PARTE 3.- Experimento básico

1. Introducción

El objetivo de esta parte es comprender las características y parámetros de funcionamiento del escenario concreto a analizar, así como el procedimiento que se pretende seguir. Veremos cómo generar un plan de pruebas que sea representativo de una población grande de usuarios del servidor y diseñaremos los experimentos de análisis de prestaciones a llevar a cabo.

2. Comportamiento de los usuarios

En las clases de teoría se han estudiado sistemas de colas en los que las llegadas seguían un proceso de Poisson. Existen varios buenos motivos para centrarse en este modelo, uno de los cuales es que es tratable matemáticamente para ofrecer resultados analíticos. Otro motivo es que el resultado de superponer un gran número de procesos independientes, con muy pocas condiciones adicionales sobre estos procesos, cumple que tiende a un proceso de Poisson al ir aumentando el número de procesos multiplexados.

En nuestro escenario, JMeter puede simular procesos ON-OFF donde el intervalo ON es aquel entre que realiza una petición y termina de obtener la respuesta y el intervalo OFF el tiempo de espera hasta hacer otra petición. Podemos suponer que aproximadamente durante todo ese periodo ON el servidor web tiene un proceso/hilo[1] ocupado. La superposición (o multiplexación) de procesos de llegadas de este tipo se lleva a cabo sin más que indicando un número de hilos (Ilustración 1).

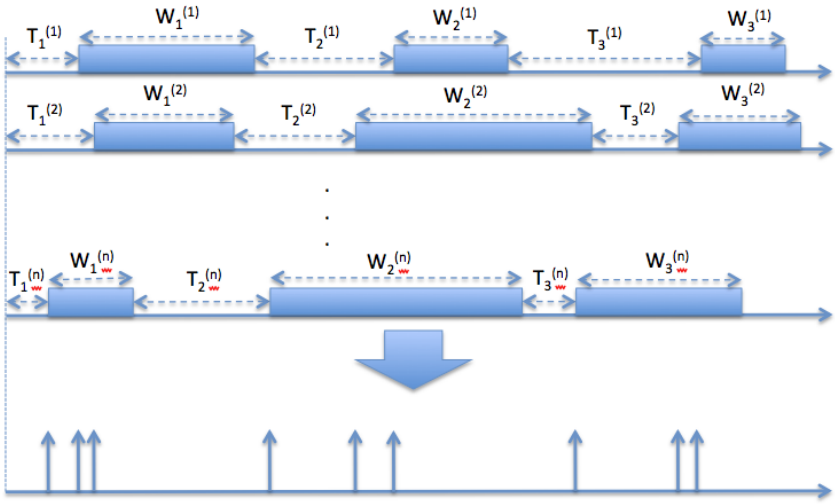

Ilustración 1 – Superposición de procesos de llegadas creados por hilos de JMeter

Construiremos a continuación un proceso de llegadas que se asemeje a un proceso de Poisson siguiendo el procedimiento descrito. Para ello comiencen un plan de pruebas de JMeter con un grupo de hilos. Incluyan en ese grupo de hilos un muestreador de petición HTTP y un timer de espera aleatoria uniforme. De esta forma, todos los $T_i^{(k)}$ de la Ilustración 1 serán variables aleatorias uniformes independientes e idénticamente distribuidas (i.i.d.). Haremos referencia a una cualquiera de estas variables aleatorias con T, siendo su media E[T]. El elemento de petición HTTP solicitará el recurso *randwait.php* indicando un tiempo de espera uniforme. Así, todos los $W_i^{(k)}$ de la Ilustración 1 serán variables aleatorias uniformes independientes e idénticamente distribuidas. Haremos referencia a una cualquiera de estas variables aleatorias con W, siendo su media E[W]. Llamaremos N_h al número de hilos.

[1] No confundir cuando hablamos de "proceso" en el sentido de proceso estocástico de llegadas por ejemplo de cuando nos referimos a un proceso como programa en ejecución con su memoria, pila, etc

Dada esta forma de crear el proceso, ¿cuál es el tiempo medio entre dos peticiones consecutivas del mismo hilo al servidor web? Llamaremos a este tiempo $E[Ti_h]$.

¿Cuál será la tasa media de peticiones por unidad de tiempo que hace un hilo cualquiera al servidor web? La llamaremos λ_h

¿Cuál será la tasa media total de peticiones que recibe el servidor web? La llamaremos λ_T

Dado que el tiempo de servicio que sufren las peticiones viene determinado por la variable aleatoria W, ¿cuál es la intensidad de tráfico que ataca al servidor? La llamaremos I

3. Sistema simplificado

Hemos descrito la forma de crear un proceso aproximadamente poissoniano. En este proceso de creación especificamos un tiempo de servicio uniforme para el trabajo del servidor. El procedimiento no requería que dicho tiempo fuera uniforme sino que podría seguir cualquier distribución con momentos finitos. Lo mismo aplica al tiempo de OFF.

A partir de ahora usaremos para W una variable aleatoria exponencial. Eso implica que tenemos no solo un proceso de llegadas de Poisson sino también tiempos de servicio exponenciales i.i.d. Para el servidor web mantendremos la configuración presentada en partes anteriores de este trabajo, de forma que como mucho se atenderá una petición a la vez y configuraremos un ListenBacklog de 1 .

¿En notación de Kendall a qué tipo de sistema se ha reducido el problema? Verifique este comportamiento con experimentos con JMeter y wireshark donde vea cuántas peticiones se quedan encoladas.

4. Desviaciones del modelo

Una vez que el kernel del servidor acepta la conexión TCP, cuando el proceso de apache esté libre tomará dicha conexión para atenderla. Desde ese punto el proceso estará ocupado y no podrá atender otras peticiones hasta que finalice esta tarea. El proceso debe cargar el script de PHP y ejecutar sus instrucciones, entre las que se encuentra la espera aleatoria según la distribución indicada. Es decir, el resultado final será que el proceso del servidor estará más tiempo ocupado que el tiempo de espera aleatoria, dado que a ese tiempo se añade el de ejecutar el código del script anterior y posterior a la pausa.

Hay que tener en cuenta también que estamos ejecutando Apache en una máquina virtual. Esto implica que es posible que el procesador físico tenga que hacer cambios de contexto entre el host real (Ubuntu) y la máquina virtual (FreeBSD), lo cual conlleva que el tiempo que esté ocupado el proceso de apache sea aún mayor. Todo este aumento de tiempos será en general pequeño, pero si el valor de tiempo medio de espera que indicamos es también pequeño eso quiere decir que ese aumento de tiempos estará muchas veces en el mismo rango que el tiempo de espera, alterándolo en un porcentaje apreciable y con él la carga que está sufriendo el sistema frente a la que creemos. Así pues es conveniente que los tiempos de espera sean superiores a estos añadidos para que su efecto sea despreciable. Sin embargo, aumentar los tiempos de espera aumentará el tiempo que se tardará en llevar a cabo el experimento con un gran número de llegadas, por lo que tampoco querremos hacerlo muy grande.

5. Experimentación

El objetivo es evaluar cómo depende el porcentaje de peticiones que no se sirven inmediatamente, de parámetros como la carga que sufre el sistema, el número de procesos que lanza el servidor, etc. En primer lugar nos planteamos evaluar simplemente la dependencia de dicho porcentaje con la intensidad de tráfico que ataca al servidor. El resultado podría ser una

gráfica que nos muestre en el eje *x* la intensidad de tráfico y en el eje *y* el porcentaje (probabilidad) de pérdidas (ejemplo en la Ilustración 2).

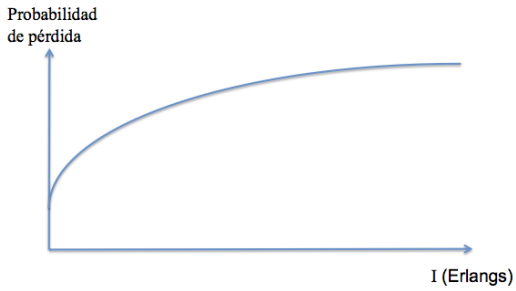

Ilustración 2 - Ejemplo de resultado

Cada punto en esta gráfica corresponderá a un experimento en el que habremos configurado unos parámetros en JMeter para que haga peticiones al servidor con la carga indicada y obtenido la probabilidad de pérdida medida como el porcentaje de peticiones que no fueron aceptadas por el servidor. Así pues, lo primero que necesitamos es decidir/calcular los valores de los parámetros de JMeter para crear una intensidad de tráfico en concreto. Esto va a ser tan simple como el proceso inverso del que se ha pedido hacer en el apartado 2 de este documento.

El número de hilos se ha planteado como determinado por el número de usuarios que queremos emular. Querríamos probar el funcionamiento del sistema ante un gran número de usuarios, sin embargo, no es factible emplear JMeter con cifras de miles de usuarios/hilos y alta carga. Así pues, en vez de centrarnos en que los hilos emulen cada uno el comportamiento de un usuario nos centraremos en que la superposición de los hilos emule el multiplex de un número grande de los mismos. Para esto necesitamos que N sea lo suficientemente grande para que el proceso resultante "parezca" de Poisson, al menos a los efectos del rango de parámetros del sistema a evaluar. No vamos a dedicar tiempo en este trabajo a decidir un valor o valores para N sino que tomaremos directamente en torno a N=30 hilos.

Así pues, para cada intensidad de carga con la que desee experimentar, calcule unos valores razonables de E[W] y E[T]. Empleando en JMeter un Timer uniforme entre 0 y un valor T_{max} es trivial decidir dicho valor T_{max} para que la media sea E[T].

Queda por decidir el número de peticiones que hará cada hilo. Cuantas menos peticiones pongamos en el experimento menos tiempo tardará en llevarse a cabo, sin embargo, dado que estamos calculando el porcentaje de peticiones no atendidas necesitamos hacer suficientes peticiones para que el cálculo de dicho porcentaje tenga un grado de confianza suficiente. Por ejemplo, resulta evidente que si tenemos 30 hilos y a cada uno le pedimos que haga 10 peticiones tenemos un total de 300 peticiones con las que es difícil por ejemplo estimar correctamente porcentajes de pérdidas del orden de 1%. Lo que vamos a hacer es lanzar experimentos de muy larga duración y observar cómo a medida que se producen más llegadas se va estabilizando la estimación de la probabilidad de pérdida calculada con el porcentaje de pérdidas. Para ello en el plan de pruebas indicaremos que se guarde en fichero un log de las peticiones[1]. Se ha preparado un script[2] para visualizar la evolución del experimento. Este script requiere que se seleccione en la configuración del registro en fichero la opción "Guardar código de respuesta" pues la necesita para reconocer en dicho fichero las peticiones que fueron rechazadas (de forma muy simple, básicamente buscando "timed out" para reconocer si

[1] No añada al plan de pruebas un *Listener* que por ejemplo guarde en memoria del programa todas las peticiones y sus respuestas pues en un experimento largo se distorsionarán las medidas por el tiempo que consumirá JMeter en mantener dicha lista en memoria.

[2] Se les deja este script como ayuda pero no es obligatorio usarlo; pueden crear sus propias herramientas para procesar los ficheros de log o usar otra como *awk*, una hoja de cálculo, etc. Es más recomendable no copiarlo y utilizar siempre la versión que haya en el directorio /opt3/varios/compartir/rss

la línea es de una petición que se atendió o que no). El fichero también debe estar en formato CSV (no en XML). El script toma el fichero de registro y calcula la probabilidad de pérdida con las K primeras peticiones, las K+1, las K+2, etc, graficando esa probabilidad frente a K. Si se cierra la figura, el script toma de nuevo el fichero (que habrá crecido entre tanto) y rehace la figura.

El script puede ejecutarlo de la siguiente forma:

```
$ /opt3/varios/compartir/rss/graphprogreso FICHERO_DE_LOG
```

Y puede interrumpirlo en el terminal con Ctrl+C[3].

Calcule los parámetros necesarios para llevar a cabo los experimentos que le permitan crear una figura del estilo de la Ilustración 2.

Ensaye dichos parámetros con algunas intensidades de tráfico para verificar que los resultados se estabilizan en un tiempo razonable. Tenga en cuenta las limitaciones mencionadas en el apartado 4. Pruebe a verificar que la carga que está sufriendo el sistema es la calculada teóricamente.

6. Resultados

En el anterior apartado se han obtenido los parámetros para realizar un estudio del comportamiento del Apache con la configuración establecida en la sesión inicial. Se ha supuesto que esta configuración con estos parámetros de tráfico que lo atacan se puede comparar a un modelo de teoría de colas de los estudiados teóricamente, pero habrá que comprobarlo.

Si en el fichero de log que guarda los resultados activa también “Guardar etiqueta de tiempo”, el JMeter guarda el timestamp de cada petición con precisión de ms. Lo que necesita es el instante que se genera la petición, no el instante que se finaliza la petición que es lo que guarda por defecto el JMeter. Ya buscó cómo cambiar este comportamiento al final de “Parte 2 - Introducción a Apache JMeter” de esta práctica lanzando el JMeter con ciertos parámetros.

Lance el JMeter tal como se pide y guarde también el timestamp inicial de las peticiones. Realice los experimentos planteados en el apartado anterior y obtenga las probabilidades de pérdidas para cada intensidad de tráfico. Compare los resultados con los teóricos esperados.

Con los timestamps de las peticiones haga los histogramas del tiempo entre peticiones y compruebe si siguen una distribución exponencial. Realice también las autocorrelaciones del tiempo entre peticiones para comprobar si las peticiones se pueden considerar un proceso independiente.

7. Aumentando la cola

Hasta ahora el Apache estaba configurado para guardar una petición en cola, vamos a probar a aumentar este valor.

Aumente el valor de la cola hasta 4. ¿En notación de Kendall, ahora a qué tipo de sistema se ha reducido el problema? Realice otra vez los mismo experimentos y saque los mismos resultados que en el apartado anterior: probabilidades de pérdidas, histogramas y autocorrelaciones del tiempo entre peticiones.

Si tiene tiempo pruebe con otros tamaños de cola.

[3] El script crea ficheros temporales en /tmp que intenta borrar pero puede que se queden si aborta el script (algo que al final tendrá que hacer pues no hay una forma más elegante de terminarlo). No se preocupe por los ficheros temporales pues la próxima vez que ejecute el script borrará los anteriores que se quedaron.