Sistemas Informáticos II
Práctica 2

Prácticas 2020/2021

Objetivos P2

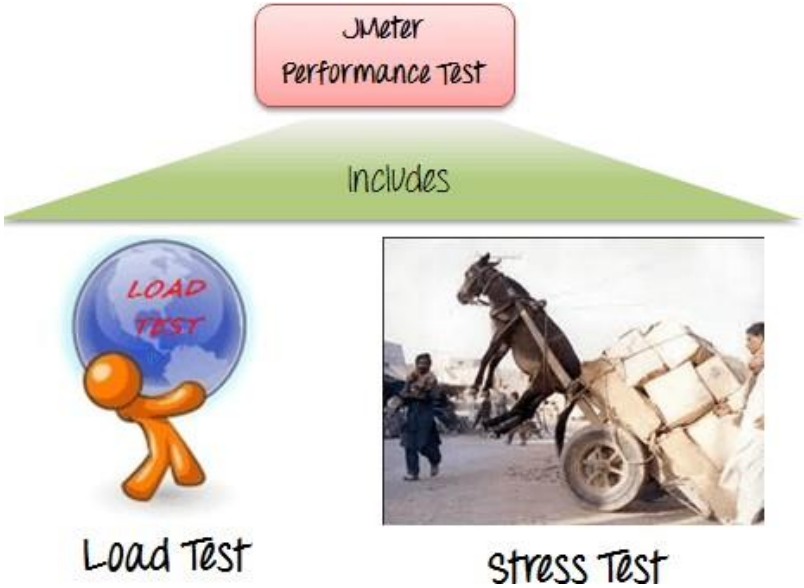
- Objetivos
  - Análisis de rendimiento de las aplicaciones realizadas siguiendo la arquitectura Java EE
  - Monitorización del servidor de aplicaciones
  - Análisis de la cadena de procesamiento dentro del servidor de aplicaciones.
  - Elementos que la componen.
  - Puntos de encolamiento.
  - Configuración del servidor de aplicaciones de cara al rendimiento
  - Mecanismos de evaluación del rendimiento, monitorización y realización de pruebas unitarias.
  - Pruebas de stress de la aplicación, mediante el empleo de JMeter
- Material entregado
  - **P2-alumnos.tgz** : Incluye nueva versión la base de datos y fichero .csv de pruebas
  - **SI2-P2-curvaProductividad.ods** (hoja de cálculo)
  - **SI2-P2-curvaProductividad.jmx** (fichero Jmeter)

Pruebas a realizar

- Primera parte
  - Creación de un plan de pruebas con Jmeter
  - Pruebas **unitarias** (un solo usuario) con el fin de identificar la mejor solución en cuanto a rendimiento de las tres desarrolladas en la P1 (P1-base, P1-ws-cliente, P1-ejb)
- Segunda parte:
  - Pruebas de stress sobre una de las soluciones (P1-base)
  - Múltiples usuarios
  - Búsqueda del punto de saturación del sistema
  - Mejora del rendimiento

Pruebas de rendimiento

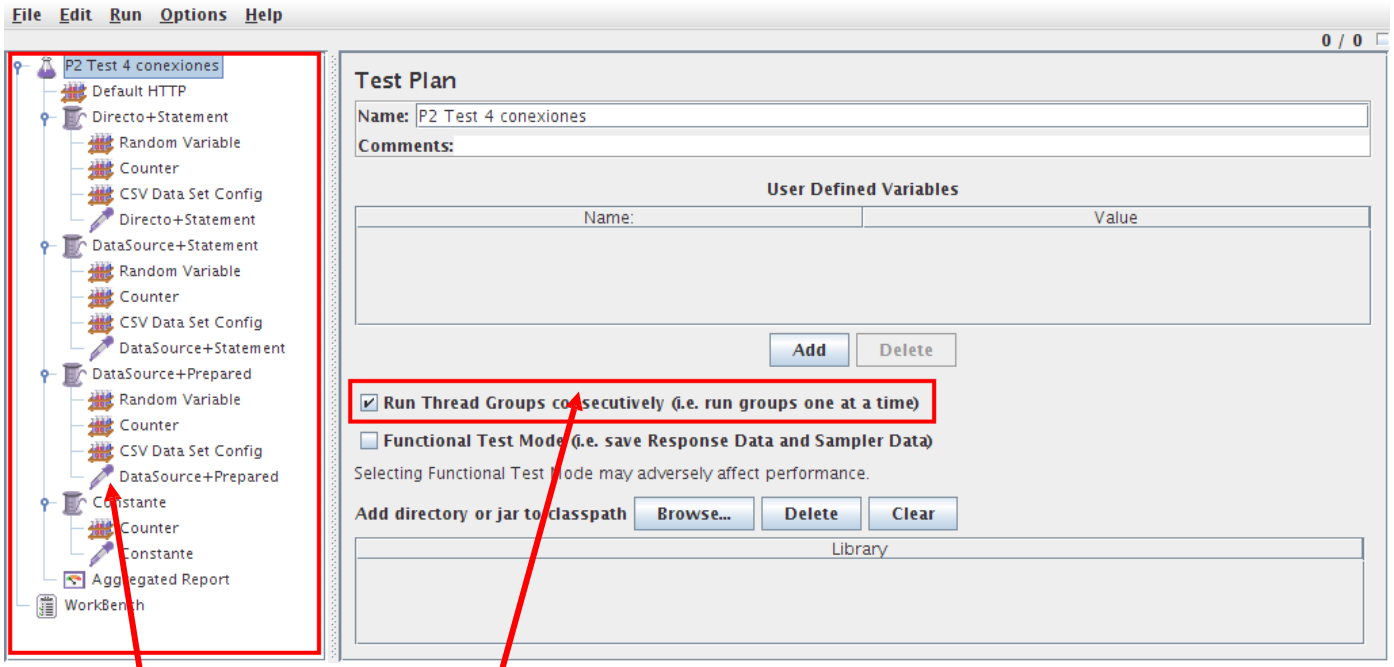
- **Prueba de carga**
  - Carga de usuarios esperada en ambiente de producción
- **Prueba de stress**
  - Encontrar la carga máxima de usuarios en la que la aplicación comienza a fallar



Fuente: https://www.guru99.com/jmeter-performance-testing.html

Apache JMeter

Jmeter es un software de código abierto diseñado en Java para realizar test de carga, pruebas funcionales y medición de rendimiento de aplicaciones web, bases de datos y otros recursos



- 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.

Apache JMeter

Podemos añadir los diferentes elementos al árbol

- **Thread Group (Grupo de Threads):** 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).

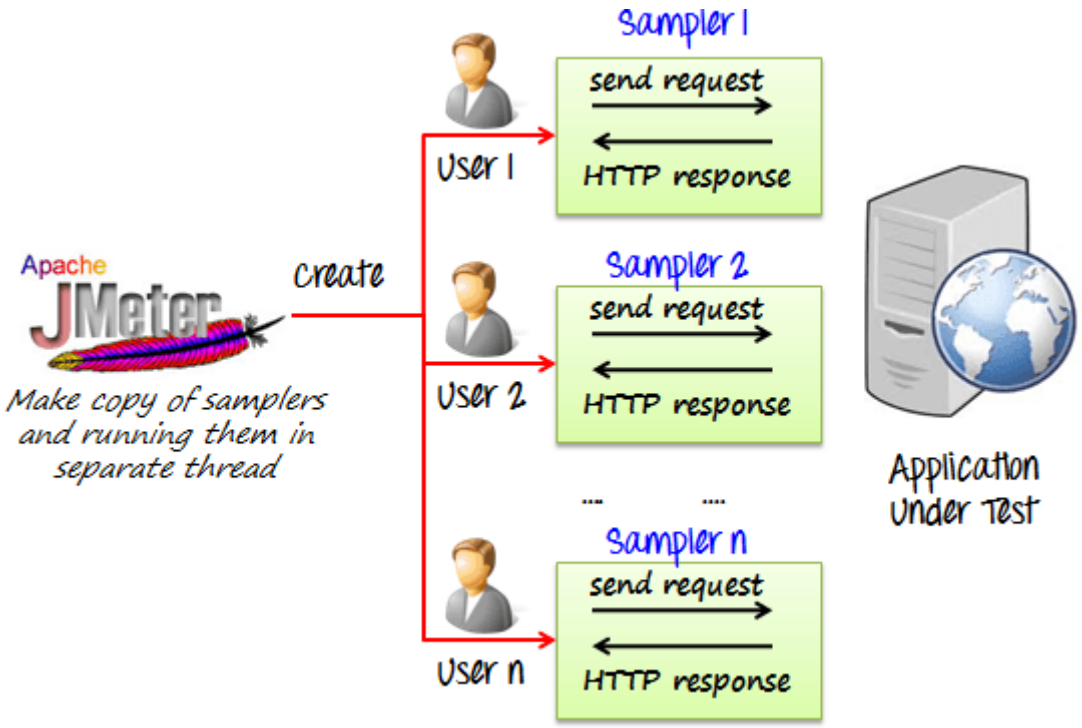
- **Config Element (Elemento de configuración):** Define un conjunto de parámetros que pueden ser reutilizados en diferentes grupos de hilos.

- **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 las peticiones de los samplers, 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

Apache JMeter

- **Simulación de usuarios (hilos)**
  - Cada usuario simulado puede mandar una o varias peticiones HTTP automáticas mediante los llamados muestreadores (samplers)

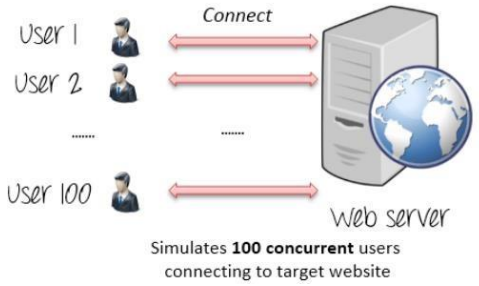


Fuente: https://www.guru99.com/jmeter-performance-testing.html

Apache JMeter

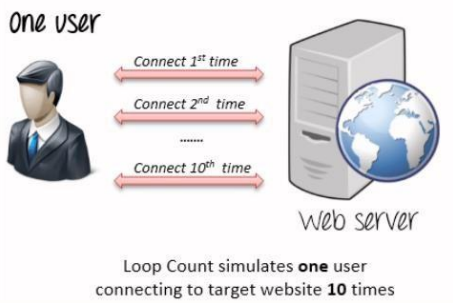
- **Grupo de usuarios (Thread Pool)**
  - *Number of threads*: número de usuarios virtuales concurrentes que se van a conectar a la aplicación
  - *Ramp-up period:* Periodo de despliegue del número total de usuarios virtuales.
  - *Loop Count:* Número de veces que se ejecutará cada usuario
- **Ejemplo: Thread Count = 20, Ramp Up Time (Seconds) = 100 & Loop Count = 4**
  - Cada 5 segundos, 1 usuario/hilo se crea y comienza inmediantamente a acceder a la app hasta que a los 100 segundos se hayan creado 20 usuarios. El proceso termina cuando los 20 usuarios ejecutan sus peticiones 4 veces.

Number of threads



Ramp-up period

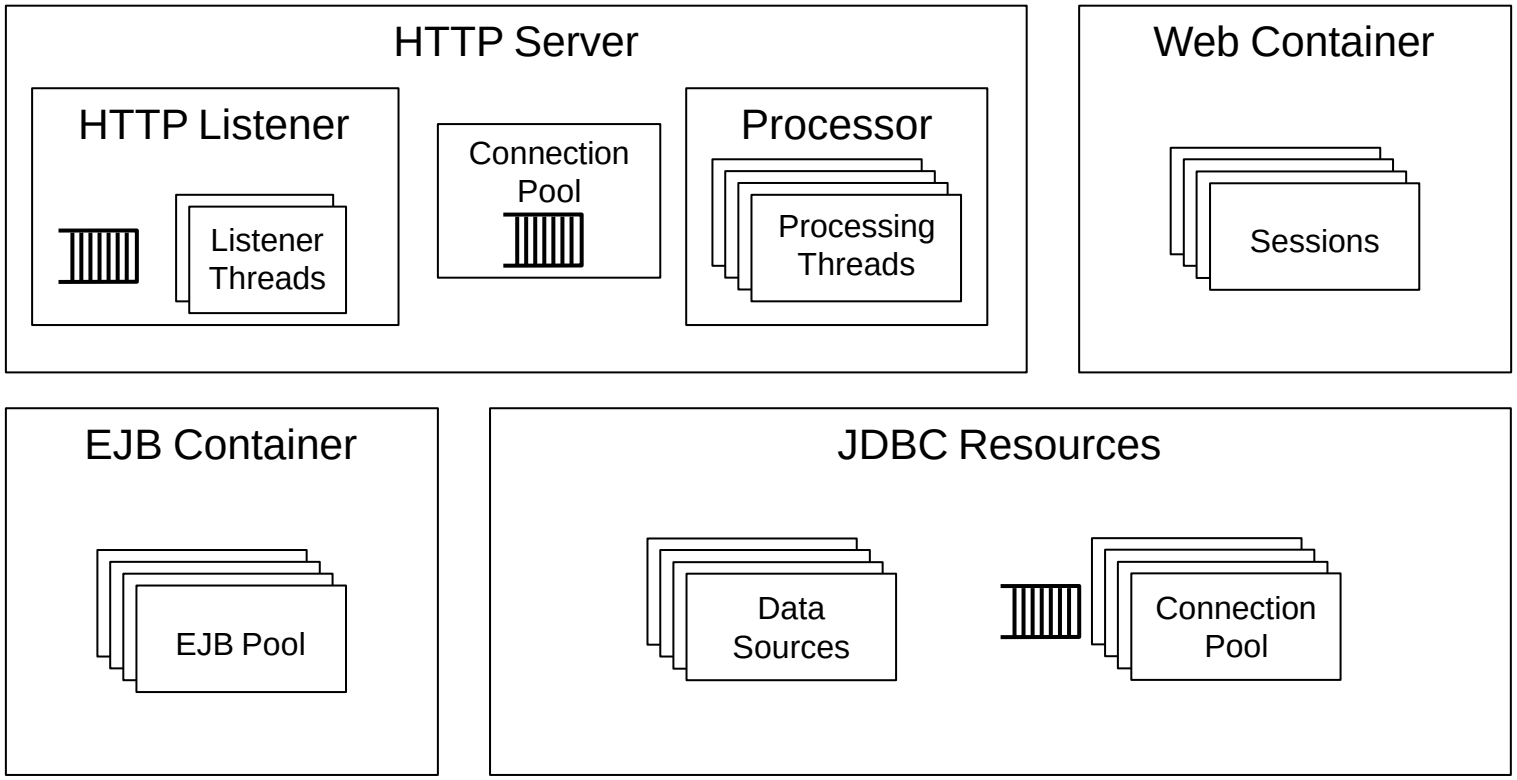
Loop count

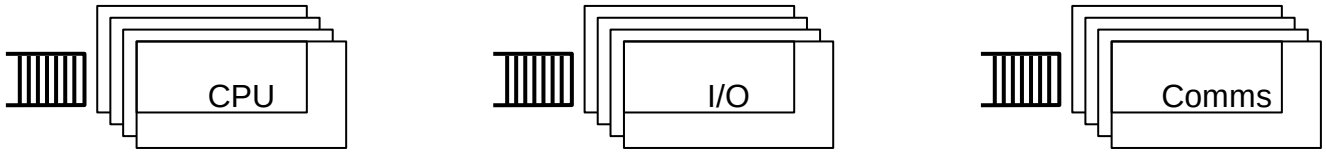


En cuanto al rendimiento, el enunciado de la práctica contiene un apéndice en la que se explican los diversos factores que influyen en el rendimiento de una aplicación JavaEE, que el alumno debe leer en detalle y comprender para abordar la ejecución de la práctica

- Aspectos generales del rendimiento en aplicaciones Java EE
  - *Best Practices* de Java orientadas al rendimiento
  - Configuración de la *Java Virtual Machine*.
  - Configuración del *hardware* y del sistema operativo.
- Cadena de procesamiento de una aplicación Java EE.
- Parámetros de configuración de los elementos de la cadena de procesamiento de cara al rendimiento.
  - Servicio HTTP.
  - Contenedores Web.
  - Contenedores de EJBs.
  - Recursos JDBC

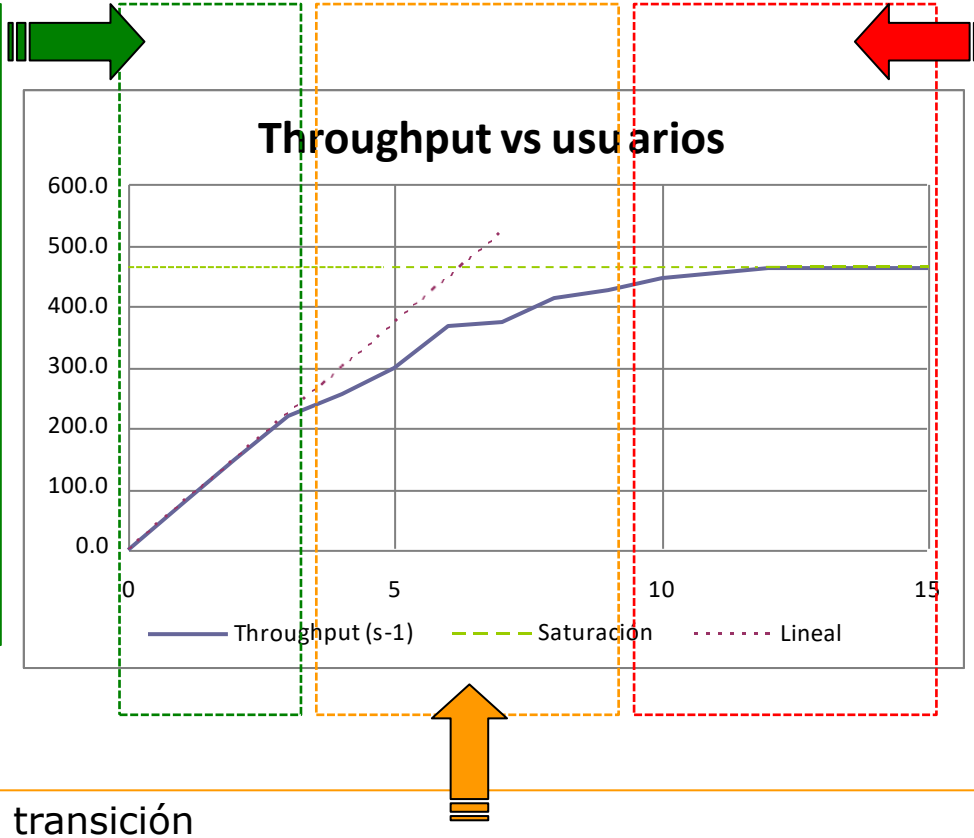
Los elementos de la cadena de procesamiento no son independientes, ya que comparten recursos en un ordenador. El sistema global es un sistema de colas complejo, en el que el primer elemento que se sature fijará el rendimiento global





La curva de rendimiento muestra la productividad *(throughput)* de una aplicación en función del número de usuarios conectados. Es decir, nos presenta el número de transacciones por segundo (o por minuto) que procesa a diferentes niveles de carga

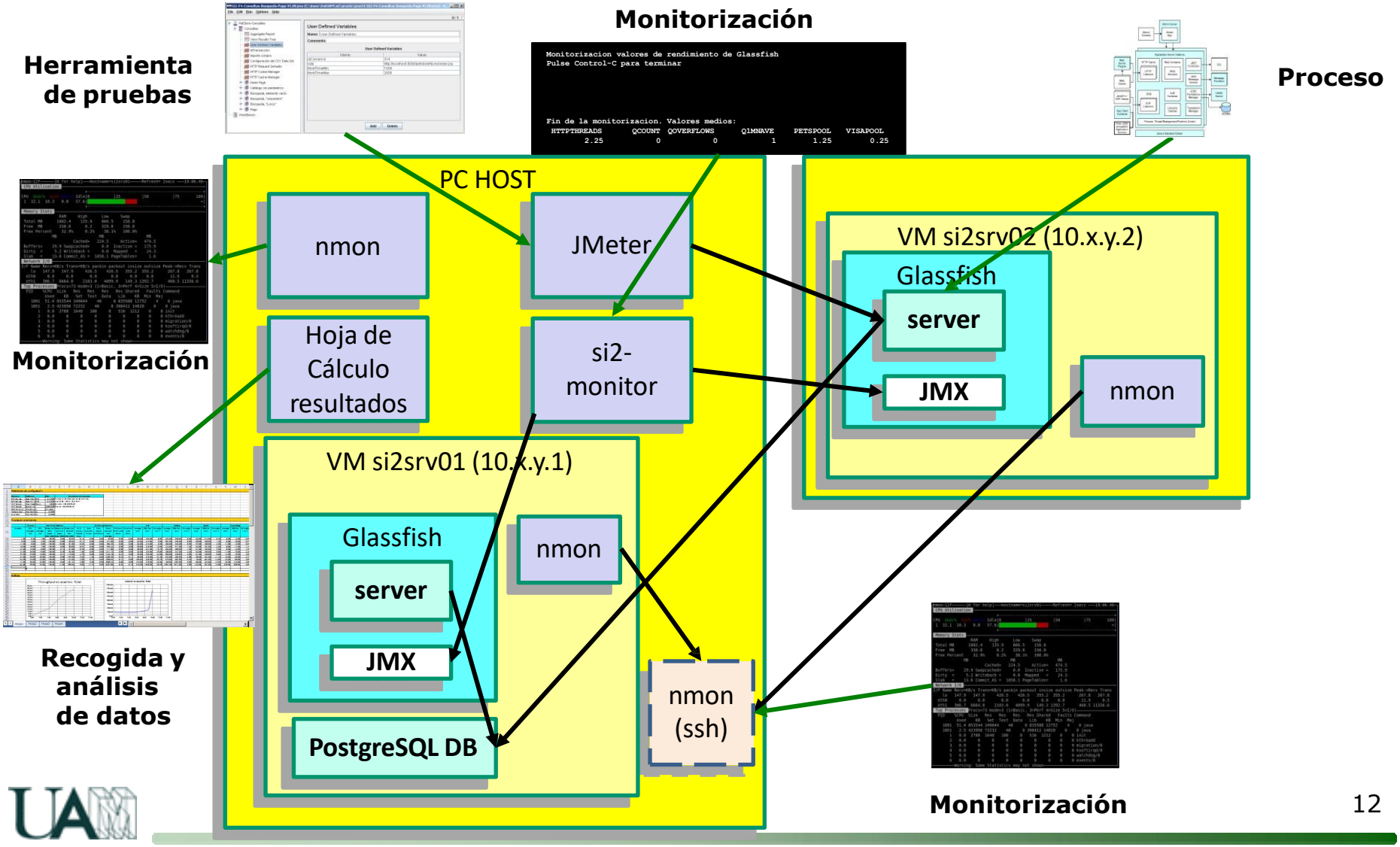
1. Zona lineal
   - Bajo número de usuarios
   - *Throughput* crece linealmente con el número de usuarios
   - No hay saturación en ningún punto de la cadena de procesamiento.
   - La latencia del sistema permanece constante al aumentar los usuarios.



3. Zona de saturación
   - Alcanzada o excedida la capacidad de proceso en algún punto de la cadena de procesamiento.
   - *Throughput* constante.
   - La latencia del sistema aumenta significativamente con el número de usuarios.
   - Se rechazan peticiones. Impacto en la disponibilidad.

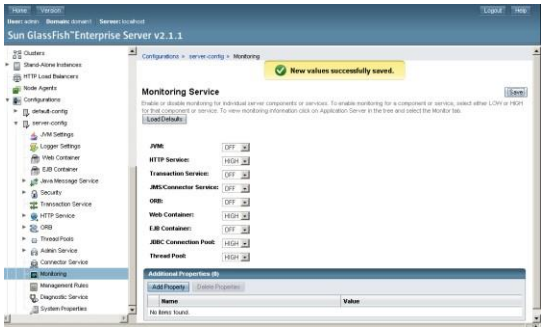
2. Zona de transición
   - Comienza a haber esperas en elementos de la cadena de procesamiento.
   - Crece el *throughput*, pero pierde la linealidad.
   - Aumenta la latencia del sistema.

El entorno de pruebas de que se dispone es limitado. Por este motivo, en la práctica se prescinde de la realización de estimaciones de escalado para reflejar un entorno real, limitándonos a obtener la curva como si el entorno utilizado fuera idéntico al de producción final



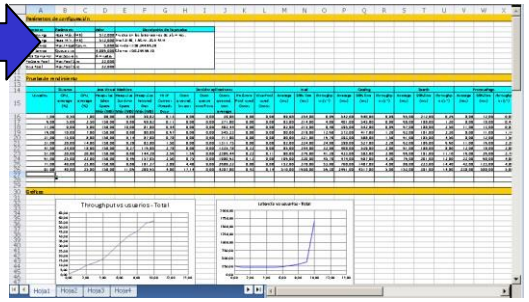
Antes de comenzar las pruebas, es necesario realizar una configuración adecuada del servidor la cual es orientada al rendimiento. Se proporcionan los principales parámetros que son necesario modificar sobre una configuración por defecto de Glassfish. Esta configuración debe permanecer estable a lo largo de todas las pruebas

1. Editar los parámetros en la consola de administración de Glassfish.



2. Registrar los parámetros de configuración requeridos

Parámetro	Posición en consola administración
Valores máximo y mínimo del heap de memoria que utiliza la máquina virtual Java	Application Server -> Pestaña JVM Settings -> Pestaña JVM Options Parámetros -Xmx y -Xms
Máximo número de conexiones a procesar simultáneamente por el servidor web	Configuration -> HTTP Service -> Pestaña Request Processing->Parámetro Thread Count
Tamaño máximo de la cola de conexiones pendientes de servicio	Configuration -> HTTP Service ->Pestaña Connection Pool->Parámetro Queue Size
Máximo número de sesiones en el contenedor Web. (Valor por defecto = -1. Ilimitadas)	Configuration -> Web Container -> Pestaña Manager Properties ->Parámetro Max Sessions
Máximo número de conexiones en *pools* JDBC	Resources -> JDBC -> Connection Pools Petstore Pool -> Parámetro Maximum Pool Size Resources -> JDBC -> Connection Pools Visa Pool -> Parámetro Maximum Pool Size



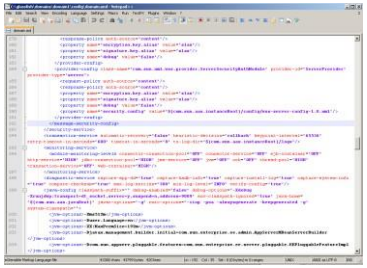
3. Desplegar las aplicaciones

¡Ojo! Configurar práctica 2:

- **debug=off,**
- **uso de DataSource y Prepared Statements,**
- **ProcesaPago lee parámetros recogidos por ComienzaPago**

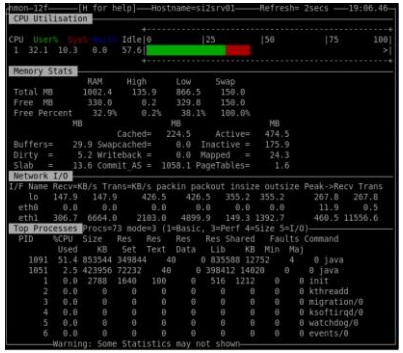
4. Rearrancar el servidor

5. Guardar domain.xml

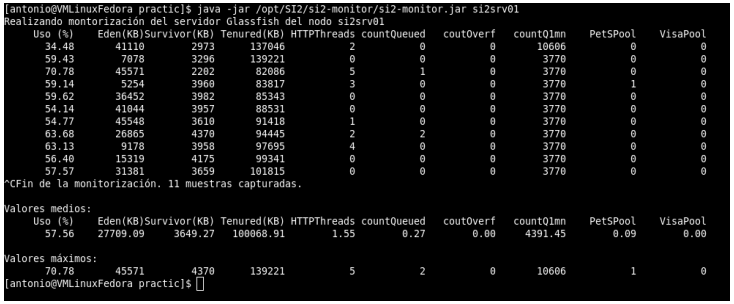


Tras la configuración del servidor de aplicaciones, es necesario realizar una serie de operaciones para preparar el entorno de pruebas y monitorizar adecuadamente la misma

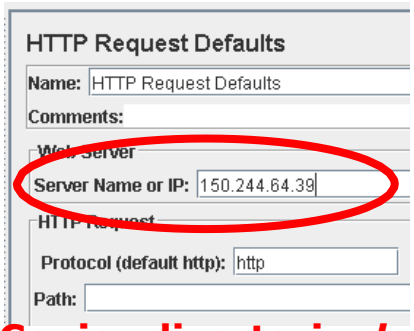
1. Arrancar la herramienta de monitorización del sistema en ambos ordenadores *(nmon)*



2. Preparar el arranque del programa de monitorización del servidor de aplicaciones (si2-monitor)



3. Poner la dirección del servidor de destino en el *script* de pruebas de JMeter



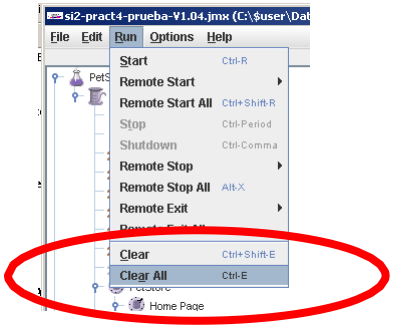
4. Ejecutar unos ciclos de prueba con JMeter sin tomar resultados para preparar todo el entorno

Comprobar con el listener *View Results Tree* que la ejecución es correcta y no hay errores.

¡Ojo! Copiar directorio *datagen*

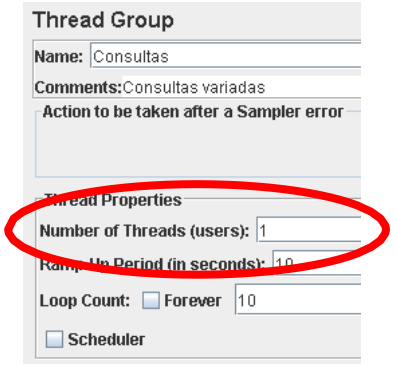
La curva de rendimiento se obtendrá punto a punto mediante distintas ejecuciones del *script* de pruebas de Glassfish. Cada ejecución nos dará un punto de la curva. Los pasos a seguir para cada ejecución son los siguientes:

1. Borrar los resultados de la ejecución anterior

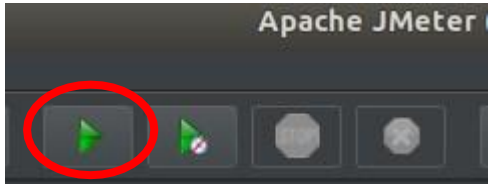


2. Borrar el contenido de la tabla de pagos de la base de datos Visa

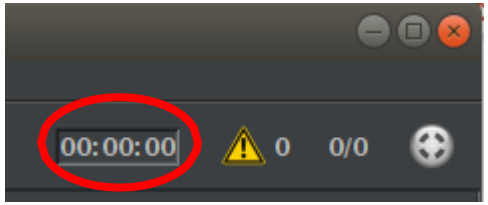
3. Seleccionar el número de usuarios para la prueba



4. Conmutar el JMeter a la ventana Aggregate Report y comenzar la prueba



5. Arrancar el programa de monitorización (si2-monitor.sh) al terminar la rampa de subida de usuarios



6. Al mismo tiempo que si2monitor, arrancar también la monitorización del PC donde se ejecuta JMeter y servidor de aplicaciones con nmon, en modo interactivo (I) o Data-collect (II) (*)(**)

(I) `victor@victor-pc ~> nmon`

(II) `si2@si2srv02:~$ nmon -f`

* Ajustar los argumentos –s y –c según el tiempo estimado de la prueba

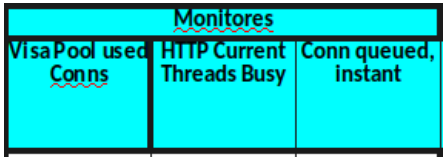
** Alternativamente, dado que sólo estamos interesados en el consumo de CPU medio de la máquina donde se encuentra el servidor de aplicaciones, podemos ejecutar el comando:
vmstat -n 1 | (trap '' INT; awk '{print; if(NR>2) cpu+=$13+$14;}END{print"MEDIA"; print "NR:",NR,"CPU: ", cpu/(NR-2);}';)

La curva de rendimiento se obtendrá punto a punto mediante distintas ejecuciones del *script* de pruebas de Glassfish. Cada ejecución nos dará un punto de la curva. Los pasos a seguir para cada ejecución son los siguientes:

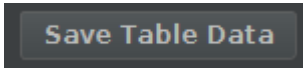
6. Parar las monitorizaciones (Ctrl-C) al comenzar a decrecer el número de usuarios.

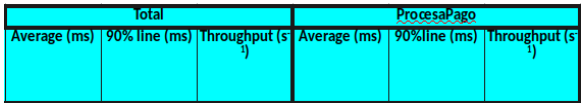
```
[antonio@VMLinuxFedora practic]$ java -jar /opt/SI2/si2-monitor/s
Realizando montorización del servidor Glassfish del nodo si2srv01
    Uso (%)    Eden(KB)Survivor(KB) Tenured(KB) HTTPThreads coun
      34.48       41110        2973      137046           2
      59.43        7078        3296      139221           0
      70.78       45571        2202       82086           5
      59.14        5254        3960       83817           3
      59.62       36452        3982       85343           0
      54.14       41044        3957       88531           0
      54.77       45548        3610       91418           1
      63.68       26865        4370       94445           2
      63.13        9178        3958       97695           4
      56.40       15319        4175       99341           0
```

7. Salvar y registrar en la hoja de cálculo los valores promedios del resultado de si2-monitor.sh



8. Salvar el Aggregated Report de JMeter y registrar resultados en la hoja de cálculo

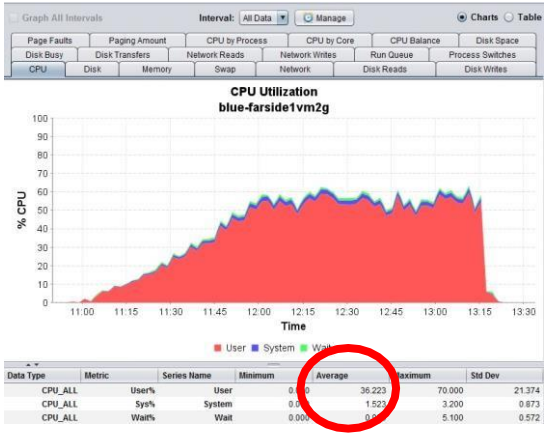


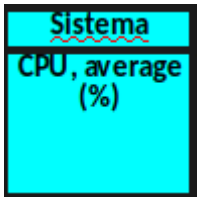


9. Copiar el fichero .nmon de la prueba desde el servidor al PC local via scp, y analizarlo mediante NMONVisualizer para obtener el CPU promedio y registrarlo en la hoja de cálculo (*) (**)

```
si2@si2srv02:~$ ls
si2srv02_190308_0339.nmon
```

```
victor@victor-pc ~> scp si2@10.1.7.2:/home/si2/si2srv02_190308_0339.nmon ./
si2@10.1.7.2's password:
si2srv02_190308_0339.nmon                     100%   14KB  10.9MB/s   00:00
```





(*) Si se usa nmon en modo interactivo, se deben sacar pantallazos del estado de la CPU durante la prueba y calcular la media a posteriori

(**) En caso de haber usado el comando vmstat para extraer directamente el consumo medio de CPU, el valor de CPU average se sacará directamente del resultado de este comando

Para interpretar los resultados de la prueba, es necesario tener en cuenta las medidas realizadas con la monitorización, y analizar su posible impacto en la saturación del sistema. Algunas preguntas a realizarse para este análisis son:

- ¿Hay CPU suficiente en el servidor para todo el trabajo que se está realizando?
- ¿Causa algún problema la red?
- ¿Tiene suficiente CPU la JVM para el trabajo del servidor de aplicaciones?
- ¿Se llena la memoria de la JVM?
- ¿Se satura el *pool* de proceso del servidor de aplicaciones?
- ¿Se satura el *pool* de conexión a la base de datos?
- ¿Se alcanza el valor máximo de sesiones permitido por el servidor de aplicaciones?

… y nunca hay que descartar que la prueba no se esté realizando bien…

- ¿Tiene CPU suficiente el ordenador donde se ejecuta el JMeter para crear todos los hilos que emulan a los clientes?
- ¿Es la monitorización la que hace que el servidor se sature?

Analizar los resultados, y estimar las causas

Entrega

- La entrega de los ejercicios de esta práctica se regirá por las normas expuestas de la asignatura.
- Nomenclatura del fichero a entregar SI2P2_<grupo>_<pareja>.zip (ejemplo: SI2P2_2311_1.zip)
- Contenido del fichero:
  - Código resultante de todas las modificaciones sugeridas: directorio P2.
  - Archivo JMX con el guion de pruebas (P2.jmx) y archivo P2-curvaProductividad.jmx.
  - Archivo SI2-P2-curvaProductividad.ods con los datos recogidos y la curva de productividad.
  - Archivo domain.xml con las modificaciones de la segunda parte de la práctica.
  - Memoria de la ejecución de la práctica con los resultados obtenidos en las tres partes de la práctica.
  - Ficheros que respalden los resultados de la curva de productividad (screenshots, .nmon, volcados de si2monitor...).
- La entrega se realizará la semana del **19 al 23 de abril de 2021.**
- En todos los casos, la entrega se realizará antes del comienzo de la siguiente sesión de prácticas.