

Una Introducción a las Pruebas de Prestaciones

(Versión 1.2)

Puedes descargar la última versión de este documento de:

http://blog.unlugarenelmundo.es/?page_id=127

José María Morales Vázquez

josemaria@morales-vazquez.com

Resumen: Es muy común en las grandes y medianas empresas que existan equipos de trabajo dedicados casi en exclusiva a probar la funcionalidad de sus nuevos desarrollos mientras que se dedique poco o ningún tiempo a comprobar si estos cumplen con unos requisitos mínimos de rendimiento. Las pruebas de prestaciones, enmarcadas dentro de lo que se viene a llamar Calidad Operacional o Calidad de Servicio son, hoy en día, cada vez más necesarias: los tiempos de respuesta por encima de lo aceptable, la excesiva variabilidad de los mismos en función de la carga del sistema y los problemas de fiabilidad o disponibilidad deben de considerarse errores tan graves como los de funcionalidad. Los problemas de rendimiento son provocados por causas que pueden clasificarse en dos categorías: predecibles e impredecibles. Una correcta metodología de pruebas de prestaciones debería de ser capaz de detectar y posibilitar la corrección anticipada de problemas pertenecientes a ambas categorías. Con esta breve introducción se pretende exponer los principales conceptos a tener en cuenta a la hora de acometer las pruebas de prestaciones de cualquier sistema, haciendo especial hincapié en las particularidades de los sistemas Web.



Este documento se encuentra bajo una Licencia [Creative Commons Atribución-CompartirIgual 3.0 Unported](https://creativecommons.org/licenses/by-nc/3.0/).

Índice

1. Introducción.....	3
1.1. Necesidad y Beneficios.....	3
1.2. Limitaciones.....	3
1.3. Objetivos.....	4
1.4. Ciclo de vida.....	4
2. Diseño del Entorno de pruebas.....	6
2.1. Software.....	6
2.2. Hardware de los generadores de carga.....	8
2.3. Network.....	8
2.4. Servidores.....	9
3. Metodología.....	10
3.1. Planificación.....	10
3.1.1. Requisitos.....	10
3.1.2. Métricas.....	11
3.1.3. Tipos de Pruebas.....	13
3.1.4. Descripción del sistema.....	15
3.1.5. Caracterización de la carga de trabajo.....	15
3.2. Construcción.....	16
3.2.1. Diseño de la carga de trabajo.....	16
3.2.2. Adecuación de la configuración.....	19
3.3. Ejecución.....	20
3.3.1. Monitorización.....	20
3.3.2. Validación de los resultados.....	21
3.3.3. Margen de error.....	22
3.4. Análisis e interpretación de resultados.....	22
3.4.1. Formas de mejorar el rendimiento.....	22
3.4.2. Ejemplos gráficos.....	23
4. Otros conceptos relacionados.....	26
4.1. Escalabilidad.....	26
4.2. Fiabilidad y Disponibilidad.....	26
5. Conclusiones.....	27
6. Bibliografía.....	28

1. Introducción.

Predecir como va a comportarse un sistema ante una carga específica es una tarea ardua y complicada, en particular si estamos hablando de un sistema *Web*. Los sistemas *Web* actuales son extremadamente complejos, con una gran cantidad de componentes hardware y software de diferentes fabricantes, infinidad de tecnologías implicadas en su funcionamiento y, en la mayoría de los casos, una total indeterminación debido al carácter impredecible de los hábitos de sus usuarios. La única forma efectiva de anticipar el comportamiento de un sistema de estas características es realizando pruebas de prestaciones.

Con el presente documento pretendemos realizar un breve repaso por los conceptos y términos relacionados con esta disciplina, intentaremos justificar la casi absoluta necesidad de utilizar una herramienta específica para auxiliarnos en la preparación de las mismas y presentaremos una metodología adecuada para su acometida.

1.1. Necesidad y Beneficios.

La principal necesidad de realizar unas pruebas de prestaciones es obvia y más aún en los sistemas *Web*: habitualmente los principales problemas que nos encontramos en nuestra utilización diaria de este tipo de aplicaciones no son ni caídas del sistema ni anomalías en su funcionamiento, sino problemas de rendimiento y degradación de recursos.

La finalidad de realizar pruebas de prestaciones es simular la normal utilización del sistema antes de su paso a explotación para predecir anticipadamente este tipo de situaciones y facilitar su corrección.

1.2. Limitaciones.

Como iremos viendo a lo largo de este documento, realizar unas correctas pruebas de prestaciones que nos devuelvan unos resultados veraces y útiles no es una tarea sencilla. Existen demasiados problemas involucrados algunos de los cuales son casi totalmente irresolubles y que sólo pueden ser aproximados: Rara vez podremos realizar pruebas sobre un entorno idéntico al de producción, el juego de datos del que disponemos para las pruebas será insuficiente, no podremos nunca simular el comportamiento exacto de un usuario real y siempre manejaremos un cierto margen de error en todos nuestros resultados.

Las pruebas de prestaciones no arrojan nunca resultados absolutos. Devuelven valores aproximados y con un cierto margen de error. Minimizar ese error es nuestra tarea y para ello se hace imprescindible un amplio conocimiento de todos los conceptos relacionados en esta tarea y seguir una metodología clara y precisa.

1.3. Objetivos.

En un sentido amplio, las pruebas de prestaciones tienen un doble objetivo:

- ❑ Verificar que la capacidad del sistema es adecuada para la demanda de trabajo que soportará.
- ❑ Detectar posibles cuellos de botella e ineficiencias proporcionando la información necesaria para un correcto dimensionado.

Idealmente, a la hora de enfrentarnos a las pruebas de prestaciones de un sistema concreto deberíamos de afinar un poco más cuales son los objetivos que perseguimos. La siguiente lista muestra algunos ejemplos de esto:

- ❑ Ajustar nuestro sistema para soportar la máxima carga de trabajo posible usando la infraestructura actual.
- ❑ Asegurarnos de que, ante una carga de trabajo determinada, el 95% de las páginas se descargan en menos de 10 segundos cuando no están cifradas y en menos de 15 cuando si lo están.
- ❑ Determinar el tiempo medio de respuesta que obtendrá el usuario.
- ❑ Determinar el máximo número de sesiones concurrentes, inicios de sesión por hora o transacciones por segundo que nuestro sistema es capaz de soportar mientras proporciona un nivel de rendimiento aceptable.
- ❑ Determinar el ancho de banda necesario que nuestro sistema necesita cuando procesa el máximo número de transacciones por segundo que puede manejar.
- ❑ Determinar como reacciona nuestro sistema ante cargas de trabajo que exceden de su capacidad.
- ❑ Identificar las transacciones más lentas y las más rápidas.
- ❑ Identificar las transacciones con una mayor desviación típica en sus tiempos de respuesta.

1.4. Ciclo de vida.

Idealmente, las pruebas de prestaciones deberían de comenzar en la fase más temprana posible del ciclo de vida de nuestro sistema. Esto nos reduce el porcentaje de posibles problemas de prestaciones que podemos arrastrar hasta fases posteriores y, por supuesto, nos reduce el coste económico de su corrección.

Imaginemos que conducimos un proyecto en fase de análisis de requisitos e identificamos la necesidad de adquirir un software de control de acceso a nuestras páginas webs. Tenemos tres posibles candidatos y tenemos la posibilidad de evaluar gratuitamente su funcionamiento. Si realizamos pruebas de prestaciones sobre una maqueta diseñada a tal efecto tendremos menos posibilidades de encontrarnos en fases posteriores que el software de acceso elegido representa un cuello de botella en cuanto al acceso a nuestro sistema se refiere.

No olvidemos tampoco que las pruebas de prestaciones no se limitan exclusivamente a las respuesta de la aplicación. Estamos probando el sistema completo y anteriormente hemos debido de probar la infraestructura de nuestro sistema a bajo nivel. A tal efecto, existen en el mercado infinidad de herramientas de entre las que merece la pena destacar las dos siguientes:

- ❑ *Webstone* de Mindcraft www.mindcraft.com
- ❑ *WebBench* de Zdnet www.zdnet.com

En cualquier caso, las pruebas de prestaciones más importantes lógicamente serán las que nos permitan una validación final de nuestro sistema. Estas deben de ejecutarse idealmente sobre un entorno estable, lo más similar posible al entorno final de producción y cuando la aplicación haya superado correctamente las pruebas funcionales y se haya comprobado que la respuesta del sistemas es correcta en todo momento.

2. Diseño del Entorno de pruebas.

Diseñar y dimensionar correctamente el entorno necesario para las pruebas es un paso previo a la realización de las mismas. Por entorno de pruebas entendemos el conjunto de herramientas y máquinas mediante las cuales generaremos la carga con la que atacaremos el sistema, el conjunto de servidores sobre los que efectuaremos las pruebas y que comúnmente no serán los mismos sobre los cuales se realizará la explotación y la infraestructura de red que los une a ambos. Estudiaremos este conjunto subdividiéndolo en cuatro apartados:

- Software para generar la carga
- Hardware de los generadores de carga
- Network
- Servidores

2.1. Software para generar la carga.

Existen diferentes formas de generar la carga de trabajo que necesitamos para realizar nuestras pruebas. La más fácil de implementar es, por supuesto, la manual pero también es la más ineficiente la menos flexible y posiblemente la más cara para grandes tests. Imaginemos que para realizar pruebas con 100 usuarios necesitamos un laboratorio con espacio suficiente para albergarlos a ellos y a los ordenadores desde los que generaran la carga más toda la infraestructura necesaria de software y hardware precisa, sin olvidarnos del coste de los operarios en si.

Un segundo método consistiría en desarrollar generadores de carga a la medida de nuestras necesidades. Aunque a simple vista nos puede parecer que esto nos reporta un considerable ahorro de dinero, dado el elevado precio de las herramientas comerciales, tenemos que considerar que las herramientas de generación de carga son enormemente complejas y su desarrollo sería muy costoso en tiempo y dinero salvo que nuestras necesidades se limiten a ciertos requisitos extremadamente rígidos y concretos.

La solución más comúnmente adoptada sería comprar una herramienta de generación de carga y utilizarla desde nuestro propio entorno de pruebas de prestaciones construido a tal efecto. Si bien es verdad que, como decíamos antes, estas herramientas poseen un precio elevado, son flexibles y se actualizan periódicamente con nuevas funcionalidades y protocolos. A la hora de elegir la herramienta adecuada a nuestras necesidades, sin duda la característica más importante que debemos de buscar es la flexibilidad en su configuración. Como veremos más adelante en los apartados dedicados al diseño de la carga de trabajo

existen infinidad de variables para caracterizar adecuadamente la forma con la que atacaremos el sistema en prueba. Nunca podremos llegar a un nivel de modelado que sea igual a la carga real, pero cuanto más flexibilidad nos aporte la herramienta escogida, más cerca estaremos de conseguirlo.

Las cinco herramientas más utilizadas son:

- ❑ *SilkPerformer* de Segue Software. www.segue.com
- ❑ *LoadRunner* de Mercury Interactive www.merc-int.com
- ❑ *QALoad* de Compuware www.compuware.com
- ❑ *SiteLoad* de Rational www.rational.com
- ❑ *WebLoad* de Radview www.radview.com

De forma más o menos integradas, todas ellas constan de cuatro componentes principales: el generador de carga virtual, el simulador de clientes virtuales, el controlador de la emulación y el analizador de resultados. Los precios varían en función del número de usuarios virtuales que pretendemos generar. Últimamente, algunas de estas compañías han adoptado una política distinta para ahorrar costes: permiten comprar licencias temporales de forma que compramos inicialmente el generador de carga, el controlador y el analizador y ‘alquilamos’ los generadores de clientes virtuales exclusivamente por el tiempo que los necesitemos.

Una cuarta opción sería comprar una de estas herramientas para diseñar la carga pero confiar en una empresa externa para la generación de la misma. Normalmente, las mismas empresas que fabrican y venden las herramientas proporcionan estos servicios. Esta estrategia nos permite ahorrar la mayor parte del coste de infraestructura hardware y el de los generadores de usuarios virtuales.

Una última estrategia consistiría en alquilar a estas empresas de servicios tanto la herramienta de creación de la carga como la generación de la misma. Esta estrategia reduce enormemente la inversión inicial. Tanto en este caso como en el anterior, la carga es generada desde las dependencias de la empresa a la que contratamos los servicios. Esto puede hacer mucho más real la simulación de la carga, pero también nos exige realizar cambios en nuestra política de seguridad para permitir que los servidores atacados por las pruebas sean accesibles desde Internet.

Quizás la estrategia más efectiva sería realizar una combinación de todas ellas: Las pruebas iniciales deberían de hacerse manualmente por una o dos personas. En una segunda fase realizaríamos pruebas a mediana escala con entre 5 y 100 usuarios virtuales mediante una herramienta comprada a tal efecto. Si es necesario realizar pruebas a mayor escala (entre 100 y 10.000 usuarios virtuales) contrataríamos la generación de la carga a una empresa de servicios.

2.2. Hardware de los generadores de carga.

La mayoría de estas herramientas funcionan exclusivamente bajos Windows NT 4.0 / Windows 2000, aunque algunas de ellas tenga la opción de utilizar agentes UNIX para simular los usuarios virtuales. El principal recurso que tenemos que vigilar en nuestras máquinas es la memoria física que es la que nos limitará el número de usuarios virtuales que podemos generar. El Sistema operativo y el núcleo de la herramienta ocupan entre 32 y 64 Megas de RAM. La generación de cada usuario virtual ocupa una cantidad de memoria que viene determinada por la herramienta en si, el tamaño del script de pruebas y por el protocolo que usamos en la emulación de carga y oscila entre algo menos de 1 Mbyte hasta 6 u 8 Mbytes por usuario.

Por ejemplo, en una máquina con 256 Mbytes de memoria física, usando SilkPerformer 4.5 y emulando usuarios que usan tráfico HTTP (aproximadamente 1 Mbyte por usuario) podemos generar hasta 192 usuarios virtuales. Si necesitamos generar hasta 1000 usuarios virtuales necesitaríamos seis máquinas de estas características.

Lógicamente, es posible utilizar la memoria virtual de la máquina para la generación de los usuarios virtuales, pero el tiempo empleado en realizar el swapping a disco de la memoria ocupada por los mismos haría disminuir visiblemente la carga generada y desvirtuaría sensiblemente el resultado de las pruebas. El swapping a disco no es deseable en general durante el desarrollo de las pruebas y no es aceptable una utilización de memoria total del equipo durante las pruebas por encima del 150% de la memoria física

Evidentemente, existen otros condicionantes a tener en cuenta a la hora de dimensionar nuestra infraestructura hardware: La CPU consumida, por ejemplo, variará ostensiblemente en función de que el tráfico generado precise de cifrado o no. Es deseable, en general, una utilización de CPU por debajo del 70% y consideraremos no aceptable una utilización por encima del 90%.

Por otro lado, las herramientas de prueba y monitorización hacen un uso intensivo de los discos para escribir los datos recogidos durante las pruebas. Se hacen, por tanto, totalmente necesarios grandes volúmenes de disco en las máquinas de prueba que han de ser convenientemente mantenidos y saneados para que la carencia de espacio no provoque un error durante las pruebas.

2.3. Network.

La infraestructura de nuestra red también debe de soportar el tráfico generado durante la simulación para no desvirtuar el resultado de las pruebas. Los anchos de banda típicamente usados en redes corporativas son:

- ❑ Ethernet 10BASE-T 10 Mbps
- ❑ Ethernet 100BASE-T 100 Mbps
- ❑ Gigabit Ethernet 1 Gbps

El ancho de banda real disponible depende fundamentalmente, además del ancho de banda teórico antes enumerado, del número de ordenadores conectados a la red, del tráfico de los mismos y del número de conmutadores y su tipo (hubs y/o switches), y suele oscilar entre un 10 y un 35 % del ancho de banda teórico.

Imaginemos una simulación realizada en una red Ethernet a 10 Mbps con un ancho de banda disponible aproximado del 15%, es decir, de 1,5 Mbps. Una simulación que implicase a 10 máquinas generando una simulación de carga permitiría a cada una de ellas un tráfico aproximado de 150 Kbps. Si el tráfico generado excede de esa cifra estamos introduciendo un retraso que desvirtuaría el desarrollo de las pruebas.

2.4. Servidores

Tan sólo en contadas ocasiones (nuevas implantaciones, sistemas que sólo funcionan en determinados horarios, etc.) es posible realizar las pruebas de prestaciones directamente sobre el entorno real de producción o explotación. En estas ocasiones, los resultados obtenidos durante las pruebas serán de aplicación casi directa puesto que no es necesario un gran esfuerzo adicional para interpretar los resultados que se obtengan.

Los servidores utilizados en el entorno de pruebas y, sobre todo, la arquitectura del mismo han de ser lo más parecidos posible a los servidores y la arquitectura utilizados finalmente en explotación. Cuanto más similares sean ambos entornos mayor será la exactitud de los resultados obtenidos.

3. Metodología.

La Metodología a seguir para la realización de unas pruebas de prestaciones adecuadas debe de comprender al menos cuatro fases imprescindibles:

- Planificación
- Construcción
- Ejecución
- Análisis

En los siguientes apartados diseccionaremos y analizaremos en detalle cada una de estas fases.

3.1. Planificación.

El corazón de una correcta planificación de las pruebas de prestaciones y una de las partes fundamentales de las mismas es la correcta caracterización de la carga de trabajo. En este apartado veremos, además de ese punto, las cuatro fases imprescindibles que debemos de abordar durante esta fase de planificación, a saber:

- Obtención de requisitos.
- Elección de las métricas adecuadas.
- Elección del tipo de pruebas.
- Descripción del sistema a probar.
- Caracterización de la carga de trabajo.

Durante la fase de planificación de las pruebas es totalmente imprescindible la participación de personal que represente a las siguientes áreas:

- Desarrollo**, que aporte el conocimiento necesario para comprender la arquitectura del sistema.
- Usuario final**, que aporta el patrón de utilización que tendrá el sistema bajo pruebas.

3.1.1. Requisitos.

Cuando una aplicación presenta una desviación con respecto a la funcionalidad que debería de tener decimos que tiene un error y aquí no hay vuelta de hoja. La funcionalidad que debe de tener la aplicación debe de estar correctamente detallada en su pliego de requisitos y dicho documento debe de estar en poder del personal del departamento de certificación responsable de realizar las pruebas funcionales.

Para certificar que las prestaciones de un sistema son correctas necesitamos igualmente un pliego de requisitos. Un sistema que no soporta más de 500 usuarios concurrentes con unos tiempos de respuesta aceptables puede (y debe) certificarse como correcta si los usuarios potenciales nunca llegaran a 200.

No obstante, como vimos en la lista de objetivos del punto de vista 1.3 de este documento, también puede ocurrir que el objeto de unas determinadas pruebas no sea exactamente validar la calidad de servicio proporcionada a los usuarios de acuerdo a unos determinados rangos de valores, sino otras muy distintas como, por ejemplo, determinar como se comporta el sistema ante cargas que superan ampliamente su capacidad. En este caso el documento de requisitos debería de especificar como deseamos que se comporte el sistema ante esta situación para utilizar este dato en la validación del sistema.

3.1.2. Métricas.

Las pruebas de prestaciones nos exigen buscar resultados enmarcados en dos grupos bien diferenciados:

- ❑ Desde el punto de vista del cliente
- ❑ Desde el punto de vista del servidor

Asimismo, tenemos un tercer grupo de métricas que abordar: las que nos definirán la carga que aplicamos al sistema. Este tercer grupo, debido a sus peculiares características, es el más difícil de definir.

Desde el punto de vista del cliente lo más importante son, lógicamente, los tiempos de respuesta. El valor medio de los tiempos de respuesta percibidos por el usuario es una buena medida de la calidad operacional de nuestro sistema. Una categorización de los tiempos de respuesta aceptables podría ser la siguiente:

- ❑ **Menos de 0,1 seg.** Tiempo de respuesta admisible para pequeña acciones que no precisan de confirmación, como por ejemplo marcar la casilla de un formulario o pulsar un botón.
 - ❑ **Menos de 1 seg.** Tiempo admisible para que el usuario no pierda la 'ilusión' de estar trabajando en continuo y en exclusiva contra una fuente de datos.
 - ❑ **Menos de 10 seg.** Tiempo invertido en descargar completamente una página sin cifrado de información.
 - ❑ **Menos de 15 seg.** Tiempo invertido en descargar completamente una página bajo un protocolo de cifrado seguro.
-

El tiempo de respuesta percibido por el usuario depende fundamentalmente de seis variables:

- ❑ Tamaño de la página a descargar.
- ❑ Ancho de banda disponible
- ❑ Número de conexiones TCP requeridas para descargar la página.
- ❑ *Round-Trip*.
- ❑ Tiempo de proceso por parte del servidor.
- ❑ Tiempo de proceso por parte del cliente.

En http 1.0, el mínimo de conexiones TCP requeridas para descargar una página es de cuatro: una para interrogar al servidor DNS y otras tres para bajar el cuerpo de la página. Por cada objeto adicional que esta contenga son necesarias otras tres conexiones. La mayoría de los navegadores están configurados para soportar por defecto cuatro conexiones http concurrentes, por lo que la descarga de una página que sólo contenga texto se puede realizar de una sola vez.

El *Round-Trip* o viaje de ida y vuelta (RTT en adelante) indica la latencia o retraso entre el envío de una petición desde el navegador del cliente hasta el servidor web y la recepción, por parte del cliente, de los primeros bytes de la respuesta. La importancia del RTT radica en que cualquier comunicación entre cliente y servidor, por trivial que sea, tiene que pagar este mínimo 'impuesto'. La forma más trivial de medir el RTT es usando un simple 'ping'

Desde el punto de vista del servidor, las medidas a usar son mucho más claras: %CPU, número de procesos en cola de ejecución, cantidad de memoria libre, actividad del sistema de *swapping*, utilización de los discos y en definitiva, cualquier parámetro que nos ayude a determinar el consumo de recursos y los posibles cuellos de botella de nuestro sistema. Aunque las principales herramientas de simulación de carga poseen monitores para obtener los principales parámetros que pueden preocuparnos, es mucho más fiable y flexible obtener directamente estas mediciones a través de los comandos y las estadísticas que nos proporciona el propio sistema operativo.

Nuestro principal problema radica en elegir una medida efectiva de la carga aplicada al sistema durante las pruebas. Veamos las distintas alternativas que poseemos y las ventajas e inconvenientes de cada una de ellas:

- ❑ **Número de usuarios.** Es la medida más efectiva que poseemos de la cantidad de carga aplicada, aunque no es muy concreta si no la caracterizamos completamente: existen demasiados parámetros involucrados en los mismos y, como todos sabemos, 200 usuarios normales pueden generar la misma carga de trabajo que 20 usuarios que
-

hacen un uso intensivo del sistema (de esos que pulsán el siguiente enlace antes de que la página esté totalmente cargada).

- ❑ **Hits por segundo.** Aunque erróneamente se utiliza el término *Hit* para designar el número de visitas que recibe una página web, un hit representa en realidad la descarga de un fichero desde un servidor web hasta un cliente. La descarga de una página web implica al menos un hit para bajar el fichero de texto html (un solo hit a pesar de que la página esté compuesta por distintos *frames*) y un hit adicional por cada fichero incrustado que exista en la página: fotos, imágenes, archivos de sonido, etc. Realmente no es una medida fiable (descargar una foto de 256 kbytes representa un hit, al igual que hacer lo propio con un hit de tan solo 7 kbytes) que, además, nos da más una idea del rendimiento del servidor, que de la carga aplicada al mismo, puesto que cuando el servidor se satura podemos observar que a pesar de que incrementemos la carga aplicada al mismo el número de *hits* por segundo tiende a estabilizarse o incluso a descender.
- ❑ **Transacciones o Ciclos de Trabajo.** Una tercera posibilidad es definir una transacción o ciclo de trabajo característico y propio del sistema a probar y tomarlo como unidad. En este caso tenemos dos problemas : al igual que ocurre con los *hits* por segundo, un servidor saturado tenderá a estabilizar el número de ciclos ejecutados a pesar de que aumentemos la presión de trabajo sobre el sistema. Adicionalmente, estamos estableciendo un sistema de medida tan propio y característico del sistema a probar que nos imposibilita el realizar comparaciones entre las prestaciones de dos sistemas diferentes.

Aunque existe alguna otra opción marginal, desgraciadamente no parece existir ninguna solución perfecta para este problema. La mejor forma parece ser el uso combinado de dos de estas medidas: el número de usuarios trabajando contra el sistema nos da una buena idea de la carga aplicada al sistema en cada momento y de como esta se incrementa, mientras que una caracterización de la misma mediante ciclos de trabajo correctamente definidos nos permite conocer el volumen de la misma.

3.1.3. Tipos de Pruebas.

Las pruebas de prestaciones nos permiten observar y evaluar las respuesta de nuestro sistema ante todas las posibles situaciones que puedan presentarse. Para ajustar los resultados obtenidos a nuestros objetivos y necesidades tenemos que elegir correctamente el tipo de pruebas que queremos realizar.

Existe entre los diferentes autores gran disparidad en los nombres aplicados a los distintos tipos de pruebas que podemos realizar Hemos elegido en nuestra

clasificación los nombres más comunes, tanto en inglés como en castellano, con que se conocen los distintos tipos de pruebas.

Una primera clasificación nos presenta las cuatro categorías más populares:

- ❑ **Carga (*Load*)**. Estas pruebas se usan para modelar una carga lo más parecida posible a la real y medir anticipadamente el rendimiento que proporcionará nuestro sistema. Se ejecutan durante cortos periodos de tiempo (típicamente inferiores a una hora) y sirven básicamente para verificar si se cumplen los requisitos de prestaciones preestablecidos. En definitiva, permiten identificar los tiempos de respuesta que obtendrá un usuario típico frente a condiciones normales.
- ❑ **Capacidad (*Capacity*)**. Con ellas se pretende someter al sistema a una actividad creciente de forma incremental hasta detectar el punto de saturación del mismo. Son las más populares puesto que, bien diseñadas, nos permiten observar el comportamiento de nuestro sistema en una amplia gama de situaciones.
- ❑ **Estrés (*Stress*)**. Estas pruebas se diseñan con el objeto de analizar el efecto que produce en nuestro sistema la aplicación continuada de una carga por encima de su capacidad. Nuestro principal objetivo debe de ser verificar que el sistema no pierde su integridad en ningún momento.
- ❑ **Estabilidad (*Stability*)**. Estas pruebas consisten en diseñar una carga lo más similar posible a la real y atacar con ella al sistema durante largos períodos de tiempo (típicamente entre 24 horas y una semana) para buscar posibles deterioros o degradaciones en las prestaciones del sistema. Una ligera variante de estas pruebas analizaría asimismo que al disminuir la carga aplicada los recursos de nuestro sistema se liberan correctamente.

Otras categorías de pruebas de prestaciones más específicas y menos comunes son las siguientes:

- ❑ **De humo (*Smoke*)**. Son pruebas preliminares que se usan para determinar si el sistema está ya listo para probar su rendimiento en mayor profundidad. Suelen ser tan sencillas como una simple comprobación manual de que la funcionalidad del sistema es correcta.
 - ❑ **De Aislamiento (*Isolation*)**. Se usan cuando tenemos identificado un problema en nuestro sistema y diseñamos una prueba con un conjunto específico de transacciones que lo provocan. Lógicamente son pruebas artificiales que no buscan reproducir una carga real.
 - ❑ **Sobrecarga (*Overload*)**. Un sistema puede ser capaz de manejar una determinada carga que se va aplicando gradualmente, pero esto no nos garantiza que sea capaz de soportar esa misma carga si esta se le aplica súbitamente durante un corto periodo de tiempo. Las pruebas de sobrecarga reproducen fuertes picos de carga durante cortos periodos de
-

tiempo con el objeto de estudiar el comportamiento de nuestro sistema ante estas situaciones.

- **Elasticidad (*Elasticity*)**. Una variante de las pruebas de sobrecarga son las de elasticidad que consisten en aplicar fuertes y breves picos de carga separados por breves espacios de baja actividad para estudiar la forma en que el sistema es capaz de reclamar los recursos que necesita y la velocidad con la que es capaz de liberarlos.

3.1.4. Descripción del sistema.

Pese a lo que pueda parecer, no siempre es imprescindible conocer la arquitectura del sistema en pruebas para poder realizar un diagnóstico de sus prestaciones y la calidad de las mismas. De forma similar a como se realizan las pruebas de caja negra, podemos realizar pruebas de prestaciones de nuestros sistemas atendiendo solo y exclusivamente al punto de vista del usuario y a los tiempos de respuesta que este va a recibir.

En el otro extremo, están las que podríamos llamar pruebas de caja blanca. En ellas necesitamos una detallada descripción del sistema en cuestión para poder realizar las mediciones oportunas de consumo y liberación de recursos en los distintos elementos que lo componen: Proxy, LDAP, Servidor Web, Servidor de Aplicaciones, Gestor de bases de datos, etc.

3.1.5. Caracterización de la carga de trabajo.

La demanda de trabajo a la que va a estar sometido un sistema y las características de la misma sólo pueden conocerse con exactitud cuando el sistema ha estado o está sometido a explotación dando un servicio real. Cuando esto no ocurre no tenemos más remedio que realizar estimaciones al respecto o recurrir a los datos proporcionados por algún sistema anterior o lo suficientemente similar como para que estos datos así obtenidos nos resulten útiles. La caracterización de la carga de trabajo, sea esta real o estimada, es un requisito indispensable para plantear las pruebas de prestaciones.

Cuando es preciso realizar estimaciones desde cero sin ningún tipo de datos iniciales al respecto, el departamento de Marketing responsable de la empresa a la que va dirigido el sistema suele ser de bastante utilidad.

Pero la situación ideal es, como ya decíamos, cuando disponemos de datos procedentes de una anterior instancia de la aplicación que ha pasado por un periodo real de explotación. Nuestro principal aliado aquí serán los ficheros de Log dejados por el sistema y las numerosas utilidades existentes para procesarlos y extraer datos útiles a partir de los mismos. Datos sobre visitas, franjas horarias a las que se producen las mismas, sistemas operativos utilizados, navegadores, versiones de los mismos, procedencia geográfica del visitante, etc.

3.2. Construcción.

El segundo apartado en nuestra metodología de pruebas de prestaciones consta de dos apartados:

- ❑ Diseño de la carga de trabajo.
- ❑ Adecuación de la configuración.

3.2.1. Diseño de la carga de trabajo.

El diseño de la carga de trabajo es una de las tareas más importantes en la realización de las pruebas de prestaciones. Un diseño erróneo puede llevarnos a tomar conclusiones equivocadas, positiva o negativamente, respecto al rendimiento de nuestro sistema. El diseño de la carga de trabajo puede orientarse de dos formas diferentes:

- ❑ **Diseño realista de la carga de trabajo.** Persigue realizar un modelado lo más exacto posible de la carga real de trabajo a la que se verá sometido el sistema para anticipar de la mejor forma posible la respuesta real que tendrá el mismo cuando pasé a explotación.
- ❑ **Diseño sintético de la carga de trabajo.** Recreamos, en este caso, una carga de trabajo totalmente irreal pero orientada con una finalidad claramente definida: estudiar la respuesta de alguna finalidad muy concreta, aislar un problema conocido de nuestro sistema, etc.

Los principales errores que debemos evitar cometer durante el diseño de la carga de trabajo son los siguientes:

- ❑ Representar sólo la carga media del sistema, por ejemplo tomar el número medio de transacciones por día para generar la carga de trabajo durante una hora ignorando los picos temporales de trabajo.
 - ❑ Ignorar los efectos de la caché olvidando personalizar la carga para que los distintos usuarios virtuales involucrados en la prueba accedan a datos diferentes del sistema.
 - ❑ Ignorar los bloqueos de recursos. Al igual que en el caso anterior, si no personalizamos la carga para que los distintos usuarios virtuales accedan a datos diferentes nuestro sistema puede verse afectado por numerosos bloqueos de acceso exclusivo a datos que arrojarán tiempos de respuesta erróneos.
 - ❑ Usar inapropiadamente los controles de que disponemos para variar la carga aplicada al sistema, a saber: el número de usuarios virtuales concurrentes, los tiempos de espera (*think times*) de los mismos y el número de recursos demandado al servidor por cada usuario. La forma más realista de variar la carga aplicada es mediante el número de usuarios virtuales concurrentes, pero también la más costosa en cuanto a recursos
-

hardware necesarios para la simulación y costes de licencias de la herramienta de pruebas. Variar los tiempos de espera entre operaciones de los usuarios es el método más sencillo y utilizado, pero hemos de tener en cuenta al aplicarlo que la concurrencia simultánea de usuarios es un factor muy importante a la hora de modelar los problemas del entorno real. Por último, variar la demanda de recursos por parte de los usuarios es el peor ajuste y el menos representativo de la realidad.

La mejor forma de diseñar la carga de trabajo apropiada es indagar en el perfil de los usuarios cuya actividad queremos reproducir. Veamos ahora las principales variables que definen el perfil de cada uno de dichos usuarios:

- ❑ **Actividad.** Evidentemente, el factor principal que determina el perfil del usuario es la actividad que este realiza contra el sistema. Por ejemplo, si sabemos que sólo 1 de cada 20 usuarios que entran en nuestro sistema entra a consultar determinada información, esa es exactamente la proporción que debemos de guardar a la hora de modelar la carga, a menos que nuestro objetivo sea precisamente monitorizar un determinado comportamiento en las consultas a dicha información.
 - ❑ **Tiempos de espera.** El tiempo que cada usuario se toma entre cada actuación en nuestro sistema tiene un significativo impacto en las prestaciones del mismo. Evidentemente, no provoca la misma carga el usuario que conoce a la perfección nuestro sistema y pulsa compulsivamente los enlaces antes de que las páginas se carguen por completo que el usuario que mantiene la aplicación en segundo plano y la consulta y actualiza casualmente.
 - ❑ **Puntos de sincronización.** ¿Cómo reproducimos un problema ya tipificado en nuestro sistema que sólo ocurre cuando un determinado número de usuarios coinciden en un determinado enlace? No hay problema: aunque la actividad de un usuario es por regla general asíncrona, las principales herramientas de modelado de carga poseen la forma de sincronizar, si así lo deseamos, la actuación de los usuarios virtuales en un determinado punto de su ejecución.
 - ❑ **Cadencias de conexión.** La forma en que los usuarios entran en el sistema también puede influir en la respuesta del mismo: no provocan los mismos efectos 100 usuarios que van entrando escalonadamente en nuestro sistema a lo largo del día que los mismos 100 usuarios que se conectan casi simultáneamente en un determinado momento.
 - ❑ **Patrones de uso.** Tampoco podemos olvidar a la hora de diseñar nuestro modelo de carga la distribución horaria con la que se suelen conectar habitualmente los usuarios del sistema bajo pruebas.
 - ❑ **Plataformas.** Los distintos clientes (emuladores de terminal, navegadores, etc.) y los sistemas operativos desde los que se ejecutan
-

(NT, Win32, Win16, Linux, etc.) pueden provocar diferencias en el tráfico generado hacia y desde nuestro sistema.

- ❑ **Preferencias.** En relación con el punto anterior, dos clientes con el mismo modelo de terminal, sistema operativo y navegador pueden ocasionar tráficos muy diferentes según la configuración de los mismos. Algunos ejemplos significativos y que no podemos olvidar son, por ejemplo, el número de tareas http de conexión, el tamaño de la caché, los diversos parámetros que afectan a la configuración de la seguridad del sistema cliente (cookies, profundidad del cifrado, etc.) y la posibilidad que tienen algunos navegadores de no descargar las imágenes y archivos multimedia.
 - ❑ **Direcciones IP.** ¿Qué ocurre cuando nuestro sistema en pruebas posee un balanceador de carga? La mayoría de estos sistemas utilizan la dirección IP del cliente (normalmente mediante una rápida función *hash*) para encaminar el tráfico hacia uno de los servidores balanceados. Si generamos todo el tráfico simulado desde un único ordenador cliente todas las peticiones irán a parar al mismo servidor con lo que, en el supuesto de que nuestro sistema posea tres diferente, estamos cargando solamente a uno de ellos mientras que los otros dos permanecen intocados. Algunas herramientas de generación de carga poseen lo que se denomina IP *spoofing*. Mediante esta técnica, se le proporciona a la herramienta una colección de direcciones IP válidas y debidamente reservadas en nuestra red para evitar conflictos, y ella se encargará de ‘engañar’ a los balanceadores de carga haciéndoles creer que cada una de las sesiones que abre contra el sistema es generada por un cliente con una dirección IP diferente.
 - ❑ **Velocidad de acceso.** No hace falta explicar la variedad de conexiones diferentes en función de su tecnología, velocidad de acceso y ancho de banda disponible y la repercusión que esto tiene en las prestaciones y la ocupación de recursos de un sistema.
 - ❑ **Proveedor de conexión.** En el caso de que nuestro sistema en pruebas sea una aplicación accesible desde Internet tenemos que valorar la calidad de servicio que prestan los diversos ISP (*Internet Service Provider*). Esta tarea se escapa de los márgenes posibles de afrontar en prácticamente todas las pruebas que podamos hacer, pero podemos hacer una aproximación tratando de averiguar los proveedores a través de los cuales se conecta un mayor porcentaje de usuarios a nuestro sistema e intentar modelar con esta información.
 - ❑ **Localización geográfica.** Si nuestro sistema es accesible desde distintos puntos de una WAN tenemos que suponer de antemano que los tiempos de respuesta serán diferentes dependiendo del punto de entrada en el mismo. Debemos de hacer esta suposición a pesar de que todos los puntos de entrada sean simétricos en cuanto a características se refiere: puede que una línea de RDSI desde
-

Barcelona a Madrid no nos proporcione las mismas prestaciones que otra idéntica desde Málaga hasta Madrid.

- ❑ **Ruido de fondo.** A menos que nuestro sistema trabaje en un entorno totalmente aislado, existen un gran número de variables que influyen en el mismo y repercuten en su rendimiento: puede que los servidores no sean dedicados, que realicen periódicas tareas de mantenimiento con grandes consumos de recursos, que exista otro tráfico en la red, etc.

Evidentemente en la mayoría de los casos no podemos tener en consideración todos estos factores. Al acometer unas pruebas de prestaciones debemos de detectar en primer lugar cuales de ellos son los que en realidad nos afectan y en una segunda fase cuales de estos son los que vamos a utilizar para nuestro modelo.

Tenemos un último problema en la construcción de nuestra carga simulada. La personalización de la misma. A veces es trivial encontrar en el tráfico grabado por la herramienta los distintos parámetros, códigos de usuario, de artículos, etc. Otras veces no es tan fácil. Una forma bastante sencilla de lograr esto es con una herramienta de comparación de archivos como por ejemplo *Windiff.exe*. El método consiste en grabar dos sesiones exactamente iguales con la única salvedad de un cierto parámetro que queremos localizar. Una posterior comparación mediante esta herramienta nos permitirá localizar donde se esconde el parámetro que buscamos.

3.2.2. Adecuación de la configuración.

A veces es preciso modificar ciertos parámetros de nuestro sistema sólo y exclusivamente para poder llevar a cabo con éxito las pruebas de prestaciones. En los sistemas Web, por ejemplo, es particularmente importante el máximo número de sesiones y el *time-out* con que caducan las mismas.

Imaginemos que estamos probando un sistema que usa un servidor iPlanet 4.1. Supongamos, asimismo, que tiene los siguientes parámetros de configuración:

- ❑ **maxSessions=1000**
- ❑ **timeOut=900**
- ❑ **reapInterval=600**

Estamos interesados en hacer unas duras pruebas de estrés sobre este sistema. Para ello hemos caracterizado un ciclo de trabajo muy corto, de apenas 10 segundos en vacío, que realiza un elevado consumo de recursos en el servidor. Si lanzamos inicialmente 15 usuarios virtuales y los tiempos de respuesta se mantienen más o menos constantes durante los primeros 15 minutos, consumiremos aproximadamente 1350 sesiones antes de que transcurran los 900 segundos a partir de los cuales caducan las primeras sesiones. Mientras que las primeras sesiones no sean liberadas

no podremos iniciar unas nuevas y esto no ocurrirá hasta que no venza el segundo intervalo de ‘cosecha’, es decir, a los 20 minutos. Este es un claro ejemplo de un caso en el que quizás la configuración del sistema sea adecuada para su explotación, pero es necesario adecuarla para el correcto funcionamiento de las pruebas de prestaciones.

3.3. Ejecución.

Ya en la fase número tres de nuestra metodología, trataremos los siguientes puntos:

- Monitorización
- Validación de los resultados
- Margen de error

3.3.1. Monitorización.

Durante la sesión de pruebas es preciso recoger una serie de mediciones que nos permitan realizar una detallada descripción de lo que está ocurriendo en el sistema en cada momento y como reacciona este en función de la carga introducida. Dichas medidas se pueden agrupar en dos bloques claramente diferenciados de los que ya hablamos en el punto 3.1.2. de este documento dedicado a las Métricas:

- Medidas de la calidad de servicio ofrecida por el sistema a los usuarios
- Medidas relativas al consumo de recursos del sistema.

Para las medidas relativas a la calidad de servicio se suele confiar en las proporcionadas por la propia herramienta de simulación de carga que suele proporcionar una amplia gama de parámetros y elabora automáticamente la practica totalidad de los estadísticos que nos serán útiles para la interpretación de los mismos.

Para las medidas relativas al consumo de recursos, las posibilidades ofrecidas por las herramientas suelen ser mucho más pobres, sobre todo cuando estamos trabajando con máquinas Solaris, Linux o cualquier otro clónico de UNIX.

En el caso de tratar de monitorizar un servidor NT, las herramientas de prestaciones suele ‘conectar’ directamente con el motor del sistema operativo proporcionándonos de forma gráfica y/o analítica los mismos datos que este presenta a través de su utilidad de monitorización. Es preciso, por supuesto, conocer un usuario del servidor que posea los privilegios adecuados para realizar esta monitorización remota.

En el caso de máquinas UNIX, las herramientas de prestaciones nos suelen presentar un subconjunto de datos tomados de la ejecución remota de comandos que recogen estadísticas publicadas por el sistema con *rmstat* o *rexec:vmstat* o *iostat*. En el primer caso es preciso que el sistema operativo esté configurado para publicar esos

datos a través de un determinado puerto. En el segundo es preciso conocer un usuario con los privilegios adecuados. No obstante, no tenemos porque conformarnos con estas migajas cuando disponemos de toda una amplia colección de comandos que nos permiten recoger datos del sistema operativo. Algunos de ellos son, por ejemplo: *sar*, *iostat*, *vmstat*, *ps*, *df*, etc.

No es el propósito de este documento enseñar UNIX y la utilización de estos comandos es trivial. Para conocer la totalidad de opciones de que disponen cada uno de ellos basta con consultar el manual en línea del sistema operativo.

3.3.2. Validación de los resultados.

En general, los resultados de las pruebas de prestaciones son validos cuando no están influenciados (o lo están sólo hasta límites aceptables) por la carga en si que provoca la simulación en las propias máquinas encargadas de generar la carga de trabajo, introduciendo por tanto un cierto retraso en la generación de dicha carga de trabajo e interfiriendo en los resultados de la prueba.

Son tres los parámetros que podemos medir efectivamente para asegurarnos de que las máquinas encargadas de generar la simulación no están trabajando en condiciones de sobrecarga:

- ❑ **Utilización de la CPU.** Como adelantábamos en el punto 2.2 de este texto, una utilización por debajo del 70% del tiempo de CPU indica un nivel aceptable mientras que una utilización por encima del 90% indica que la máquina está sobrecargada.
- ❑ **Utilización de memoria.** Igualmente, una utilización de memoria por encima del 150% de la memoria física nos indica que la actividad de *swapping* que está realizando la máquina cliente invalida los resultados de la prueba. Valores por debajo de 125% son aceptables.
- ❑ **Capacidad de reacción (*Responsiveness*).** Define la velocidad con la que el sistema cliente es capaz de responder ante un evento. Es el único parámetro que nos es inmediato de medir. La mejor forma de hacerlo es definir como una transacción un tiempo de espera de, por ejemplo, un segundo en el que la máquina cliente no haga nada. Cuando recojamos los resultados de la prueba analizaremos los resultados de esta transacción: Si los valores devueltos están por encima de 1,2 segundos consideraremos que los resultados de la prueba no son válidos.

Existen dos factores adicionales, que pueden invalidar los resultados de nuestras pruebas de prestaciones:

- ❑ **Ancho de banda consumido.** Como vimos en el punto 2.3, el ancho de banda de que disponemos para las pruebas está limitado por un porcentaje inferior a la capacidad teórica de nuestra red. Si durante las pruebas superamos el ancho de banda de que disponemos estamos introduciendo un retraso adicional en la respuesta de nuestro sistema.
- ❑ **Tasa de errores.** Una elevada cantidad de errores durante las pruebas puede significar varias cosas: problemas con los *drivers* de comunicaciones de las máquinas clientes, un sistema con problemas en su funcionalidad que lo invalida para ser sometido a pruebas de prestaciones o algún error en la grabación o personalización de la carga a la que se somete al sistema. Por lo general, no consideraremos admisible un porcentaje de errores superior al 1% del total de transacciones ejecutadas durante las pruebas.

3.3.3. Margen de error.

Una vez hechas todas las consideraciones hasta el momento tenemos que hacer una preocupante afirmación. No importa lo cuidadosos y detallistas que seamos que nuestras pruebas jamás reproducirán con total verosimilitud la carga real a la que se verá sometido el sistema cuando pase a explotación. ¿Podemos al menos calcular el margen de error con el que debemos de contar?

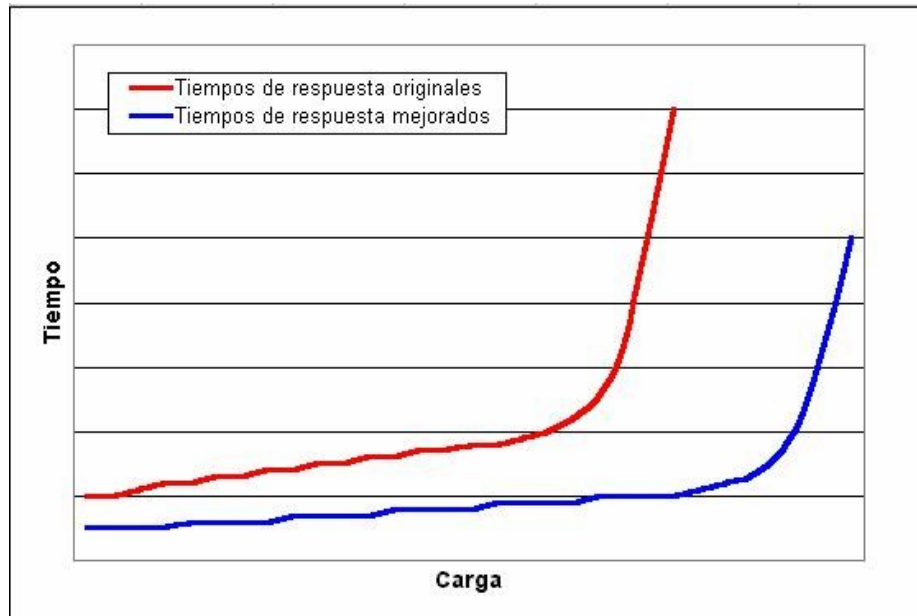
El mejor método consiste en visitar todas las asunciones que hemos hecho a la hora de parametrizar nuestra carga de trabajo virtual y construir dos pruebas adicionales: una con los valores más pesimistas posibles y otra con los más optimistas. Contaremos, entonces, con una familia de resultados con la que calcular el margen de error de nuestras pruebas.

3.4. Análisis e interpretación de resultados.

Ya en la fase final de nuestra metodología, analizaremos en este apartado la forma típica de la curva de tiempos de respuesta en función de la carga que suele presentar un sistema, las formas básicas de atacarla para mejorar sus prestaciones y un par de ejemplos en los que gráficamente intentaremos tipificar algunos de los casos que más frecuentemente nos encontramos durante las pruebas de prestaciones.

3.4.1. Formas de mejorar el rendimiento.

Sin ánimos de entrar en consideraciones de diseño (páginas web de menor tamaño, con menor número de objetos, índices óptimos en las bases de datos, etc.) la finalidad de esta sección es estudiar la curva de respuesta típica de un sistema y analizar sucintamente como podemos mejorarla.

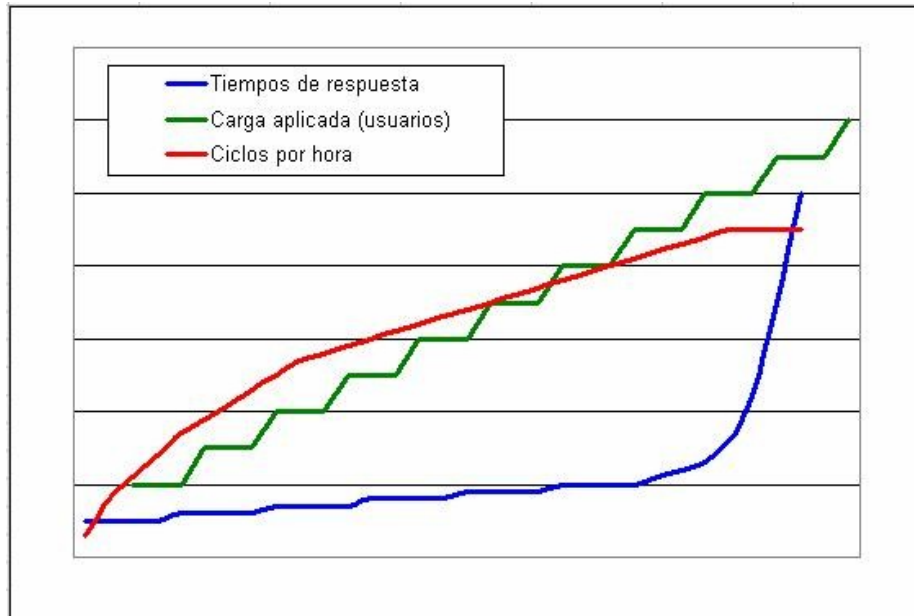


Como podemos ver en la imagen anterior, existen tres formas básicas de mejorar la curva de tiempos de respuesta de nuestro sistema:

- ❑ Reducir el punto de entrada, es decir el tiempo de respuesta 'en vacío' del sistema y por tanto de todas las actuaciones del usuario contra el. Esto se consigue generalmente estudiando cuales son los ciclos más frecuentes o los más costosos en tiempo y actuando directamente contra ellos.
- ❑ Reducir el grado de la pendiente de degradación del sistema. Por lo general, un buen método para lograr esto es aumentar el ancho de banda de conexión de nuestro sistema para que la degradación de los tiempos de respuesta sea más lenta.
- ❑ Desplazar el 'codo' o punto a partir del cual los tiempos de respuesta se hacen exponenciales. Normalmente ese punto en el que los tiempos de respuesta se hacen intratables viene provocado por uno o más cuellos de botella en el sistema (utilización de la CPU, de la memoria, etc.) Tenemos que estudiar detenidamente los datos provenientes de la monitorización del sistema durante las pruebas, detectar donde se encuentran las carencias del mismo y actuar sobre ellas ampliándolas o actualizándolas.

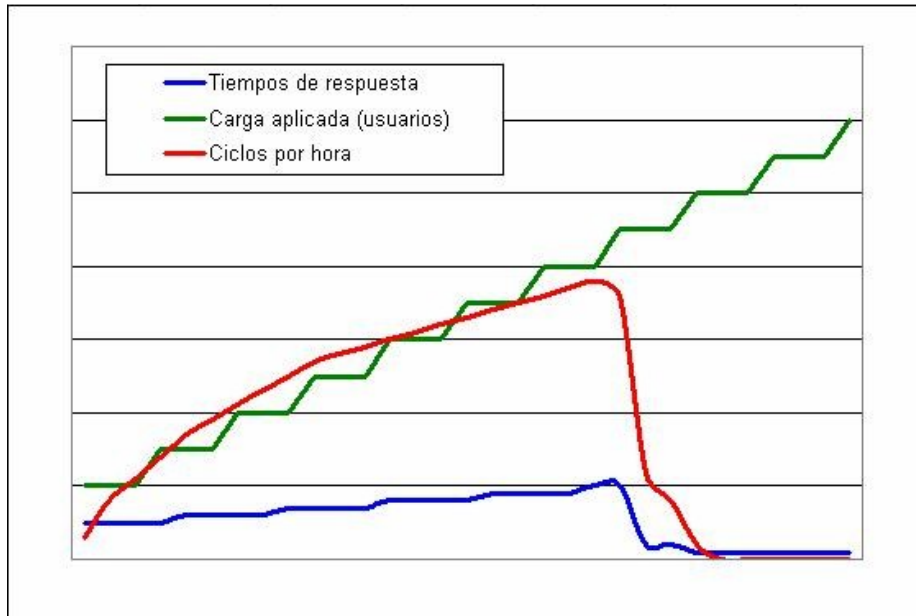
3.4.2. Ejemplos gráficos.

La colección de curvas que con más frecuencia nos encontraremos durante unas pruebas de prestaciones es como sigue:



Como podemos observar en ella, a pesar de que subimos progresivamente la carga (tipificada como número de usuarios virtuales con una caracterización particular) el número de ciclos de trabajo que estos ejecutan no sigue la misma tendencia casi lineal que llevan estos, sino que va decrementando su pendiente hasta que llega un momento que prácticamente se estabiliza y, a partir de aquí, puede incluso descender. Es en este punto donde hemos encontrado el 'codo' de nuestra curva y donde debemos empezar a buscar el cuello de botella que ha provocado esta situación

En el siguiente gráfico podemos ver un ejemplo de lo que ocurre cuando durante las pruebas de prestaciones detectamos el fallo de algún servicio. En este caso, las prestaciones del sistema (medidas como ciclos de trabajo ejecutados) caen velozmente en un punto determinado. Puede resultar engañoso que también caen los tiempos de respuesta, pero esto también es lógico: la transacción definida termina rápidamente con la devolución de un error.



Para averiguar que está ocurriendo en este caso, una primera aproximación la podemos realizar intentando reproducir el ciclo de trabajo definido de forma manual con nuestro navegador y ver donde se detiene el mismo. En un segundo paso, deberíamos de investigar en los ficheros de *Log* de acceso y error de todos los sistemas involucrados (proxy, LDAP, servidor Web, servidor de aplicaciones, motor de bases de datos, etc.) o directamente en aquellos en los que sospechemos que se ha producido el error.

4. Otros conceptos relacionados.

Escalabilidad, Disponibilidad y Fiabilidad son tres conceptos íntimamente relacionados con el tema que nos atañe por dos motivos principales: primero porque los tres deberían de tenerse en cuenta a la hora de hablar de sistemas Web y segundo porque, como veremos someramente a continuación, todos los términos y herramientas que hemos utilizado hasta ahora nos serán muy útiles para afrontarlos.

4.1. Escalabilidad.

La escalabilidad de un sistema puede definirse como la capacidad de crecimiento para adecuarse al futuro crecimiento del mismo. Es evidente que para realizar pruebas de escalabilidad nos son útiles las mismas herramientas y los mismos conceptos que se barajan en las pruebas de prestaciones.

4.2. Disponibilidad y Fiabilidad.

La disponibilidad de un sistema es una medida relativa a la preparación para su utilización, mientras que la fiabilidad es una medida relativa a su capacidad para mantenerse operativo en el tiempo. Ambas propiedades están englobadas dentro de un concepto mucho más amplio, la confiabilidad, que incluye además aspectos relativos a la seguridad del sistema, la confidencialidad y la integridad de los datos del mismo. Las pruebas de disponibilidad suelen ser variantes de las pruebas de estabilidad y, al igual que nos ocurría en el caso anterior, utilizan las mismas herramientas y manejan los mismos conceptos que las pruebas de prestaciones.

5. Conclusiones.

Como hemos podido ver, realizar unas pruebas de prestaciones es bien sencillo: basta con poseer una herramienta adecuada y, casi sin apenas necesidad de aprendizaje (pues su funcionamiento es bien sencillo e intuitivo) atacar al sistema cuyo rendimiento queremos probar.

Ahora bien: existe una gran diferencia entre realizar unas pruebas de prestaciones y que los resultados obtenidos sean representativos de la realidad y nos sirvan para algo.

En el presente documento hemos pretendido realizar un breve repaso por la mayoría de los conceptos involucrados en unas pruebas de prestaciones, intentando hacer ver la complejidad de los mismos y la casi absoluta necesidad de utilizar una metodología adecuada para su realización y una herramienta específica para su ejecución.

6. Bibliografía

1. S. Splaine y S. P. Jaskiel. *The Web Testing Handbook*. STQE Publishing.
 2. T. Kolish y T. Doyle. *Gain eConfidence: The e-Business Reliability Survival Guide*. Segue Press.
 3. E. Ambichi. *Very Large Scale Load Testing with SilkPerformer*. Segue Press.
 4. G. Conde. *Introducción a SilkPerformer y Pruebas de carga*. MTP.
 5. A. Sabiola. *Web Load Test Planning. Predicting how your Web Site will respond to stress*. STQE Magazine
 6. K. Johnson. *Mining Gold from Server Logs. Using existing site data to improve Web Testing*. STQE Magazine.
 7. B. Shea. *Avoiding Scalability Shock. Five Steps to Managing the Performance of E-business Applications*. STQE Magazine.
 8. A. Sabiola. *Response Time. Understanding and measuring performance test results*. STQE Magazine.
 9. E. J. Weyuker, F. I. Vokolos. *Experience with Performance Testing of Software Systems: Issues, an Approach, and Case Study*. IEEE Transactions on Software Engineering.
-