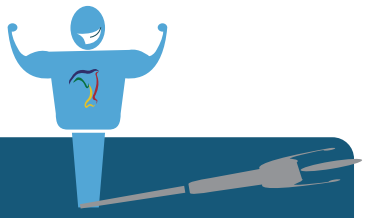
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

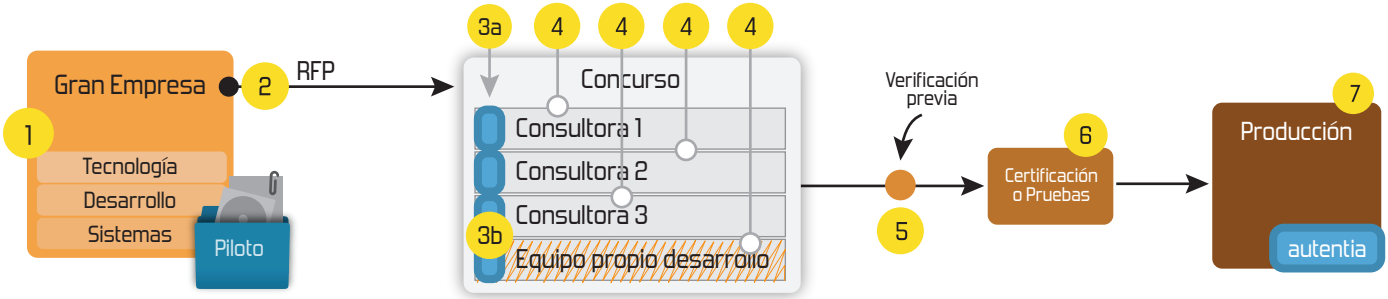
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

AdictosAlTrabajo
Terrakas 1x03 ¡¡Ya está en la web!!
terrakas.com

autentia
Soporte a desarrollo informático
Hosting patrocinado por enredados

Entra en Adictos a través de
E-mail
Contraseña
Entrar
Deseo registrarme
Olvidé mi contraseña

Inicio Quiénes somos Formación Comparador de salarios Nuestro libro Más

» Estás en: **Inicio** **Tutoriales** Configuración y tuning de servidores de producción

Alberto Barranco Ramón

Consultor tecnológico de desarrollo de proyectos informáticos.

Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/JEE

Ver todos los tutoriales del autor

Descarga
Aplicación gratuita nº1 de 2012
Smileys, emoticonos y guiños gratuitos para messenger y redes sociales. Descárgalos en nuestra página web.
Anuncios

Fecha de publicación del tutorial: 2009-02-26

Tutorial visitado 7 veces Descargar en PDF

Configuración y tuning de servidores de producción

0. Índice de contenidos.

1. Introducción

En este tutorial vamos a ver algunos de los distintos parámetros de configuración que podemos tocar para aumentar el rendimiento de nuestras aplicaciones en sistemas de producción.

En los distintos tutoriales que tenemos sobre las herramientas de medición de rendimiento se nos ha preguntado sobre casos específicos o sobre como interpretar los datos. Este tutorial pretende precisamente cubrir ese apartado. Es un pasito más, centrado en la toma de decisiones, detección de cuellos de botella y sus distintas soluciones.

Para poder sacar todo el partido a este tutorial el lector debería tener conocimientos sobre **JMeter** y **AppDynamics**. El resto de herramientas que vamos a ir usando se van a ir explicando todo lo detalladamente posible.

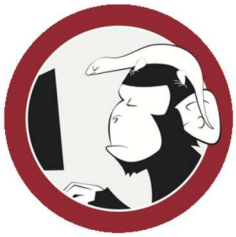
2. Entorno

- Hardware: Portátil MacBook Pro 15' (2.0 GHz Intel i7, 8GB DDR3 SDRAM, 500GB HDD).
- AMD Radeon HD 6490M 256 MB
- Sistema Operativo: Mac OS X Snow Leopard 10.6.7
- Software: Apache/2.2.17 (Unix), java version 1.6.0_31, JMeter 2.7, Apache-Tomcat-7.0.14

3. Introducción y montado de una arquitectura de producción

En el contexto de las aplicaciones web, las arquitecturas de producción no solo cuentan con un Tomcat al que todos los clientes atacan, sino que suele ser algo más parecido a la imagen siguiente, y es muy parecida a la que vamos a montar en este tutorial.

Catálogo de servicios Autentia



Síguenos a través de:

Últimas Noticias

» Autentia colabora con la ONG Proyecto Ciclista Solidario

» Curso de Kanban Core en Madrid con Masa K. Maeda

» Failure demand

» Comentando el Libro: Lean StartUp (el método)

» VIII Autentia Cycling Day

Histórico de noticias

Últimos Tutoriales

» Revisión de jBPM5

» Database MessageSource: obtener los literales de una base de datos

» Comparando diferencias entre ficheros con java-diff-utils

» Mejorando la calidad de nuestro código PHP

» JQuery Waypoints: realizando acciones al llegar a un punto de la página con el scroll.

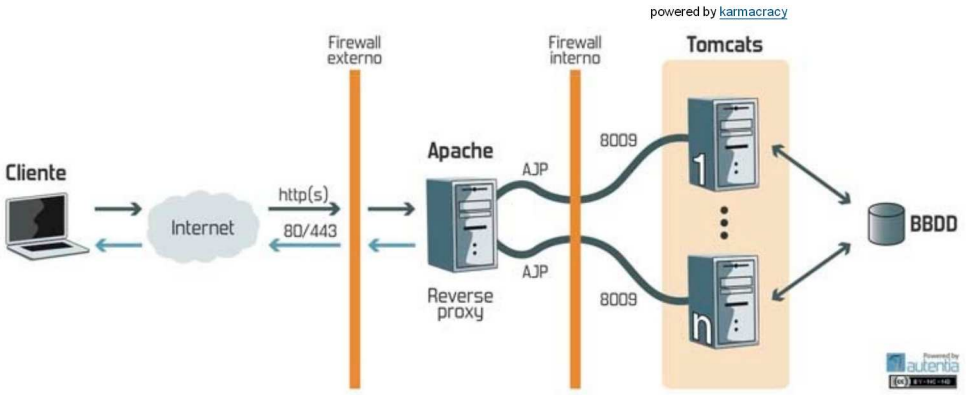
Últimos Tutoriales del

Impulsores Comunidad ¿Ayuda?

---- sin clicks

0 personas han traído clicks a esta página

+ + + + + + + +



Esto se hace así por varios motivos:

- Primero porque teniendo un Apache delante del Tomcat, se puede servir el contenido estático directamente por él, ya que es para esto para lo que mejor nos sirve un apache, y reducimos la carga en el Tomcat.
- Segundo porque lo podemos utilizar para balancear entre distintos Tomcats, por si necesitamos un sistema de alta disponibilidad o simplemente porque el número de usuarios en concurrencia requiere dos máquinas.

Comprobamos nuestra versión de java con el comando **java -version**. Si no tenéis instalado Java, podéis obtener la última versión de este enlace.

Descargamos el Apache Tomcat. Simplemente lo descomprimimos en el directorio que queramos. Podéis utilizar el comando **tar -xvf nombre_del_archivo.tar.gz**

Como vamos a hacer pruebas sobre una aplicación Web (cortesía de Francisco Javier), os la voy pasando para comprobar que la instalación del Tomcat ha sido correcta. Os podéis descargar la aplicación Web del siguiente enlace.

Como veis es un zip que contiene un **war** ,un fichero **c3p0.properties** y un **mysql-connector-java-5.1.11.jar**. El archivo **war** lo copiais en la carpeta **/TOMCAT_HOME/webapps/**, en donde **/TOMCAT_HOME/** es el directorio en el que habéis instalado el Tomcat. Los archivos **c3p0.properties** y **mysql-connector-java-5.1.11.jar** los copiais en **/TOMCAT_HOME/lib/**. Los más avispados de la clase se habrán dado cuenta de que vamos a utilizar base de datos. Para el ejemplo vamos a usar MySql. Os la podéis descargar del siguiente enlace.

Vamos a necesitar una base de datos con tan solo una tabla, aquí os dejo el script para que podáis crearosla fácilmente.

```
view plain print ?
01. CREATE DATABASE sorteo;
02.
03. USE sorteo;
04.
05. CREATE  TABLE `sorteo`.`participante` (
06.   `id` INT NOT NULL AUTO_INCREMENT ,
07.   `nombre` VARCHAR(45) NOT NULL ,
08.   `apellidos` VARCHAR(100) NOT NULL ,
09.   `nif` VARCHAR(10) NOT NULL ,
10.   `email` VARCHAR(45) NULL ,
11.   `direccion` VARCHAR(45) NULL ,
12.   PRIMARY KEY (`id`) )
13. ENGINE = InnoDB;
```

También vamos a necesitar un **usuario** sorteo con una **password** sorteo, que pueda acceder a dicha base de datos, es decir, con los privilegios necesarios. Lo podemos hacer con el interfaz gráfico pero os dejo los comandos para los que no dispongáis de uno.

```
view plain print ?
01. CREATE USER ´sorteo´@´localhost´IDENTIFIED BY ´sorteo´;
02. GRANT ALL PRIVILEGES ON ´sorteo´.* to ´sorteo´@´localhost´;
```

Ya tenemos todo lo necesario para que nuestra aplicación Web funcione, vamos a comprobar que estamos en lo cierto, y que la instalación que hemos hecho funciona. Abrimos un terminal y nos dirigimos a **/TOMCAT_HOME/**. Arrancamos el Tomcat con el siguiente comando **./bin/startup.sh**. Si todo ha ido bien, podremos acceder a la aplicación web en la siguiente url: http://localhost:8080/tuning-web/. Nos logamos con **usuario**:admin **password**:admin.

Ahora vamos a montar el **Apache**. Con mi sistema operativo tengo ya instalado uno, asique es el que voy a utilizar. Si estás utilizando un sistema operativo Unix puedes descargártelo con el comando **apt-get install apache2** y si etás en Windows te lo puedes bajar de este enlace.

Ya tenemos los dos servidores montados y funcionando. Ahora vamos a conectarlos mediante el protocolo **AJP 1.3** con **mod_proxy**. Abrimos primero el archivo **/TOMCAT_HOME/conf/server.xml**. Por defecto el Tomcat viene configurado para que el apache se pueda conectar a él mediante el puerto 8009, como podéis ver en la siguiente línea:

```
view plain print ?
01. <!-- Define an AJP 1.3 Connector on port 8009 -->
02. <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
```

Entonces lo único que tenemos que hacer es configurar el Apache. Vamos a editar ahora el archivo **/APACHE_HOME/**httpd.conf. En los que tengáis sistema operativo MAC, **APACHE_HOME** se encuentra en **/etc/apache2/**, y los que tengáis un sistema operativo Linux me parece que la instalación os la hace también en ese path, pero el archivo de configuración no se llama httpd.conf, sino apache2.conf. Al final da igual que editéis uno u otro porque si no estoy muy equivocado el fichero apache2.conf al final hace un include del httpd.conf, por lo que podéis editar el que queráis.

Autor

» Responsive CSS. Redimensionado de imagenes con Skeleton

» Analiza tu tráfico de red con Wireshark

» Spring mvc. Servicios Rest respondiendo en Json o XML

» Tapestry frente a JSF

» Agregar fuentes a la librería PdfBox

Últimas ofertas de empleo

2011-09-08
Comercial - Ventas - MADRID.

2011-09-03
Comercial - Ventas - VALENCIA.

2011-08-19
Comercial - Compras - ALICANTE.

2011-07-12
Otras Sin catalogar - MADRID.

2011-07-06
Otras Sin catalogar - LUGO.

Añadimos las siguientes líneas al final del fichero **/APACHE_HOME/**httpd.conf

```
view plain print ?
01. ProxyPass / ajp://localhost:8009/
02. ProxyPassReverse / ajp://localhost:8009/
```

Con la línea 1 estamos consiguiendo redirigir todo el tráfico al puerto 8009. En este caso nos da lo mismo pero si queréis afinar un poco más podéis poner lo siguiente:

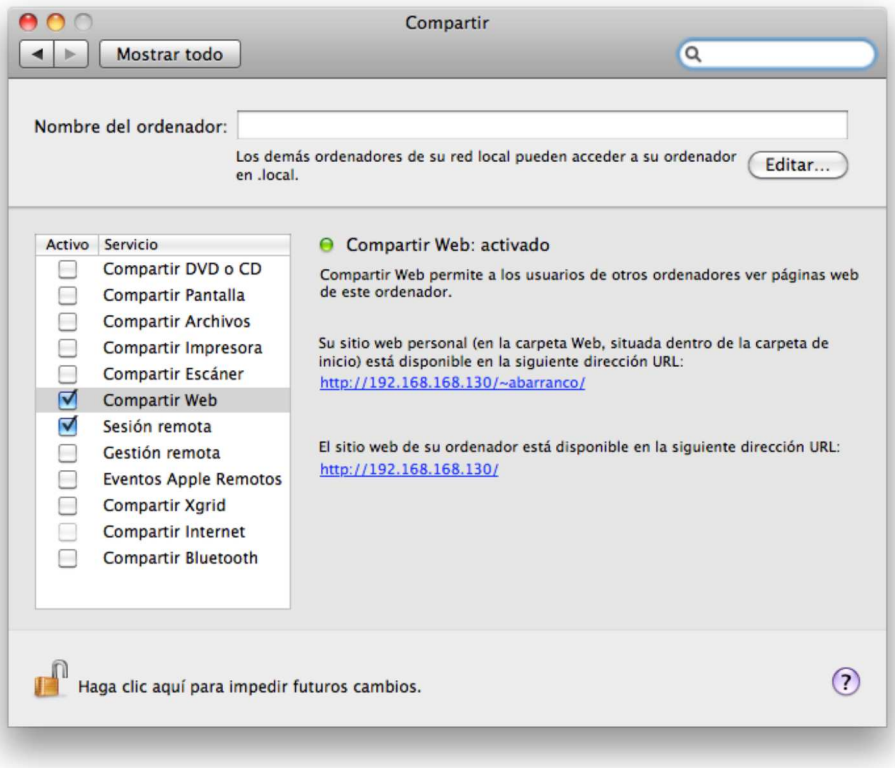
```
view plain print ?
01. ProxyPass /tuning-web/ ajp://localhost:8009/tuning-web/
02. ProxyPassReverse / ajp://localhost:8009/
```

Con eso conseguimos que todas las peticiones que van al contexto **/tuning-web/** se redirijan al Tomcat. Con la segunda línea lo que estamos haciendo es montar un Reverse Proxy, es decir, le permitimos al Apache cambiar las cabeceras *Location*, *Content-Location* y *URI* de las respuestas que nos manda el Tomcat, de tal forma que para un usuario externo, el que le está respondiendo es el Apache, y no un Tomcat por detrás.

Quiero aconsejaros un par de páginas de la documentación de **mod_proxy** por si os queda alguna duda o queréis ampliar algún concepto:

- http://httpd.apache.org/docs/2.2/mod/mod_proxy.html
- http://httpd.apache.org/docs/2.2/mod/mod_proxy_ajp.html

Reiniciamos el Apache para que coja los cambios que hemos hecho en la configuración. En Mac tenemos una forma muy fácil de reiniciar el Apache. Las imágenes siguientes hablan por si solas.



Vamos a hacer ahora la prueba de fuego. Abrimos un navegador y vamos a la siguiente url: http://localhost/tuning-web/. Nos debería salir lo siguiente:

Si todo es correcto, ¡genial!, quiere decir que hemos conectado el Apache con el Tomcat.

A partir de ahora vamos a necesitar **JMeter**, ya que vamos a simular carga en la aplicación para ver distintas situaciones y como podríamos arreglarlas. Podéis descargalo desde la página oficial.

Una vez lo tenemos instalado vamos a hacerle unos cambios ya que le vamos a meter un poco de caña y vamos a necesitar más memoria de la que trae configurada por defecto. Editamos el fichero **/JMETER_HOME/**bin/jmeter.

```
view plain print ?
01. # This is the base heap size -- you may increase or decrease it to fit your
02. # system's memory availablity:
03. HEAP="-Xms1024m -Xmx2048m"
```

Le he puesto 2GB porque en este ordenador tengo 8GB y ahora mismo con todo funcionando tengo incluso otros 4 GB que no estoy usando, pero en casa si no disponéis de tanta memoria, ponerle menos. Tanto en windows como en Unix como en Mac, es el mismo fichero el que tenéis que tocar para configurar la memoria, pero si estás en windows para ejecutar JMeter tendrás que ejecutar jmeter.bat, los que estamos con Mac/Unix jmeter.sh.

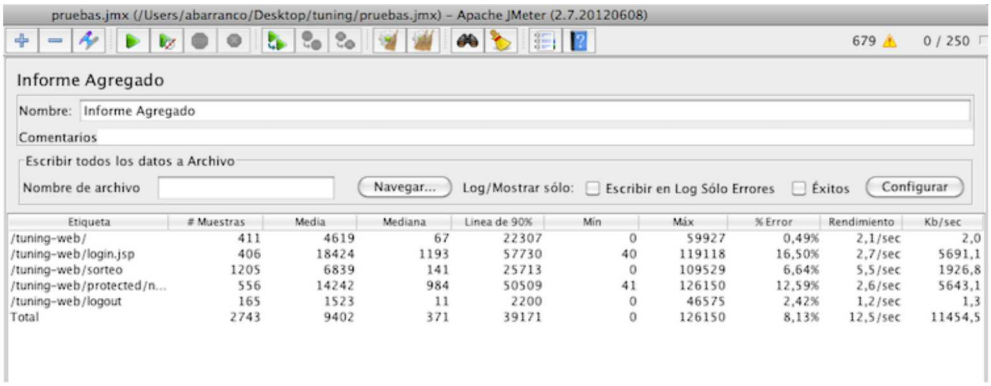
Aquí tenéis un plan de pruebas sencillito que me he creado grabando una navegación por la aplicación web. Si queréis saber como se hace esto, aquí tenéis un tutorial que lo explica muy bien: **JMeter, pruebas de stress sobre aplicaciones web: Grabando y reproduciendo navegaciones**

Vamos a necesitar también **AppDynamics Lite** para ver donde se nos va el tiempo. Aquí os dejo un tutorial que explica perfectamente como os lo podéis descargar e instalar: **AppDynamics Lite, encontrar problemas de rendimiento en aplicaciones Java en un entorno de producción**

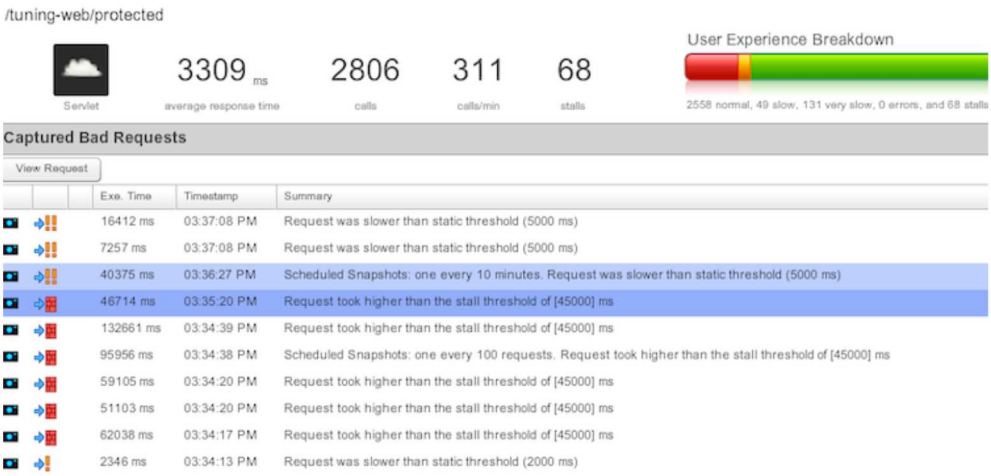
Pues nada, ejecutamos JMeter con el comando **./JMETER_HOME/**bin/jmeter.sh y pinchamos en abrir y seleccionamos el plan de pruebas que os acabais de descargar (**pruebas.jmx**). Sin más dilación le damos al play y nos posicionamos en Informe Agregado.

A medida que más usuarios están haciendo peticiones el tiempo máximo de cada petición sube. Ahora mismo tenemos 250 usuarios creando participantes y sorteando premios entre ellos, mientras se logan y se salen de la aplicación.

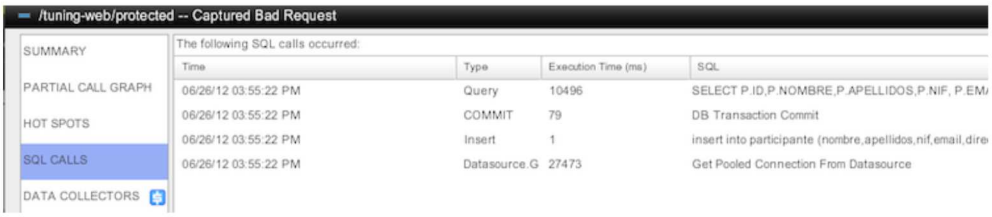
Tras un breve tiempo ya empiezan a aparecer los primeros errores, como puede verse en la imagen



Abrimos también un navegador y accedemos a la consola de AppDynamics en: http://localhost:8990/. Podemos observar por ejemplo peticiones de este tipo.



... que si la analizamos en profundidad ...



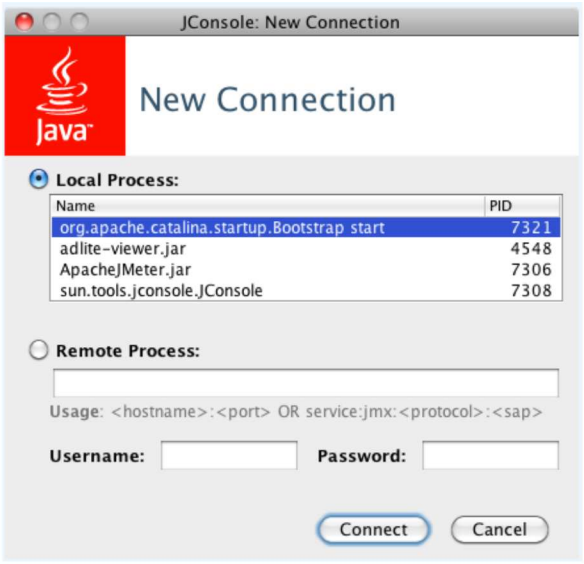
... nos damos cuenta de que lo que en realidad está penalizando es el obtener una conexión a la base de datos, de hecho espera 27 segundos hasta poder acceder a una conexión. Esto nos indica lo siguiente:

- Estamos gestionando mal las conexiones a la base de datos. No las cerramos, abrimos más de la cuenta ...
- Tenemos un pool de conexiones que no aguanta 250 usuarios en concurrencia.

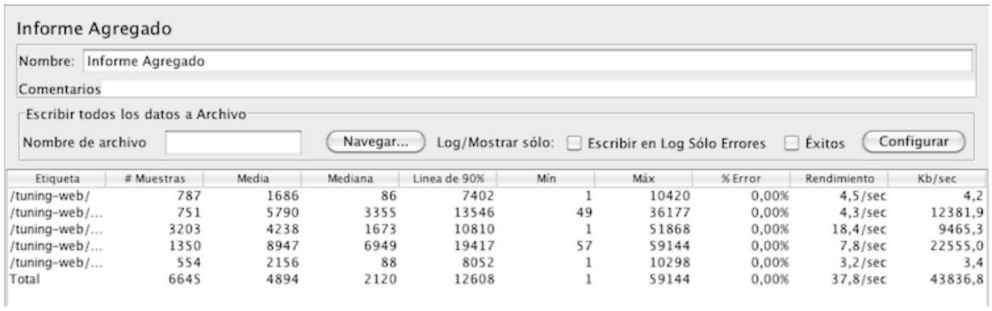
3.1 Configuración del pool de conexiones a la base de datos

La configuración a la base de datos la tenemos precisamente en el archivo **c3p0.properties** que copiamos en /TOMCAT_HOME/lib. Si abrimos este fichero, vemos como tenemos seteado el atributo **c3p0.maxPoolSize** a tan solo **10 conexiones**. Vamos a probar a aumentar el atributo a **100**, reiniciamos el Tomcat y repetimos la prueba. Esta vez vamos a monitorizar también con **jconsole**, por lo que abrimos un terminal y simplemente con el comando **jconsole**, se nos abrirá esta herramienta.

Seleccionamos el servidor Tomcat en la lista y damos a conectar.



En la siguiente imagen vemos como se ha aumentado el rendimiento tras el cambio a las **100 conexiones**.



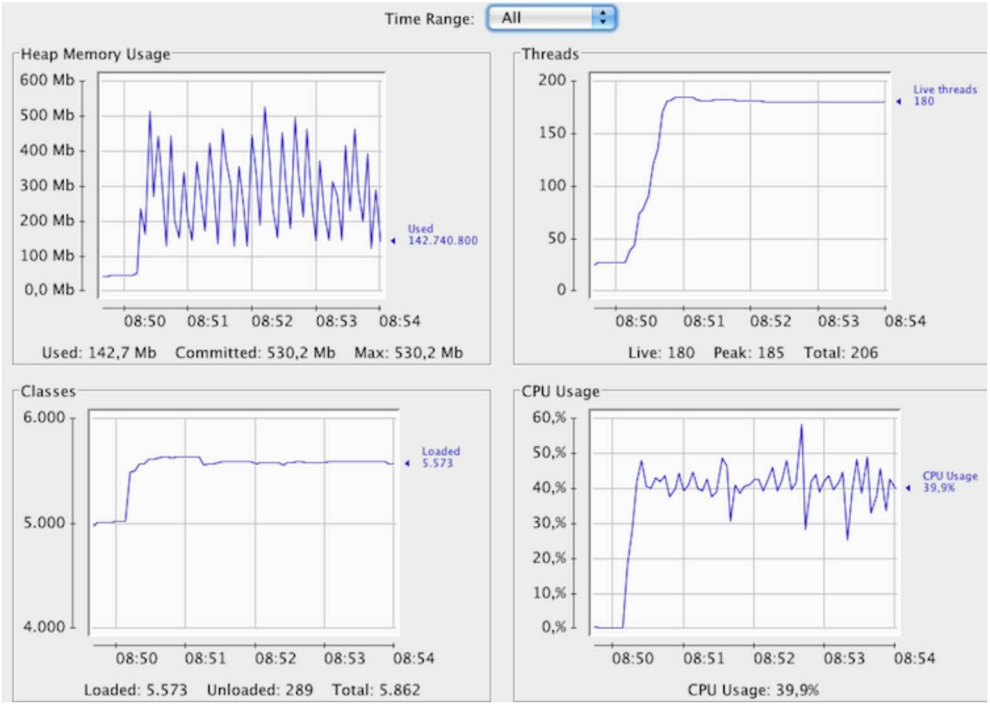
Etiqueta	# Muestras	Media	Mediana	Linea de 90%	Min	Máx	% Error	Rendimiento	Kb/sec
/tuning-web/	787	1686	86	7402	1	10420	0,00%	4,5/sec	4,2
/tuning-web/...	751	5790	3355	13546	49	36177	0,00%	4,3/sec	12381,9
/tuning-web/...	3203	4238	1673	10810	1	51868	0,00%	18,4/sec	9465,3
/tuning-web/...	1350	8947	6949	19417	57	59144	0,00%	7,8/sec	22555,0
/tuning-web/...	554	2156	88	8052	1	10298	0,00%	3,2/sec	3,4
Total	6645	4894	2120	12608	1	59144	0,00%	37,8/sec	43836,8

El rendimiento se ha aumentado en un **300%**, y ya no hay errores. Podemos ver lo importante que es este parámetro. Para sacar este dato lo que tenéis que comparar es el **rendimiento** (también llamado **throughput**) de la primera prueba (**12,5 por segundo**) con el de la segunda (**37,8 por segundo**).

Si nos fijamos ahora en AppDynamics podemos ver que hay algunas consultas que están tardando mucho en obtener una respuesta, pero ya no es por obtener el pool de conexiones. Por la propia prueba de la aplicación y por la aplicación en si, tenemos que 250 usuarios están continuamente consultando y escribiendo en **UNA sola tabla**, por lo que se va mucho tiempo en la gestión de la transaccionalidad, como es lógico. Si abrís las consultas **SQL** que se ejecutan en las peticiones lo podéis observar. Aquí no podemos hacer nada.

2381 ms	08:53:45 AM	Scheduled Snapshots: one every 100 requests. Request was slower than static threshold (2000 ms)
47439 ms	08:52:55 AM	Request took higher than the stall threshold of [45000] ms
52999 ms	08:52:54 AM	Request took higher than the stall threshold of [45000] ms
50772 ms	08:52:49 AM	Request took higher than the stall threshold of [45000] ms
5113 ms	08:52:31 AM	Scheduled Snapshots: one every 100 requests. Request was slower than static threshold (5000 ms)
4477 ms	08:51:42 AM	Scheduled Snapshots: one every 100 requests. Request was slower than static threshold (2000 ms)
2879 ms	08:51:00 AM	Request was slower than static threshold (2000 ms)
5214 ms	08:50:55 AM	Request was slower than static threshold (5000 ms)

Vamos a ver si los resultados de jConsole nos dicen algo más.



Viendo esta gráfica podemos decir a simple vista 3 cosas:

- Primero: que la aplicación **está gestionando bien la memoria** (no parece haber memory leaks). Si veis la gráfica de **Heap Memory Usage**, está en serrucho, lo que indica que se libera memoria de forma correcta. Si tuviera una pendiente ascendiente, que no se recupera la memoria ... podríamos pensar que tenemos **memory leaks** Hemos llegado al pico de memoria, que lo tenemos puesto en 512 MB en 3 ocasiones, lo que nos hace ver que el uso de memoria es intensivo. Un aumento de esta podría beneficiar el rendimiento de la aplicación.
- Segundo: que la **CPU** va muy relajada. Es decir, si fuera por la CPU se podrían poner en estas condiciones otros 250 usuarios. Ahora mismo vamos a ver que en realidad tenemos otro cuello de botella que hace que no estemos con 250 usuarios.
- Tercero: que pese a lanzar la prueba con 250 usuarios el **Tomcat solo tiene 180 threads funcionando**, es decir, estos 80 hilos que faltan podrían estarse quedando en el Apache. Tenemos otro **cuello de botella**, vamos a ver si como sospechamos es el Apache.

3.2 Configuración de Threads de Apache y Tomcat

Vamos a probar a aumentar el número de Threads que puede atender el **Apache**. Lo primero que tenemos que hacer es, ver en que modo se está ejecutando nuestro servidor Apache. Para ello ejecutamos el siguiente comando en un terminal **httpd -V**. Este comando es válido tanto para **Windows**, como para **Mac**, em cambio si estás siguiendo el tutorial y haciendo las pruebas con un sistema operativo **Linux** el comando sería **apache2 -V**. El resultado sería algo parecido a lo siguiente:

```
localhost:apache-tomcat-streaming-pruebas abarranco$ httpd -V
Server version: Apache/2.2.17 (Unix)
Server built:   Dec  1 2010 09:58:15
Server's Module Magic Number: 20051115:25
Server loaded:  APR 1.3.8, APR-Util 1.3.9
Compiled using: APR 1.3.8, APR-Util 1.3.9
Architecture:   64-bit
Server MPM:     Prefork
  threaded:     no
    forked:     yes (variable process count)
Server compiled with....
 -D APACHE_MPM_DIR="server/mpm/prefork"
 -D APR_HAS_SENDFILE
 -D APR_HAS_MMAP
 -D APR_HAVE_IPV6 (IPv4-mapped addresses enabled)
 -D APR_USE_FLOCK_SERIALIZE
 -D APR_USE_PTHREAD_SERIALIZE
 -D SINGLE_LISTEN_UNSERIALIZED_ACCEPT
 -D APR_HAS_OTHER_CHILD
 -D AP_HAVE_RELIABLE_PIPED_LOGS
 -D DYNAMIC_MODULE_LIMIT=128
 -D HTTPD_ROOT="/usr"
 -D SUEXEC_BIN="/usr/bin/suexec"
 -D DEFAULT_PIDLOG="/private/var/run/httpd.pid"
 -D DEFAULT_SCOREBOARD="logs/apache_runtime_status"
 -D DEFAULT_LOCKFILE="/private/var/run/accept.lock"
 -D DEFAULT_ERRORLOG="logs/error_log"
 -D AP_TYPES_CONFIG_FILE="/private/etc/apache2/mime.types"
 -D SERVER_CONFIG_FILE="/private/etc/apache2/httpd.conf"
localhost:apache-tomcat-streaming-pruebas abarranco$
```

podéis consultar la documentación oficial sobre los módulos de multiproceso (MPM) en el siguiente enlace. En esta página básicamente nos está diciendo que los modos por defecto para sistemas **Unix es prefork** (tal y como nos decía httpd -V), y para sistemas **Windows es WinNT**. Si pinchamos en el enlace que se nos ofrece para MPM prefork, nos están diciendo que el número de hilos que ofrecen por defecto es **256**. El cuello de botella entonces no está en Apache !!!. Tiene que estar en el Tomcat. También nos dicen que si necesitamos ampliarlo o decrementarlo, la directiva que tenemos que usar es **MaxClients**. ¡OJO! los usuarios de windows, que la directiva es distinta, os la chivo, para sistemas Windows la directiva es **ThreadsPerChild**. Parámetros como estos, yo los considero importantes, y aunque por defecto se sirven 256 hilos, a mi me gusta poner la directiva en el Apache, ya que hoy me acuerdo que son 256 porque lo acabo de mirar en la documentación. Dentro de un mes seguro que no me acuerde, asique para no tener que volver a buscarlo, **voy a editar el fichero http.conf** para añadir lo siguiente: **MaxClients 300**. Voy a poner 300 para no dejarlo justo a 250 sino tener un pequeño margen. **Reiniciamos el Apache**.

Vamos a ver entonces el número de hilos que tiene configurado el Tomcat.

Como estamos conectando Apache con Tomcat, voy a leerme primero la documentación de AJP 1.3, a ver donde se puede cambiar este parámetro. Esta es la documentación de AJP 1.3 para Tomcat 7. En **maxThreads** nos está diciendo que podemos setear el número máximo de hilos que va a crear este conector, o sea el número de peticiones simultaneas que vamos a poder atender. Por defecto son 200. Además nos dice que si tenemos definido un **executor** y está asociado con este conector este atributo se ignora y se usa el del executor. Si nos vamos al fichero **/TOMCAT_HOME/**conf/server.xml, vemos que efectivamente tenemos definido un executor.

```
<!--The connectors can use a shared executor, you can define one or more named thread pools-->
<!--
<Executor name="tomcatThreadPool" namePrefix="catalina-exec-"
    maxThreads="150" minSpareThreads="4"/>
-->
```

... y que algunos conectores lo tienen asociado ...

```
<Connector executor="tomcatThreadPool"
           port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
```

... pero nuestro conector no lo está usando, asique le setemos el atributo maxThreads de nuevo a 300 para dejarnos un margen. Debería quedarnos de la siguiente forma:

```
<!-- Define an AJP 1.3 Connector on port 8009 -->
<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" maxThreads="300" />
```

... reiniciamos el Tomcat.

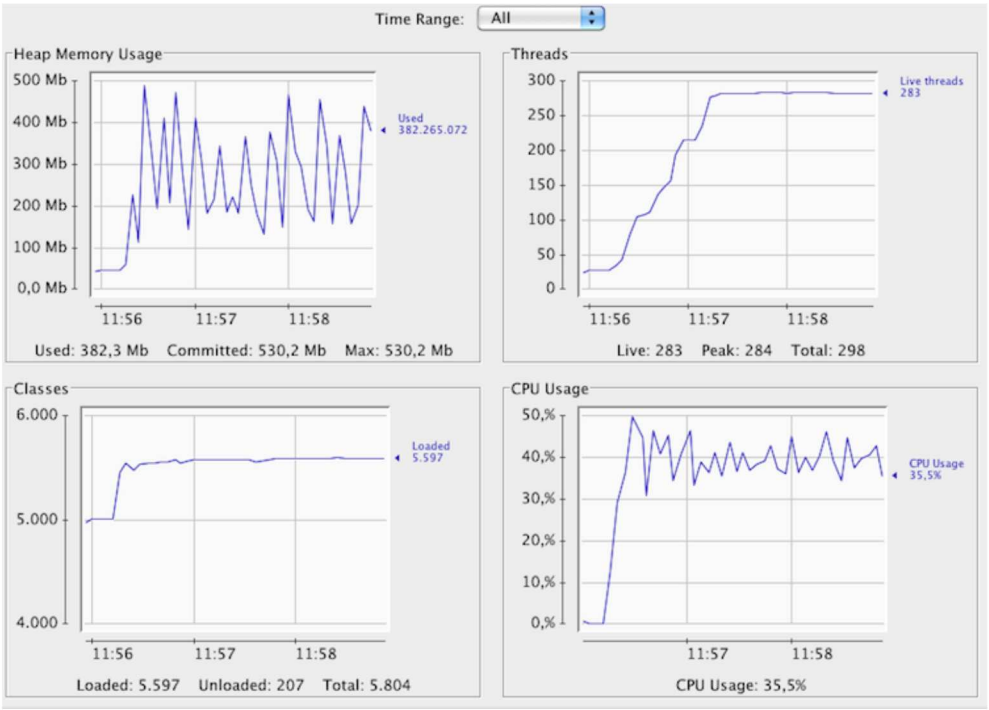
También hay otro parámetro a nivel de configuración del proxy en el Apache, que me parece importantísimo. Lo podéis ver en este enlace. Me refiero al parámetro **retry**. Tal y como veis en la documentación este parámetro está puesto por defecto a 60 segundos, lo que significa que cuando el Tomcat nos devuelve en una petición un error, Apache no le va a mandar más peticiones en 60 segundos. Esto puede pasar en picos altos de uso, en los que hay muchos usuarios y el Tomcat nos devuleve un **timeout** para una de ellas. No ha sido capaz de atender la petición y nos devuelve un error. Con la configuración por defecto tardaríamos 60 segundos en volver a mandar peticiones. Este tiempo es excesio y provoca que tengamos un rendimiento menor, si haceis pruebas con muchos usuarios observaréis que las peticiones van como en ráfagas y que hay tiempos en que la CPU está con un rendimiento del 0%, es porque cuando una petición falla tarda un rato en volver a mandar nuevas, y mientras tanto el servidor está totalmente relajado. Vamos a poner este parámetro a **5 segundos** y vamos a ver como **el rendimiento aumenta considerablemente**. Editamos el fichero **/APACHE_HOME/**http.conf y añadimos **retry=5** a la configuración del **ProxyPass**. Nos debería quedar de la siguiente forma:

```
MaxClients 300
ProxyPass / ajp://localhost:8009/ retry=5
ProxyPassReverse / ajp://localhost:8009/
```

... reiniciamos el Apache.

Pues nada, vamos a darle caña de nuevo con 250 usuarios a ver como afectan estos cambios al rendimiento.

Vamos a analizar primero las gráficas que nos muestra el **jConsole**.



Podemos sacar varias conclusiones:

- Primero: La **CPU** tiene **prácticamente el mismo uso que en la prueba anterior**. Esto es debido a que con 180 hilos o con 280 está trabajando lo mismo, es decir, está trabajando poco. Aquí quien está trabajando es la **BBDD**, ya que la aplicación es muy sencillita y lo único que hace es persistir usuarios en la misma tabla de base de datos. En un caso normal de aplicación tendréis más tablas, entrada/salida, algoritmos más complicados y estas optimizaciones os pondrán la CPU al 100%, y os subirán el rendimiento en un alto porcentaje.
- Segundo: **Hemos conseguido los 250 usuarios en concurrencia**, pero vemos que lo que realmente estamos midiendo aquí es la **transaccionalidad sobre una tabla gestionada por MySQL**, es decir, la base de datos no da para más. No podemos gestionar concurrencia y exclusión mutua sobre una tabla a más velocidad para 250 usuarios. Seguramente el rendimiento óptimo no sea con 250 usuarios. Antes teníamos unos 150 y teníamos más rendimiento. Para ver los usuarios en concurrente os tenéis que fijar en el número de threads que son **250 del JMeter + X threads que usa el Tomcat internamente = 283 Live threads**.

Etiqueta	# Muestras	Media	Mediana	Linea de 90%	Min	Máx	% Error	Rendimiento	Kb/sec
/tuning-web/	625	318	80	932	1	2948	0,00%	3,7/sec	3,5
/tuning-web/...	585	7895	4332	19601	69	82037	0,00%	3,5/sec	11705,5
/tuning-web/...	2358	4865	998	15906	1	76047	0,00%	13,9/sec	7671,7
/tuning-web/...	1011	13783	8942	34267	73	82083	0,00%	6,0/sec	20368,3
/tuning-web/...	375	230	56	833	1	1899	0,00%	2,3/sec	2,4
Total	4954	6118	1605	19781	1	82083	0,00%	28,9/sec	39176,1

Los resultados del **JMeter** van en consonancia. Ahora tenemos más usuarios, pero **la base de datos es la misma**, y seguimos insertando datos en una tabla, luego habiendo más usuarios a la vez tardaremos más, porque hay que gestionar más usuarios concurrentes, y el rendimiento en consecuencia baja. Hay que tener claro que el punto óptimo de una aplicación para una configuración determinada es X usuarios. Si te pasas de esos usuarios el rendimiento decrece, lo que no quiere decir que necesariamente la aplicación no escale, sino que tienes que adaptar la configuración a ello, por eso **lo primero que tienes que tener claro es el número de usuarios máximo que se espera en una aplicación, para configurar que la aplicación tenga un comportamiento óptimo para ese número**. Si luego se meten el doble de usuarios es obvio que el rendimiento decrecerá.

Con este tutorial hemos visto como interpretar las gráficas, e identificar posibles cuellos de botella. Me gustaría animar a los lectores a probar una aplicación desarrollada por ellos para ver su comportamiento e intentar ajustar la configuración para optimizarlo todo lo posible. Hay otros parámetros que podéis configurar para aumentar el rendimiento de vuestra aplicación. Os recomiendo este enlace.

4. Conclusiones

A nivel personal he descubierto un tema que me gusta bastante y he aprendido y recordado muchas cosas haciendo este tutorial. Espero que al lector le haya servido y parecido tan interesante como a mi. El rendimiento de las aplicaciones es algo que tenemos que tener muy en cuenta. Os voy a recomendar un libro que me ha ayudado y me ha parecido muy interesante: **Packtpub JBoss AS 5 Performance Tuning Dec 2010**. Un saludo !!

5. Información sobre el autor

Alberto Barranco Ramón es Ingeniero Técnico en Informática de Gestión y Graduado en Ingeniería del Software por la Universidad Politécnica de Madrid

Puedes estar al tanto de los nuevos tutoriales que publique en mi Twitter @barrancoalberto

Mail: abarranco@autentia.com.

Autentia Real Business Solutions S.L. - "Soporte a Desarrollo".

A continuación puedes evaluarlo:

Regístrate para evaluarlo

Por favor, vota +1 o compártelo si te pareció interesante

Share | 0

Anímate y coméntanos lo que pienses sobre este **TUTORIAL:**

» Registrate y accede a esta y otras ventajas **«**

SOME RIGHTS RESERVED Esta obra está licenciada bajo licencia Creative Commons de Reconocimiento-No comercial-Sin obras derivadas 2.5

Copyright 2003-2012 © All Rights Reserved | Texto legal y condiciones de uso | Banners | Powered by Autentia | Contacto

W3C XHTML 1.0 W3C CSS XML RSS XML ATOM