Variable

Name: Random Variable

Comments:

Output variable

Variable Name: importe

Output Format:

Configure the Random generator

Minimum Value: 1

Maximum Value: 1000

Seed for Random function:

Options

Per Thread(User)?: ☐ False

Figura 11: La variable de *importe* estará comprendida entre 1 y 1000

6.7 Añadiendo una variable contador

El identificador de la transacción es un parámetro clave que no se puede repetir en cada pago de la tienda. Vamos a “simular” la generación de este identificador desde JMeter, para proporcionárselo a la página de pagos. Para ello, sobre el grupo de hilos P1-base, botón derecho, *Add* → *Config Element* → *Counter*.

Para este primer contador utilizaremos valores iniciados en 1 e incrementados de 5 en 5.

Para ello rellenaremos los campos: *Start* (valor 1), *Increment* (valor 5) y *Reference Name* (idTransaccion)

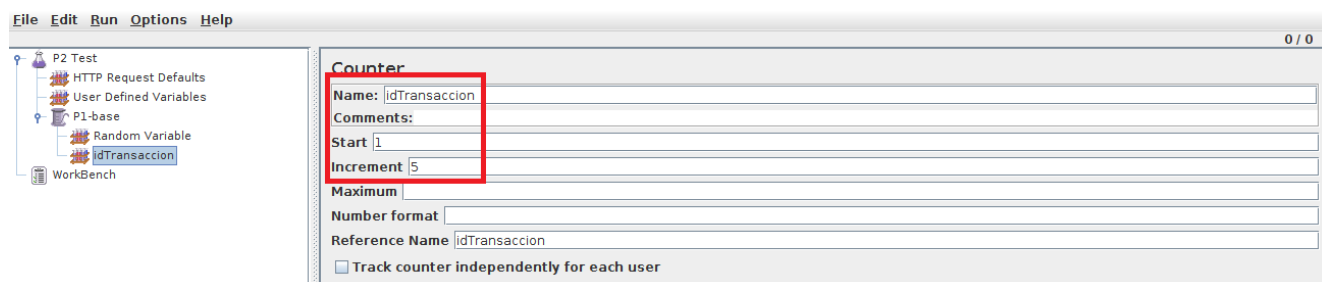


Figura 12: Variable de contador

6.8 Añadiendo un conjunto de datos aleatorios

El resto de los datos del pago (número de tarjeta, titular, fecha de emisión, caducidad y código de verificación de la tarjeta) los tomaremos aleatoriamente de un archivo de datos .csv. Se proporciona un fichero de ejemplo (datagen/listado.csv). Este fichero contiene filas de datos separados por el carácter |.

Para añadirlo, pulsamos sobre el grupo de hilos P1-base, botón derecho, *Add* → *Config Element* → *CSV Data Set Config*.

Los archivos CSV pueden ser editados con cualquier editor de texto, o editados con hojas de cálculo como OpenOffice.org Spreadsheet o Excel.

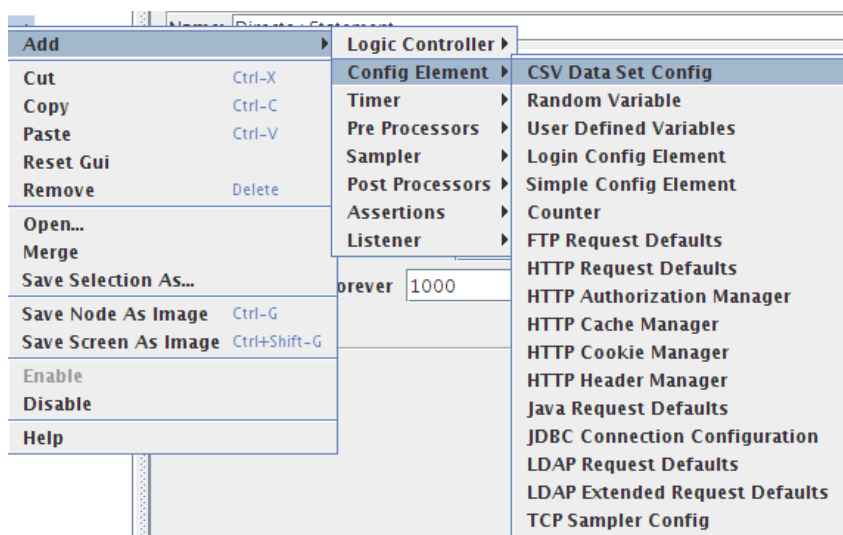


Figura 13: Añadiendo elemento de configuración de conjunto de datos

A continuación especificaremos la ruta relativa al archivo, el nombre de las variables que se definirán para cada campo separadas por comas y el carácter de separador (|). **El fichero datagen/listado.csv deberá existir en ruta relativa al fichero .jmx. Se proporciona un archivo de prueba en P2-alumno. Este fichero coincide con las tarjetas que serán insertadas en la creación de la base de datos.**

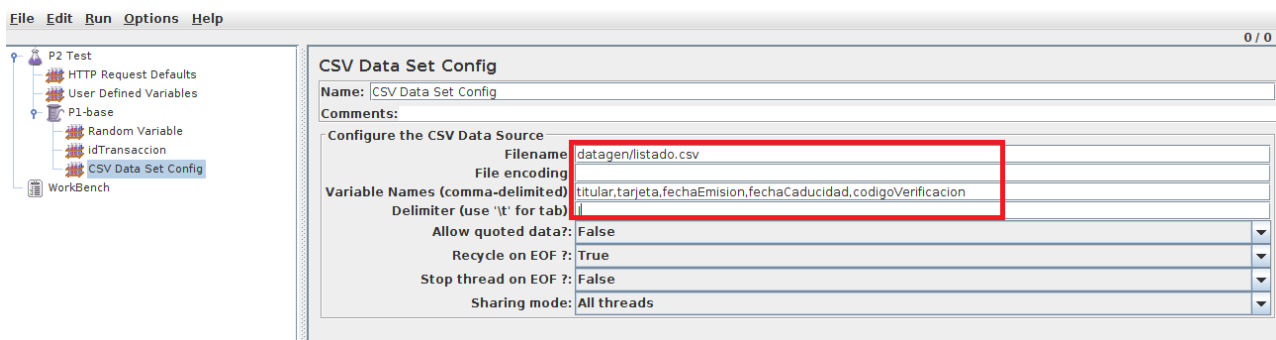


Figura 14: En el conjunto CSV especificamos el mapeo a variables

6.9 Generando peticiones HTTP

El elemento más importante de la prueba es un generador de peticiones HTTP. Lo añadiremos pulsando sobre el grupo de hilos P1-base, botón derecho, *Add→Sampler→HTTP Request*.

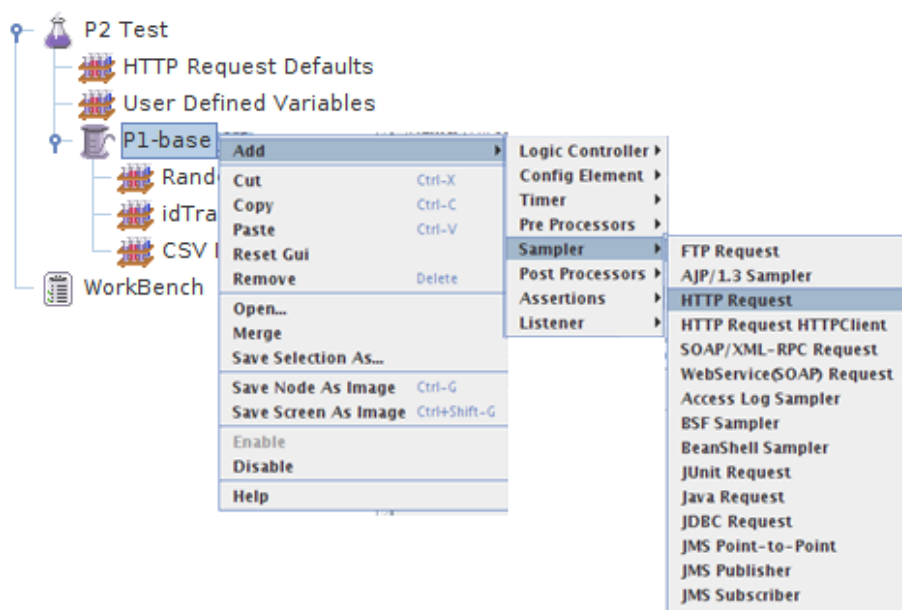


Figura 15: Añadimos un muestreador HTTP para simular peticiones

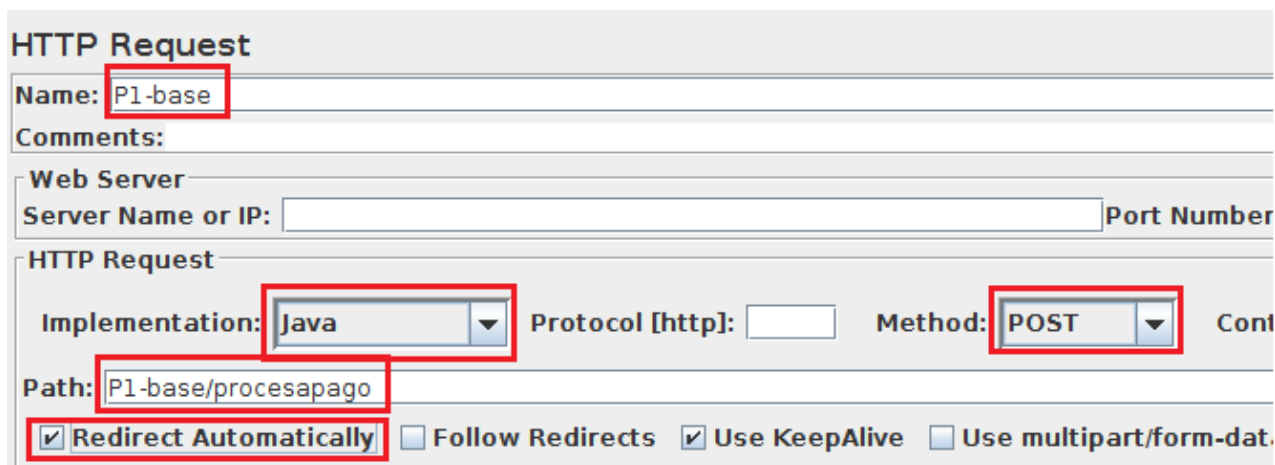


Figura 16: Establecemos la dirección de la petición y parámetros tomados de variables

Deberán definirse los siguientes parámetros (aquellos que no queden definidos se tomarán del elemento *HTTP Request Defaults* introducido anteriormente):

- **Name** (nombre), que debe ser único para este generador de peticiones. Este aspecto es fundamental, ya que los elementos que definiremos a continuación dependen de que se establezcan nombres distintos a cada *sampler* existente en el proyecto. En este primero, introduciremos **P1-base**
- **Path** (ruta) al *servlet* que realizará este primer conjunto de peticiones: **P1-base/procesapago**
- **Send parameters with the request.** Los siguientes parámetros de petición por POST (en todos ellos marcaremos la casilla “*Encode*”).

Parámetro	Valor
numero	\${tarjeta}
titular	\${titular}
fechaEmision	\${fechaEmision}
fechaCaducidad	\${fechaCaducidad}
codigoVerificacion	\${codigoVerificacion}
importe	\${importe}
directConnection	false
usePrepared	true
idTransaccion	\${idTransaccion}
idComercio	1

Figura 17: Parámetros de la petición

Obsérvese que los valores aleatorios tomados del CSV o de cualquier variable, los introduciremos en la forma `${variable}`. También introducimos algunos valores constantes, como por ejemplo “directConnection” / “usePrepared” / “idComercio”.

Es importante que, en todos ellos, se respeten las minúsculas y mayúsculas.

6.10 Midiendo resultados

De entre los diferentes *listeners* (escuchadores) que existen, añadiremos al **plan de pruebas** el “*Aggregate Report*” (informe agregado) que presenta información muy útil acerca de la medición. También son interesantes *Graph Results* o *Summary Report*, que dejaremos para más adelante.

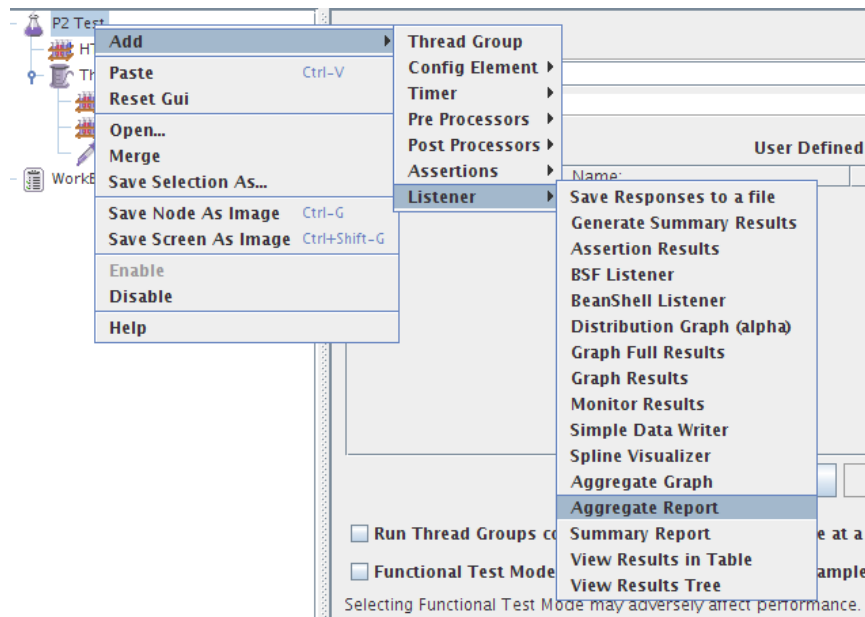


Figura 18: Añadimos un elemento Informe Agregado al plan de pruebas

En el informe agregado se pueden encontrar las siguientes columnas:

Aggregate Report

Name: Aggregate Report

Comments:

Write results to file / Read from file

Filename

Browse...

Log/Display Only: ☐ Errors ☐ Successes

Configu

Label	# Samples	Average	Median	90% Line	Min	Max	Error %	Throughput	KB
TOTAL	0	0	0	0	0	92233720...	0.00%	.0/hour	

Figura 19: El informe agregado presenta tiempos medios, mínimos, máximos y valores de *throughput*

- **Label (etiqueta):** Indica el nombre del *sampler* que se ha utilizado para la medición. Por tanto es fundamental que en adelante cada generador de peticiones HTTP tenga un nombre distinto.
- **# Samples:** Número de operaciones.
- **Average (media):** Indica el tiempo de respuesta medio, en milisegundos.
- **Median (mediana):** Indica el tiempo de respuesta correspondiente a la mediana, en milisegundos.

- **90% line:** Indica el tiempo para el percentil 90, es decir, por debajo del cual se encontraron el 90% de las muestras.
- **Min:** Valor mínimo.
- **Max:** Valor máximo.
- **Error %:** Porcentaje de respuestas que han recibido un código HTTP de error.
- **Throughput:** Rendimiento (número de muestras por minuto).

6.11 Completando la definición del plan de pruebas

Hasta el momento hemos construido un plan de pruebas que consta de un solo grupo de hilos que realizan una de las pruebas, así como el escuchador de informe agregado.

A continuación deberemos generar el resto de los casos mediante Copiar y Pegar. Para ello, elegiremos el grupo de hilos, lo copiaremos (botón derecho → *Copy*) y luego lo pegaremos (botón derecho → *Paste*). Se nos preguntará en qué orden queremos insertar la copia (antes o después del elemento seleccionado), e iremos repitiendo el procedimiento.

En total generaremos los casos para realizar las siguientes pruebas:

- **Pruebas contra P1-base/procesapago**
- **Pruebas contra P1-ws-cliente/procesapago**
- **Pruebas contra P1-ejb-cliente-remoto/procesapago**

Importante: Todos los elementos deberán permanecer inalterados en cada copia del grupo de hilos, salvo los siguientes

- **El nombre del generador de peticiones (*Http Request*)**, que deberá ser P1-base, P1-ws, P1-ejb.
- **El parámetro *path* del generador de peticiones HTTP** deberá coincidir con cada uno de los anteriormente indicados para las aplicaciones P1-base, P1-ws, P1-ejb (EJB remoto).
- Se deberá emplear un ***idTransaccion*** que **comenzará en un valor distinto** (1, 2 y 3) para evitar colisiones.
- Se empleará un **identificador de comercio (*idComercio*) distinto** de los demás (1, 2 y 3)

JMeter por defecto, si no se le indicara lo contrario, ejecutaría concurrentemente los tres casos de prueba.

Para conseguir que éstos se ejecuten **de forma secuencial**, deberemos pulsar sobre el plan de pruebas y marcar **“Run thread groups consecutively”**.

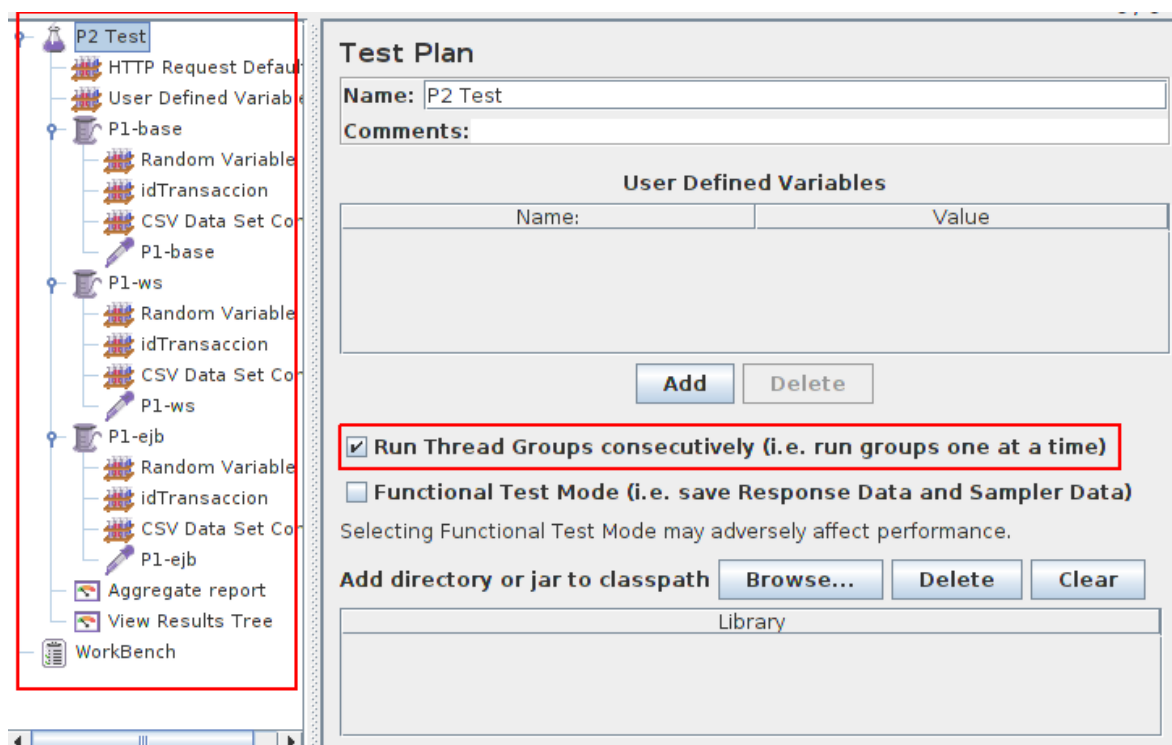


Figura 20: Aspecto final del plan de pruebas

Ejercicio 1: Siguiendo todos los pasos anteriores, defina el plan completo de pruebas para realizar las tres ejecuciones secuenciales sobre los tres proyectos definidos hasta ahora (P1-base, P1-ws, P1-ejb). Adjunte el fichero generado P2.jmx al entregable de la práctica.

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

7 Primera prueba: ejecución iterativa de las tres versiones

7.1 Despliegue de las aplicaciones

El despliegue de la prueba se realizará en dos PCs, separando el cliente del servidor y de la base de datos. Emplearemos PC1, PC1VM, PC2, PC2VM (PCs con sus respectivas máquinas virtuales):

- En el primero (PC1) ejecutaremos:
 - JMeter
 - Máquina virtual VMWare con: 1 CPU y 768 MB de RAM
 - Dentro de la máquina virtual del PC1 (PC1VM) tendremos asignada la dirección IP 10.X.Y.1 e iniciados tanto el GlassFish como el PostgreSQL

- En el segundo (PC2) ejecutaremos:
 - Máquina virtual VMWare con: 1 CPU y 768 MB de RAM
 - Dentro de la máquina virtual del PC2 (PC2VM) tendremos asignada la dirección IP 10.X.Y.2 e iniciado el GlassFish. **El PostgreSQL estará detenido.** Para detenerlo habrá que ejecutar en la máquina virtual el comando `sudo /etc/init.d/postgresql-8.4 stop`. También se puede usar el siguiente comando: `sudo service postgresql-8.4 stop`
- Prepararemos el script JMX para que acceda desde PC1 a PC2VM a través de su dirección IP 10.X.Y.2 (**es decir, accede a la máquina virtual del otro PC**)

En la siguiente figura se resume, de manera gráfica, la configuración del entorno de pruebas

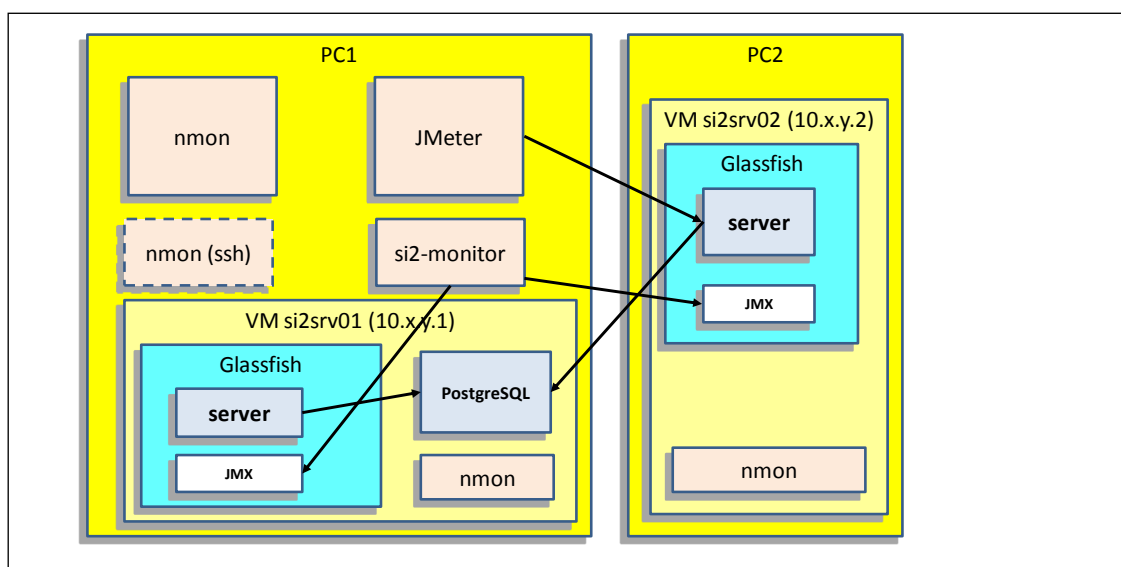


Figura 21: Diagrama de despliegue

Ejercicio 2: Preparar los PCs con el esquema descrito en la Figura 21. Para ello:

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

- El destino del despliegue de la parte cliente de P1-ws y de P1-ejb-cliente-remoto será PC2VM con IP 10.X.Y.2 (as.host.client)
- El destino del despliegue de la parte servidor de P1-ws y de P1-ejb-servidor-remoto será PC1VM con IP 10.X.Y.1 (as.host.server)
- La base de datos en todos ellos será la de PC1VM con IP 10.X.Y.1 (db.host)

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

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

7.2 Ejecución

Para comenzar la prueba se tomarán las siguientes precauciones:

- Crear un directorio P2 como raíz de todo el proyecto.
- Copiar dentro el fichero P2.jmx generado en el apartado anterior.
- Copiar dentro la carpeta datagen proporcionado en esta práctica. Esto es importante para que el script jmx encuentre los ficheros .csv de listas de tarjetas.
- Copiar dentro las carpetas P1-base / P1-ws / P1-ejb-servidor-remoto /P1-ejb-cliente-remoto
- Copiar dentro de **cada uno de los proyectos** anteriores la nueva carga de datos sql (insert.sql) para que todos los proyectos tengan los mismos datos en base de datos y para que concuerden con los valores en listado.csv.
- Limpiar y desplegar todos los proyectos desde este directorio. Se recomienda crear un Shell script que automatice el proceso.

```
#!/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
```

Nota: Fíjese que en la práctica P1-ws el fichero build.xml no dispone del target replegar. Deberá crear uno que llame a replegar-servicio y a replegar-cliente.

- Lanzar jmeter desde este directorio:
 - cd /ruta/a/P2
 - ~/apache-jmeter-4.0/bin/jmeter.sh &

- Abrir el archivo .jmx (*File*→*Open*)
- Pulsar en el menú *Run* → *Start*
- Si se van a repetir las pruebas, es recomendable limpiar los resultados previos (*Run* → *Clear*). Si se quiere seguir la evolución de la misma, hay que seleccionar (hacer click) sobre el “*Aggregated Report*” antes de empezar.

IMPORTANTE: La correcta repetición de esta prueba depende de varias precauciones que se deberán tomar:

- **La base de datos debe estar en el estado inicial, sin pagos preexistentes.** Para ello se deberá limpiar todo con *ant unsetup-db* o bien utilizar SQL para realizar el borrado de todos los pagos existentes.
- El PC debe estar con el menor número posible de aplicaciones abiertas. Cualquier proceso en ejecución que reste uso de CPU o de memoria puede influir en la prueba.
- Utilice el comando *top/nmon* (Linux) para controlar que la prueba se realiza correctamente:
 - En ningún caso el PC debe usar más RAM de la que dispone, y por tanto, estar paginando (empleando la zona de RAM de intercambio)
 - El acceso a disco y CPU debe estar a unos valores cercanos a cero antes de comenzar la prueba.

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

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

Una vez que los resultados han sido satisfactorios:

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

Incluir el directorio P2 en la entrega.

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

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

7.3 Depurando la prueba anterior

En caso de no obtener ningún pago, o de obtener errores en la columna, tenga en cuenta los siguientes consejos:

- Para comprobar el correcto funcionamiento de la simulación, realizar una ejecución del plan de pruebas, con una única repetición (variable “samples” a 1). Comprobar, mediante el *listener View Results Tree* que las peticiones se ejecutan correctamente, no se produce ningún tipo de error y los resultados que se obtienen son los adecuados. Una vez comprobado que todo el proceso funciona correctamente, reestablecer el valor de la variable “samples” a 1000 y desactivar dicho *listener* del plan de pruebas para que no aumente la carga de proceso de JMeter durante el resto de las pruebas.
- Asegúrese de que el directorio datagen/ y el resto del proyecto está estructurado tal y como se indicó anteriormente.
- El elemento escuchador (*Listener*) denominado “View Results Tree” permite obtener la salida HTML que ha generado la petición y la respuesta del servidor, lo que permite inspeccionar posibles errores.
- Habilite el modo debug (parámetro debug=true en el generador HTTP).
- Habilite el uso de statements no preparados. El código entregado imprime la consulta SQL ejecutada para cada caso en dicho modo.
- Acceda al fichero server.log del GlassFish, y analice las trazas impresas. Verifique manualmente las consultas SQL fallidas que ahí debieran estar registradas.
- Asegúrese que, tras arreglar los posibles problemas, vuelve a deshabilitar el modo debug, deshabilitar el escuchador “View results tree” y a habilitar el uso de *statements* preparados.

8 Monitorización

La monitorización es el proceso por el cual se recogen de manera dinámica estadísticas de funcionamiento de un equipo para evaluar su funcionamiento.

El proceso de monitorización debe siempre manejarse con cuidado. Como todo sistema de medida en tiempo real, es intrusivo en mayor o menor grado según las demandas de monitorización que se requieran. Una monitorización muy intensiva puede degradar significativamente el rendimiento de un servidor.

La monitorización se debe extender a todos los elementos en los que se despliega una aplicación. Eso incluye, por tanto, parámetros como son los siguientes:

- Hardware: Uso de CPU, memoria, ancho de banda utilizado por las comunicaciones, etc.
- Sistema operativo: Paginación, ocupación de los diversos sistemas de archivos, número de procesos o hilos disponibles por usuario, número máximo de archivos abiertos por usuario, etc.
- Máquina virtual Java: Uso del *heap*, funcionamiento del *garbage collector*, fragmentación de la memoria, generación de excepciones, etc.
- Servidor de aplicaciones: nivel de ocupación de las distintas colas, *pools* de proceso y conexiones descritos en el apartado anterior.
- Servidor de bases de datos: niveles de ocupación de procesos y conexiones, rendimiento de las consultas ejecutadas, bloqueos, etc.

Glassfish permite monitorizar un gran número de parámetros internos de funcionamiento del servidor. Su activación se realiza desde la consola de administración, tal como se muestra en la siguiente figura:

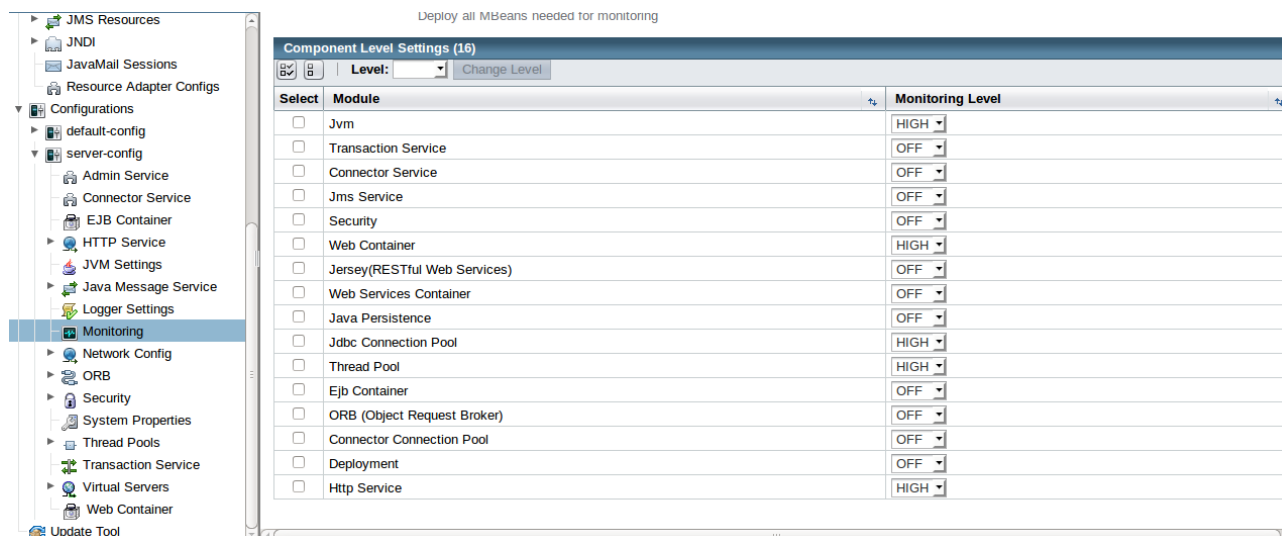


Figura 22: Parámetros de monitorización

8.1 Herramientas de monitorización

La monitorización de los parámetros citados requiere del uso de herramientas, cuyas principales funciones son:

- Visualización interactiva de los parámetros bajo monitorización.
- Registro periódico de sus valores.
- Cálculo de estadísticas.
- Generación de informes.

8.2 Monitorización del servidor

- En Linux existen diversas herramientas para realizar estas mediciones (top, sar...). En la distribución instalada en los laboratorios se puede utilizar el Monitor del Sistema, herramienta gráfica disponible desde la barra de acción Aplicaciones -> Herramientas del sistema -> Monitor del sistema.
- En las prácticas, para monitorizar el servidor en el que se ejecutan las aplicaciones utilizaremos la herramienta *nmon*. Se encuentra instalada en las máquinas virtuales que utilizaremos como entorno de producción.

Tras arrancar, *nmon* reconoce una serie de teclas, que activan o desactivan los distintos elementos de monitorización que proporciona. La primera pulsación de la letra activa la monitorización, mientras que, si se encuentra activada, una nueva pulsación la desactiva.

En esta práctica utilizaremos las siguientes monitorizaciones:

- c: activa / desactiva la monitorización del uso de la CPU del servidor.
- m: activa / desactiva la monitorización del uso la memoria de servidor.
- n: activa / desactiva la monitorización de las estadísticas de uso de la red.
- t: activa / desactiva la monitorización del uso del servidor por parte de los procesos que más recursos demandan ("*top*").

Para finalizar todas las monitorizaciones y volver a la *shell* del sistema, basta con pulsar la tecla q.

La siguiente figura muestra la pantalla de *nmon* con todas las monitorizaciones citadas activas.

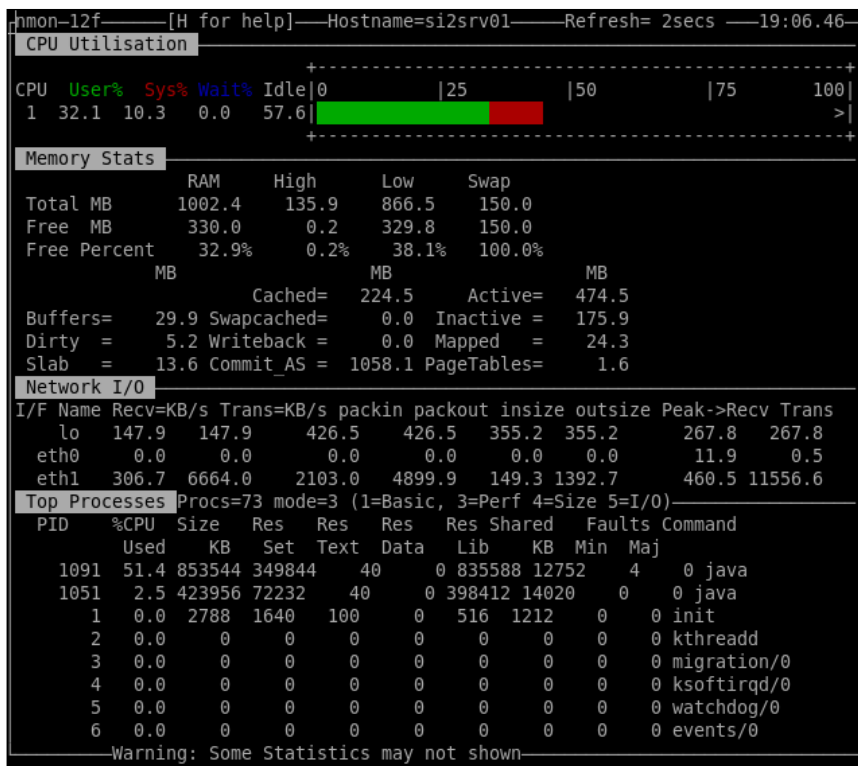


Figura 23: Monitorización mostrada por la herramienta *nmon*

8.3 Monitorización Java

Dentro del estándar de Java están definidas las *Java Management Extensions*, *JMX*, como arquitectura de monitorización y gestión. Los recursos gestionados de una máquina virtual se encapsulan mediante *Management Beans*, *MBeans*. Estos *beans* especiales permiten a las aplicaciones clientes de gestión el acceso local y remoto a los objetos y parámetros gestionados.

Aplicaciones clientes de gestión construidas sobre esta arquitectura son:

- Herramienta de administración de nivel de comandos *asadmin*. Permite consultar de manera interactiva los *MBeans* monitorizados.
- La consola de administración de Glassfish. Se puede acceder a los parámetros a monitorizar a través de las pantallas de gestión de *Stand Alone Instances* (para servidores aislados), o las de *Clusters*. También se puede acceder a partir de la configuración que hereda cada instancia. En nuestro caso accederemos así:

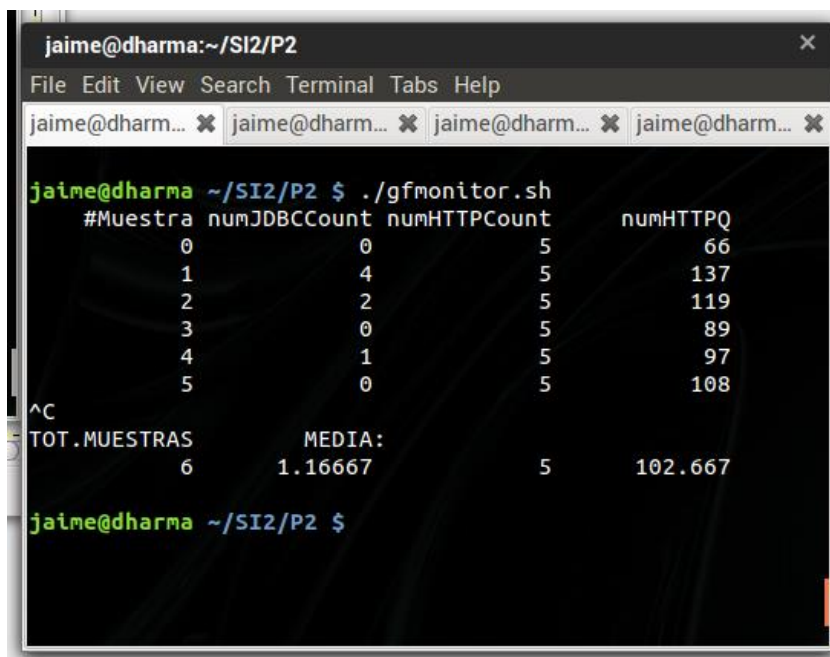
Configurations -> Server Config

- JConsole*. Es una aplicación cliente que se suministra con el *Java Development Kit*, *JDK*. El programa a arrancar es `$JAVA_HOME/bin/jconsole`.
- Dado que la arquitectura *JMX* presenta una interfaz pública de acceso a los *MBeans*, y que los fabricantes publican los que definen para gestionar sus aplicaciones, cualquier programador puede hacer su propia aplicación de monitorización a medida. En el caso de nuestras prácticas, se ha preparado una sencilla aplicación de monitorización en modo texto, que permite recoger los parámetros de interés para el problema a analizar. Para ejecutarla:

./si2-monitor.sh <host_glassfish>

El parámetro host_glassfish: indicará la dirección IP donde ejecuta el servidor de aplicaciones.

La siguiente imagen muestra la pantalla de monitorización.



```

jaime@dharma:~/SI2/P2
File Edit View Search Terminal Tabs Help
jaime@dharma... X jaime@dharma... X jaime@dharma... X jaime@dharma... X

jaime@dharma ~/SI2/P2 $ ./gfmonitor.sh
#Muestra numJDBCCount numHTTPCount numHTTPQ
0 0 5 66
1 4 5 137
2 2 5 119
3 0 5 89
4 1 5 97
5 0 5 108
^C
TOT.MUESTRAS MEDIA:
6 1.16667 5 102.667

jaime@dharma ~/SI2/P2 $

```

Figura 24: Monitorización mostrada por la aplicación si2-monitor

Tras su ejecución, la aplicación si2-monitor.sh toma muestras de los parámetros de interés cada segundo. Para finalizarla, basta interrumpir su ejecución pulsando Ctrl-C. En ese momento, calcula los valores medios de los parámetros monitorizados, y los visualiza en la pantalla.

Importante: El script si2-monitor.sh requiere que el mandato asadmin se encuentre en el PATH de ejecución. De no ser así, se puede incluir, por ejemplo para los PCs del laboratorio, con:

```
export PATH=/usr/local/glassfish-4.0/glassfish/bin:$PATH
```

Importante: El script si2-monitor.sh asume que el fichero 'passwordfile' se encuentra en el mismo directorio que el script. De no ser así (por ejemplo, si se ejecuta en la VM), editar el script y cambiar la línea:

```
GFPASSFILE=./passwordfile
```

Importante: Si se detectan problemas (lentitud, etc.) en la ejecución del script si2-monitor.sh en el host (PC), se puede copiar el script a la máquina virtual PC2VM mediante scp y ejecutarlo allí con localhost como argumento, o bien copiarlo a la máquina virtual PC1VM y ejecutarlo allí con argumento 10.X.Y.2 (la IP de PC2VM). Para copiarlo al directorio HOME de si2 se puede usar el mandato:

```
scp si2-monitor.sh si2@10.X.Y.Z:
```

La monitorización del servidor de aplicaciones en PC1VM se llevaría a cabo de forma similar.

8.4 Parámetros a monitorizar en la práctica

Los parámetros que se van a monitorizar en la práctica son los que presenta la aplicación anterior. A continuación se describen, identificándolos con el nombre que **si2-monitor.sh** les da en la cabecera de su tabla de resultados:

- **numHTTPCount.** Es el número de subprocesos o hilos (*processing threads*) que se encuentran activos en el momento de tomar la muestra. Su número máximo, por tanto, vendrá dado por el parámetro de configuración *Max Thread Pool Size*, descrito en 13.4.1.
- **numHTTPQ.** Es el número de elementos en cola de espera en el instante de tomarse la muestra. Su número máximo vendrá dado por el parámetro de configuración *Max Queue Size* descrito en 13.4.1.
- **numJDBCCount.** Son los valores de ocupación actual del **pool** de conexión a la base de datos Visa. Su valor máximo, por tanto, vendrá dado por el parámetro de configuración *Maximum Pool Size* de cada *pool*, tal como se explica en el apéndice 13.4.4.

8.5 Configurar el servidor de aplicaciones con parámetros correctos de rendimiento y monitorización

Comenzamos con la activación de los parámetros del servidor de aplicaciones que permitan simular, en todo lo posible, un entorno de producción.

Esta activación se realizará únicamente sobre el servidor Glassfish instalado en PC2VM.

- Utilizar la máquina virtual java en modo servidor. Si estuviera en modo cliente realizar los siguientes cambios.
 - Sobre la consola de administración:
Configurations -> server-config -> JVM Settings -> Pestaña JVM Options
 - Eliminar la opción *-client*
 - Agregar la opción *-server*
- Configurar los valores de memoria mínimo y máximo asignados a la máquina virtual java.
 - Sobre la consola de administración:
Configurations -> server-config -> JVM Settings -> Pestaña JVM Options
 - Comprobar el valor máximo de memoria para el Heap de java (opción *-Xmx*). Debe ser 512m (512 MB).
 - Añadir la opción que establece el Heap mínimo: *-Xms512m*
- Eliminar el despliegue automático y recarga automática de aplicaciones.
 - Sobre la consola de administración:
Domain -> Pestaña Applications Configuration
 - Eliminar el *checkbox* Reload
 - Eliminar el *checkbox* Autodeploy
- Activar la precompilación de JSPs.
 - En la misma pantalla anterior, marcar el *checkbox* Precompile
- Activar un nivel de monitorización adecuado:
 - Sobre la consola de administración:
Configurations -> server-config -> Monitoring
 - Activar el modo de monitorización "HIGH" para los servicios Web Container, Thread Pool, JVM, *JDBC Connection Pool*, HTTP Service. Poner el resto de la monitorización en "OFF".

Ejercicio 4: Adaptar la configuración del servidor de aplicaciones a los valores indicados. Guardar, como referencia, la configuración resultante, contenida en el archivo de configuración localizado en la máquina virtual en `$J2EE_HOME/domains/domain1/config/domain.xml`¹. Para obtener la versión correcta de este archivo es necesario detener el servidor de aplicaciones. **Incluir este fichero en el entregable de la práctica. Se puede copiar al PC del laboratorio con scp.**

Revisar el script `si2-monitor.sh` e indicar los mandatos `asadmin` que debemos ejecutar en el Host PC1 para averiguar los valores siguientes, mencionados en el Apéndice 1, del servidor P1VM1:

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

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

8.6 Registrar los parámetros de configuración del servidor

Se deben registrar en la hoja de cálculo de resultados los parámetros de configuración del servidor de cara al rendimiento que se presentan en la siguiente tabla:

Parámetro	Posición en consola administración
Valores máximo y mínimo del heap de memoria que utiliza la máquina virtual Java	Configurations -> server-config -> JVM Settings -> Pestaña JVM Options -> Parámetros -Xmx y -Xms
Máximo número de conexiones a procesar simultáneamente por el servidor web	Configurations -> server-config -> Thread Pools-> http-thread-pool -> Parámetro Max Thread Pool Size
Tamaño máximo de la cola de conexiones pendientes de servicio	Configurations -> server-config -> Thread Pools-> http-thread-pool -> Parámetro Max Queue Size
Máximo número de sesiones en el contenedor Web. (Valor por defecto -1, ilimitadas)	Configurations -> server-config -> Web Container -> Pestaña Manager Properties -> Parámetro Max Sessions
Máximo número de conexiones en <i>pools</i> JDBC	Resources -> JDBC -> Connection Pools Visa Pool -> Parámetro Maximum Pool Size

Ejercicio 5: Registrar en la hoja de cálculo de resultados los valores de configuración que tienen estos parámetros.

¹ ¡Ojo! Este archivo contiene información específica de cada ordenador, como es, por ejemplo, el nombre del servidor y las aplicaciones instaladas en él, por lo cual no se puede copiar de una máquina a otra.

9 Segunda prueba: ejecución iterativa de P1-base con monitorización

Para realizar la prueba de rendimiento tomaremos como aplicación bajo estudio únicamente la versión P1-base de la práctica, es decir, con acceso a la base de datos a través de DataSource y sin hacer uso de los Web Services ni EJBs. Por tanto, en este punto, repliegue las otras versiones de la práctica (P1-ws, P1-ejb-servidor-remoto y P1-ejb-cliente-remoto), dejando únicamente desplegada P1-base.

9.1 Reiniciar el servidor de aplicaciones

Puesto que ya hemos modificado la configuración del dominio para la prueba, realizaremos las siguientes acciones previas:

- Detener completamente el servidor de aplicaciones de **si2srv01** del modo habitual (asadmin stop-domain)
- Rearrancar el servidor de aplicaciones en **si2srv02** del modo habitual (asadmin stop-domain y asadmin start-domain)

Además de servir para activar los cambios de configuración realizados, con ello también se conseguirá trabajar con un entorno lo más estable posible. **No arrancar la aplicación de administración (no conectar al puerto 4848), ya que se ejecuta sobre el mismo servidor de aplicaciones y compite por los recursos del sistema.**

9.2 Iniciar herramientas de monitorización del servidor

Como ordenador de monitorización se utilizará PC1, que es **otro ordenador del laboratorio distinto al que ejecuta la máquina virtual si2srv02**. En él se arrancará:

- La herramienta *nmon* para monitorización del propio ordenador cliente.
- Un terminal remoto al servidor si2srv02 mediante *ssh*, en el que se arrancará también la herramienta *nmon* para monitorizar el servidor.
- El programa de monitorización del servidor de aplicaciones Glassfish, *si2-monitor.sh* <dirección_ip_si2srv02>

9.3 Cambios en el JMX

Puesto que la nueva prueba va a ejecutarse únicamente sobre la versión *P1-base* de la práctica, deshabilitaremos el resto de los tests del plan de pruebas (P1-ejb, P1-ws)

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

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

10 Saturación de un servidor: la curva de productividad

En esta práctica se a realizar un análisis del rendimiento de la aplicación del sistema de pagos realizado en las prácticas anteriores. Para ello se va a emplear como metodología básica el análisis de la curva de productividad (*throughput*) versus el número de usuarios concurrentes.

Según se ha visto en las clases de teoría,

- Productividad (*throughput*) es el número de respuestas a eventos que se realizan por unidad de tiempo en un intervalo de observación. En los modelos teóricos se identifica con el parámetro λ .
- Capacidad de un sistema es la medida de la cantidad máxima de trabajo que el sistema o un componente del mismo es capaz de realizar. Equivale, por tanto, a la productividad máxima que el sistema podría, en teoría, soportar. Se identifica con el parámetro μ .
- Latencia de un sistema es el intervalo de tiempo en el cual se produce la respuesta a un evento. Se identifica con el parámetro W estudiado en la teoría de la asignatura.

La curva de productividad de un sistema es uno de los principales mecanismos para analizar su comportamiento de cara al rendimiento. Esta curva tiene una forma como la que se muestra en la siguiente figura:

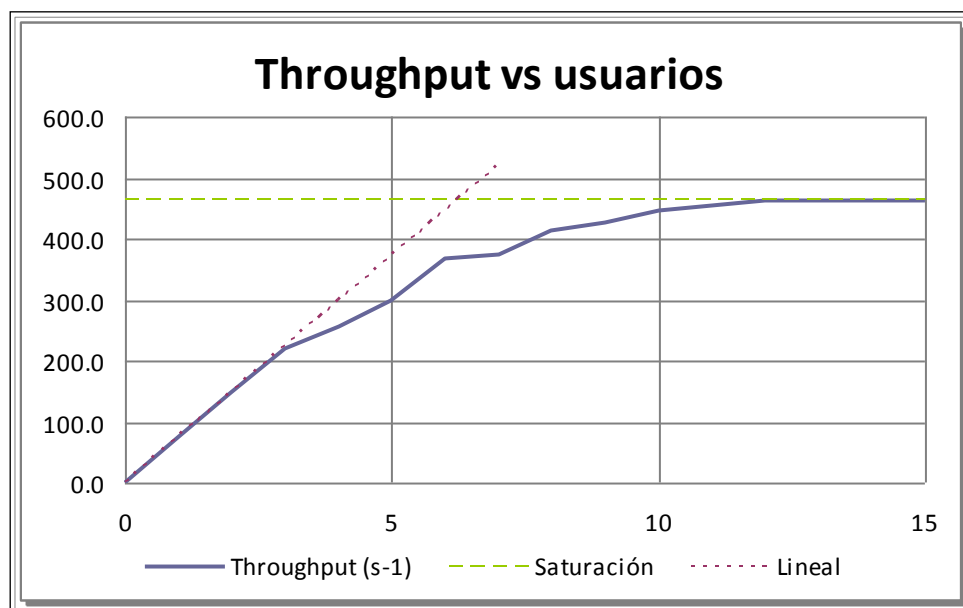


Figura 25: Curva de productividad

En la curva de distinguen tres zonas:

- Zona lineal: Para bajo número de usuarios, el *throughput* que procesa el sistema crece linealmente con el número de usuarios conectados. En esta situación, el servidor funciona correctamente y no hay saturaciones en ninguno de los puntos de la cadena de procesamiento. La latencia del sistema aumenta de modo moderado, permaneciendo prácticamente constante.
- Zona de transición: Al aumentar el número de usuarios, comienza a haber esperas en alguno de los servicios por los que tienen que pasar las peticiones, bien sean *pools* de procesamiento, de conexiones o cualquier otro de los recursos compartidos que existen en el sistema. Las peticiones se encolan, y, por tanto, el rendimiento baja. El *throughput* sigue aumentando, pero pierde el ritmo de crecimiento lineal. La latencia del sistema aumenta.

- Zona de saturación: En alguno de los puntos de la cadena de procesamiento se ha alcanzado o incluso excedido la capacidad máxima de proceso. El sistema no es capaz de procesar más peticiones, y la curva de *throughput* permanece constante aunque aumenten los usuarios conectados. La latencia del sistema continúa aumentando a mayor ritmo que en la zona anterior.

El punto de corte de las rectas que definen la tendencia de las zonas lineal y de saturación se denomina **punto de saturación**. Es el punto que no conviene que se rebase durante la operativa normal del sistema.

De cara a obtener un rendimiento adecuado, un sistema se debe diseñar para que trabaje habitualmente en la zona lineal, previendo que en picos de carga pueda entrar en la zona de transición, pero nunca entrar en zona de saturación.

Para que una prueba de rendimiento sea real y se puedan obtener conclusiones satisfactorias de ella, se deben tener en cuenta múltiples factores. En el apéndice 2 se realizan consideraciones a tener en cuenta sobre estas pruebas.

11 Tercera prueba: determinación de la curva de productividad de P1-base

11.1 Preparar el cliente para la prueba de rendimiento

Para realizar la prueba de rendimiento se utilizará la herramienta JMeter con el nuevo script de pruebas (P2-curvaProductividad.jmx) que se proporciona a los alumnos para simplificar la ejecución de la práctica.

Las funciones que realiza son las siguientes:

- Generación de datos de una transacción de pago, y llamada al servlet ComienzaPago.
- Generación de los datos de la tarjeta sobre la que se realiza el pago, y llamada al servlet ProcesaPago.
- Este script simula una situación más realista que la anterior al introducir un elemento, **think time**, que no es otra cosa que el tiempo que un usuario normal pasaría “pensando”. Este elemento introduce pausas que permiten al sistema soportar un mayor número de usuarios.

La siguiente figura muestra una captura de JMeter con el script proporcionado para la prueba de rendimiento.

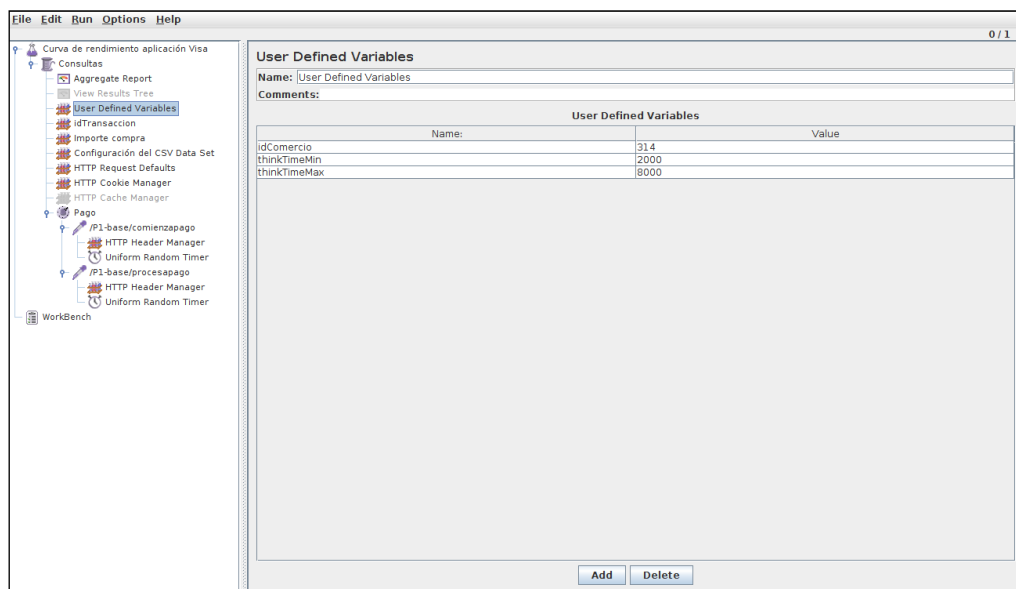


Figura 26: JMeter con el script P2-curvaProductividad

La configuración del script de pruebas permite modificar:

- El número de usuarios concurrentes a simular, a través del parámetro *Number of Threads (users)* del *Thread Group* "Consultas", elemento raíz del script de prueba.
- El número de iteraciones del script que realizará cada usuario, a través del parámetro *Loop Count*, en el mismo elemento.
- El destino del servidor donde se dirigen las peticiones, a través del objeto *HTTP Request Defaults*, como se ha realizado en prácticas anteriores.
- El identificador de comercio del que se realiza la transacción. Variando su valor se pueden realizar, si se desea, múltiples ejecuciones del script sin necesitar borrar la tabla de pagos en la base de datos. Su valor se puede establecer en el objeto *User Defined Variables*, tal y como se presenta en la figura anterior.
- El *think time* del usuario entre peticiones. Se configura como una variable aleatoria con distribución uniforme entre unos valores máximo y mínimo que se pueden establecer en el objeto *User Defined Variables*, tal y como se presenta en la figura anterior. Los valores de dichos parámetros están expresados en milisegundos. Para que la práctica se desarrolle en un periodo de tiempo no excesivamente largo, el rango de variación del *think time* se establecerá entre 1 y 2 s.
- El valor inicial a asignar del identificador de transacción para la compra en Visa, en el objeto contador *idTransaccion*.

Los datos para realizar el pago se cargan desde el archivo `datagen/listado.csv`, que se tomará de las prácticas anteriores.

Ejercicio 7: Preparar el script de JMeter para su ejecución en el entorno de pruebas. Cambiar la dirección destino del servidor para que acceda al host en el que se encuentra el servidor de aplicaciones. Crear también el directorio `datagen` en el mismo directorio donde se encuentre el script, y copiar en él el archivo `listado.csv`, ya que de dicho archivo, al igual que en las prácticas anteriores, se obtienen los datos necesarios para simular el pago.

A continuación, realizar una ejecución del plan de pruebas, con un único usuario, una única ejecución, y un

think time bajo (entre 1 y 2 segundos) para verificar que el sistema funciona correctamente. Comprobar, mediante el *listener View Results Tree* que las peticiones se ejecutan correctamente, no se produce ningún tipo de error y los resultados que se obtienen son los adecuados.

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

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

11.2 Obtención de la curva de productividad

Muchas herramientas permiten obtener la curva de productividad de una manera automática. En el caso de JMeter, esta curva se puede obtener también mediante el objeto *listener* denominado *graph results*, ejecutando una prueba con un número variable de usuarios que se van introduciendo en el sistema con una periodicidad establecida a través del parámetro *Ramp-Up Period*, del elemento *Thread Group* del script de JMeter.

Dado que los resultados de esa prueba son difíciles de capturar correctamente e interpretar, en esta práctica vamos a obtener la curva paso a paso mediante diversas ejecuciones del *script* de prueba, recogiendo los resultados en una hoja de cálculo y presentando la curva de rendimiento mediante un gráfico obtenido en la misma.

En cada una de las ejecuciones que vamos a realizar vamos a simular el funcionamiento del sistema en un régimen permanente que incluye un determinado número **C** de clientes conectados. Estos clientes estarán lanzando peticiones de manera continua. De esta manera, los valores promedio que obtengamos a través del *listener Aggregate Report* representarán el punto de la curva de rendimiento cuya abscisa (número de usuarios) sea igual a **C**.

Esta medición se repetirá para diversos valores de **C**, hasta que obtengamos los suficientes puntos para representar correctamente la curva de rendimiento. Este cambio, como ya se ha explicado, se realiza mediante el parámetro *Number of Threads (users)*. **Se comenzará con 1 en la primera ejecución, y se irá aumentado de 25 en 25 hasta detectar que se ha llegado a la zona de saturación.**

Cada uno de los pasos se realizará con unas 10 repeticiones del *script* por cliente. Dado que lo que nos interesa es el análisis del régimen permanente del sistema en estas condiciones, la rampa de entrada de usuarios se establecerá para un periodo corto de tiempo, fijado en 10 s.

Nota: El valor 25 es orientativo y dependerá del rendimiento de los ordenadores empleados en el laboratorio. Si se observa que al aumentar el número de hilos (usuarios) de 25 en 25 no llega a saturarse el servidor, se deberá utilizar un valor mayor.

El uso concurrente de la red de comunicaciones de los laboratorios también puede influir en la prueba negativamente. Por ello, se recomienda realizarla en momentos en los que no haya un número elevado de alumnos compitiendo por el uso de la red (por ejemplo, otros grupos realizando la misma prueba).

11.3 Ejecución de la prueba

Una vez realizadas todas las acciones previas, se está en disposición de realizar la prueba.

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

- Previamente a la ejecución de la prueba se lanzará una ejecución del *script* de pruebas (unas 10 ejecuciones de un único usuario) de la que no se tomarán resultados, para iniciar el sistema y preparar medidas consistentes a lo largo de todo al proceso.

Borrar los resultados de la ejecución anterior. En la barra de acción de JMeter, seleccionar Run -> Clear All.

- Borrar los datos de pagos en la base de datos VISA.
- Seleccionar el número de usuarios para la prueba en JMeter (parámetro **C** de la prueba)
- Conmutar en JMeter a la pantalla de presentación de resultados, *Aggregate Report*.
- Ejecutar la prueba. En la barra de acción de JMeter, seleccionar Run -> Start.
- Ejecutar el programa de monitorización si2-monitor.sh.
 - Arrancarlo cuando haya pasado el tiempo definido como rampa de subida de usuarios en JMeter.
 - Detenerlo cuando esté a punto de terminar la ejecución de la prueba.
 - Registrar los resultados que proporciona la monitorización en la hoja de cálculo.
- Durante el periodo de monitorización anterior, vigilar que los recursos del servidor si2srv02 y del ordenador que se emplea para realizar la prueba no se saturen, mediante inspección del programa de monitorización *nmon* que se ejecuta en ambas máquinas.
- Finalizada la prueba, salvar el resultado de la ejecución del *Aggregate Report* en un archivo, y registrar en la hoja de cálculo de resultados los valores *Average*, *90% line* y *Throughput* para las siguientes peticiones:
 - ProcesaPago.
 - Total.

Una vez realizadas las iteraciones necesarias para alcanzar la saturación, representar la curva de *Throughput* versus usuarios.

Incluir el fichero P2-curvaProductividad.jmx en la entrega.

¿Cómo debemos invocar a *nmon* para recolectar muestras cada 5 segundos durante 10 minutos que incluyan los 'top processes' en el fichero log-file.nmon?

Ejecutar el mandato anterior en PC1VM1 durante una de las pruebas de P1-ws y usar *nmon Analyser* (https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Power+Systems/page/nmon_a_nalyser) para generar un fichero Excel con los resultados. Revisadlo y comentad algún aspecto relevante.

11.4 Interpretación de los resultados

Tras la realización de la curva de rendimiento es el momento de analizar los resultados obtenidos y obtener conclusiones.

Ejercicio 9: Responda a las siguientes cuestiones:

- A partir de la curva obtenida, determinar para cuántos usuarios conectados se produce el punto de

saturación, cuál es el máximo *throughput* que se alcanza en el mismo, y el *throughput* máximo que se obtiene en zona de saturación.

- Analizando los valores de monitorización que se han ido obteniendo durante la elaboración de la curva, sugerir el parámetro del servidor de aplicaciones que se cambiaría para obtener el punto de saturación en un número mayor de usuarios.
- Realizar el ajuste correspondiente en el servidor de aplicaciones, reiniciarlo y tomar una nueva muestra cercana al punto de saturación. ¿Ha mejorado el rendimiento del sistema? Documente en la memoria de prácticas el cambio realizado y la mejora obtenida.

12 Entrega

La fecha de entrega de esta práctica será la **semana del 2 al 6 de abril de 2018**. En todos los casos, la práctica se entregará antes del comienzo de la clase.

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

Nomenclatura del fichero a entregar: SI2P2_<grupo>_<pareja>.zip (ejemplo: SI2P2_2311_1.zip)

Contenido del fichero:

- El directorio P2, con el código resultante de todas las modificaciones sugeridas.
- Archivo JMX con el guion de pruebas creado por el alumno en la primera parte (P2.jmx)
- Archivo JMX con el guion de pruebas de productividad (P2-curvaProductividad.jmx)..
- Archivo SI2-P2-curvaProductividad.ods con los datos recogidos y la curva de productividad.
- Archivo domain.xml (del dominio domain1) tras realizar las modificaciones sugeridas en la segunda parte de la práctica.
- Memoria de la ejecución de la práctica, en la que se comparen los valores de latencia (al percentil 90%) y throughput para las tres modalidades descritas en la primera parte, así como la respuesta a todas las cuestiones planteadas y los resultados experimentales de la segunda y tercera parte de la práctica.

13 Bibliografía recomendada

- *Java EE 7 Tutorial*:
<http://download.oracle.com/javase/7/tutorial/doc/>
- Apache JMeter:
<http://jmeter.apache.org/>
- Oracle, *GlassFish Enterprise Server v3.1 Performance Tuning Guide*,
http://docs.oracle.com/cd/E18930_01/pdf/821-2431.pdf
- Oracle, *GlassFish Enterprise Server v3 Administration Guide*,
<http://docs.oracle.com/cd/E19226-01/820-7692/820-7692.pdf>
- Oracle, *Java 6 Virtual Machine Garbage Collection Tuning*,
<http://www.oracle.com/technetwork/java/javase/gc-tuning-6-140523.html>

Apéndice 1: Aspectos generales del rendimiento en aplicaciones J2EE

Java es un lenguaje de gran potencia y que hace muchos aspectos de la implantación transparentes al programador. Esto puede llegar a la generación de código poco optimizado si no se toman las debidas precauciones. Algunas de las mejores prácticas de programación en Java orientadas al rendimiento se presentan a continuación.

13.1 Aplicación

- Evitar el uso de registros de salida desde el programa, en especial en elementos de la arquitectura J2EE. No emplear llamadas `System.out.println()`.
- Evitar en todo lo posible los procesos de serialización / deserialización de objetos.
- Declarar las constantes como `static final`, ya que permite al compilador mayores optimizaciones.
- Declarar como `final` las variables que no se modifiquen tras la asignación de su valor inicial. Declarar de esta manera también los argumentos que no sean modificados en el interior del método.
- Evitar concatenaciones de cadenas de caracteres mediante operador '+'. Utilizar, en su lugar, el método `StringBuilder.append()`. La clase `String` produce objetos fijos, cuya concatenación requiere creación de objetos nuevos y destrucción de los antiguos.
- Asignar `null` a las variables cuando ya no se utilicen. Mejora el rendimiento del *garbage collector*.
- Evitar los finalizadores de las clases. Su ejecución desde el *garbage collector* puede producir efectos impredecibles.
- Emplear sincronización de código sólo cuando sea estrictamente necesario.
- Minimizar el uso de la sesión en *servlets* y JSPs. Procurar mantener el tamaño de la sesión por debajo de los 7 KB.
- Evitar siempre que sea posible la persistencia de la sesión entre nodos de un mismo servidor de aplicaciones. Es preferible almacenar valores necesarios para toda una sesión en base de datos (ejemplo: un "carrito de la compra" en una tienda web).
- Cuidar el diseño de las consultas a bases de datos, y optimizarlas de manera que no consuman excesivos recursos.

13.2 Java Virtual Machine (JVM)

- Iniciar la JVM optimizada para servidor (opción `-server`).
- Optimizar el tamaño del *heap* de Java. Ajustar la memoria máxima utilizable por la JVM a un valor que no exceda la memoria prevista. Un exceso de memoria hace que el proceso del *garbage collector* sea más lento de lo necesario.
- Arrancar la JVM con un valor de *heap* mínimo y máximo iguales. Hace que el sistema se inicie con la memoria correcta y evita expansiones dinámicas de la misma.

13.3 Configuración del hardware y sistema operativo

- Utilizar sistemas multiprocesador siempre que sea posible.

- Disponer de una capacidad de CPU y memoria acorde a todos los procesos que se ejecuten en el ordenador, y evitar trabajar en situaciones de alta ocupación en estos parámetros. Los valores máximos permisibles dependen del tipo de servidor y del sistema operativo empleado, pero pueden oscilar entre el 70 y el 80% en la mayoría de los casos.
- Ajustar la memoria asignada al servidor de aplicaciones a los valores recomendados por los fabricantes.
- Evitar la paginación del sistema.
- Colocar los directorios de log, paginación y documentos en distintos sistemas de archivos.
- Determinar el ancho de banda de comunicaciones adecuado al número de usuarios concurrentes previsto y tamaños medios de las respuestas a sus peticiones.
- Comprobar las recomendaciones de configuración del fabricante del servidor de aplicaciones para optimizar el rendimiento de la plataforma hardware / sistema operativo elegida.

13.4 Servidor de aplicaciones

13.4.1 Servicio HTTP

Es el proceso que se encarga de la ejecución de peticiones de los usuarios a través del protocolo HTTP.

Consta de dos elementos:

- Los subprocesos de recepción de llamadas, *HTTP listeners*. Esperan la recepción de peticiones en el puerto correspondiente. Una vez aceptadas las peticiones, las pasan a la *Connection Pool*, donde esperan a ser atendidas por el servidor.
- El servidor HTTP, propiamente dicho. Extrae las peticiones de la *Connection Pool* y realiza el proceso solicitado sobre los hilos de proceso (*Processing Threads*).

Principales parámetros de configuración

Sus parámetros de configuración se acceden en la consola de administración a través de diferentes paneles de configuración. Los principales parámetros de cara al rendimiento son los siguientes:

- **Max Thread Pool Size:** Es el máximo número de subprocesos o hilos (*processing threads*) que se pueden activar para procesar peticiones. Representa, por tanto, el máximo número de peticiones que el servidor puede atender simultáneamente. Su configuración puede realizarse a través del panel *Configurations* → *server-config* → *Thread Pools* → *http-thread-pool*.
- **Request time-out:** Es el máximo tiempo que un subproceso espera por una respuesta del servidor. Su configuración puede realizarse a través del panel *Configurations* → *server-config* → *Network Config* → *Network Listeners* → *http-listener-1* (pestaña HTTP).
- **Max Queue Size:** Tamaño de la cola del *Connection Pool*. Su configuración puede realizarse a través del panel *Configurations* → *server-config* → *Thread Pools* → *http-thread-pool*.
- **Max Connections Count:** Indica el tamaño máximo de la cola de peticiones en el socket de entrada. Su configuración puede realizarse a través del panel *Configurations* → *server-config* → *Network Config* → *Transports* → *tcp*.
- **HTTP Listener Acceptor Threads:** Número de subprocesos que ejecuta el Listener HTTP. Su configuración puede realizarse a través del panel *Configurations* → *server-config* → *Network Config* → *Transports* → *tcp*.

Para más información puede consultarse: <https://glassfish.java.net/docs/4.0/upgrade-guide.pdf>.

13.4.2 Web Container

Es el contenedor de los objetos necesarios para la ejecución de servlets y JSPs. Los parámetros de configuración se acceden con Configurations -> server-config -> Web Container.

Principales parámetros de configuración

Sus parámetros de configuración principales de cara al rendimiento son los siguientes:

- **Session Timeout:** Tiempo máximo de vida de una sesión inactiva. Transcurrido este tiempo, se elimina. Su valor por defecto es de 30 min. Se encuentra en la pestaña Session Properties.
- **Max Sessions:** Es el máximo número de sesiones que se permite. Se accede en la pestaña Manager Properties.

13.4.3 EJB Container

Es el contenedor de los Enterprise Java Beans. Los parámetros de configuración se encuentra en Configuration -> server-config -> EJB Container.

Principales parámetros de configuración

Contiene parámetros de configuración independientes para los *EJB Session* y para los *Message Driven*. Los primeros se encuentran en la pestaña de configuración *EJB Settings*, mientras que los segundos están en la denominada *MDB Settings*. Los principales para los primeros son:

- **Maximum Pool Size:** Número máximo de EJB Session que se pueden mantener activos simultáneamente.
- **Pool Idle Timeout:** Tiempo máximo de espera inactivo para eliminar un EJB del pool, si se está por encima del mínimo establecido.

13.4.4 Recursos JDBC

Los recursos JDBC son los que centralizan el acceso de la aplicación a las bases de datos, tal y como se ha presentado en anteriores prácticas. Las conexiones disponibles en un *Connection Pool* son recursos compartidos para todos los *Data Sources* que la utilicen. Por tanto, su dimensionamiento adecuado es imprescindible para que el rendimiento de una aplicación sea el correcto.

Es importante destacar también que estos parámetros sólo regulan el comportamiento de la conexión con el gestor de base de datos, y no el propio gestor. Según el gestor que se utilice será necesario realizar una configuración adecuada para optimizar su rendimiento. Esta configuración debe ser compatible con la definida en el recurso JDBC que se utiliza para acceder a él. Por ejemplo, de nada sirve aumentar el número máximo de conexiones de un *Connection Pool* por encima del límite máximo de conexiones por usuario definido en el gestor.

Principales parámetros de configuración

La configuración de los recursos JDBC no se realiza en el submenú *Configurations* de la consola de administración, sino que hay que seleccionar, en el marco de la izquierda de la misma y en secuencia, las opciones Resources -> JDBC. Desde ese punto se accede tanto a la configuración de los *Data Sources* como de los *Connection Pools*, si bien esta última es la que tiene importancia de cara al rendimiento.

Una vez en esa opción, seleccionar el *Connection Pool* que utiliza nuestra aplicación.

- **Resource Type:** Indica el tipo de recurso que se define. Los tipos de recursos determinan la funcionalidad de que dispondrá el recurso elegido. Por ejemplo, es necesario seleccionar el tipo `javax.sql.XDataSource` si hay que utilizar transacciones que realicen two phase commit.
- **Initial and Minimum Pool Size:** Número de conexiones mínimas que se mantienen abiertas al gestor de bases de datos.
- **Maximum Pool Size:** Número máximo de conexiones que se pueden mantener activos simultáneamente.
- **Pool Idle Timeout:** Tiempo máximo de espera inactivo para eliminar una conexión, si se está por encima del mínimo establecido.
- **Transaction Isolation:** Permite especificar el grado de aislamiento que tienen las transacciones que se ejecuten sobre esta conexión. El valor por defecto depende del driver de base que se esté utilizando. Habitualmente suele ser `read committed`.

14 Apéndice 2: Obtención de la curva de rendimiento en entornos de pruebas

Para tener la seguridad de que el rendimiento en producción de un sistema va a ser el requerido antes de su puesta en funcionamiento es necesario realizar pruebas. Normalmente, estas pruebas no se pueden hacer sobre el sistema real ni en condiciones reales. Por este motivo, las pruebas se van a realizar de manera simulada. Para la realización de la simulación hay que tener en cuenta una serie de aspectos que son críticos para la obtención de resultados que reflejen realmente el comportamiento del sistema en producción:

- Escalado correcto de los sistemas de producción en el entorno de pruebas. Normalmente, en el entorno de pruebas no se dispondrá de la misma potencia de sistemas que en el entorno de producción. Si esto es así, es necesario poder tener la seguridad de que se conoce el factor de escala que es necesario aplicar para extrapolar los resultados obtenidos al entorno de producción.

Esto no siempre es fácil de conseguir, aunque en algunos casos la arquitectura de los servidores de aplicaciones puede ayudar a conseguirlo. En sistemas con escalado horizontal, en los que la aplicación se desplegará sobre un *cluster* de servidores, las pruebas se pueden realizar sobre un número reducido de nodos, para estimar una capacidad por nodo. El factor multiplicativo para el sistema total será aproximadamente igual al factor por el que se multipliquen los nodos (normalmente algo inferior por no ser la escalabilidad completamente lineal con pendiente unidad).

- Manejo en el entorno de pruebas de un conjunto de datos representativo. Sobre todo en el contenido de las bases de datos, el contenido de las tablas debe ser igual o lo más parecido posible al de producción. Esto incluye no sólo el volumen de datos, sino también su distribución y nivel de aleatoriedad. De nada sirve incluir una tabla de productos con un millón de registros, en igual número al de producción, si todos los productos que se han incluido son el mismo repetido, ya que la ejecución de las consultas variará notablemente en ambos casos.
- Simulación correcta del comportamiento de los usuarios. Las pruebas se realizarán con usuarios simulados, no con usuarios reales realizando trabajo real. Se realizarán *scripts* de simulación de usuarios, que se reproducirán con una herramienta de pruebas, como, por ejemplo, JMeter, aplicación empleada en nuestras prácticas. Si estos *scripts* no reflejan el trabajo real de los usuarios, se estará comprobando el comportamiento del sistema en unas condiciones distintas a las que se va a encontrar en el entorno de producción. Algunas consideraciones a tener en cuenta para lograr unas pruebas reales son:
 - Realizar los *scripts* de pruebas mediante captura de navegaciones reales de usuarios de producción sobre el entorno de pruebas. JMeter, por ejemplo, proporciona el *HTTP Proxy Server* que, interpuesto entre el navegador del cliente y el servidor de aplicaciones, es capaz de registrar las peticiones realizadas por el usuario y las respuestas entregadas por el servidor. A partir de estas navegaciones, se pueden construir *scripts* de prueba más reales, haciendo los cambios necesarios para sustituir los parámetros empleados en las peticiones por otros de tipo aleatorio que reflejen el acceso por múltiples usuarios de manera concurrente.
 - Asignar unos valores adecuados a los tiempos de espera entre peticiones por parte de cada usuario. El tiempo que el usuario emplea en analizar los resultados recibidos del servidor (conocido en las pruebas de rendimiento como *think time*) es vital para determinar los tiempos que el usuario permanecerá inactivo, y calcular correctamente el número de usuarios simultáneos que se pueden mantener con un determinado nivel de concurrencia.