UNIDAD Nº 4: PRUEBAS DE DESEMPEÑO - PROPIEDADES EMERGENTES

➔ Pruebas de Desempeño

Verifica los requerimientos no funcionales del sistema. Lo esencial es definir:

- **Procedimientos de prueba**.
  - Ejemplo: Tiempo de respuesta → Simular la carga, tomar los tiempos de respuesta.
- **Criterios de aceptación**.
  - Ejemplo: Límite → El cliente lo valida si en el 90 % de los casos de pruebas se cumplen los requerimientos.
- **Características del ambiente**. Éstas hacen grandes diferencias en los requerimientos no funcionales.

¿Qué tan bien se desempeña tu sistema?

Desafortunadamente, muchas empresas liberan sus sistemas sin saber la respuesta. El desempeño es un aspecto fundamental en una aplicación, ya que si ésta no responde en el debido tiempo, se pueden perder clientes, o dañar la imagen ante los usuarios.

Para comenzar, es necesario introducir algunos términos y conceptos básicos; para ello, recurriremos a un ejemplo de la vida real: un supermercado.

En un supermercado hay clientes caminando entre los pasillos, que compran productos, y pagan en las cajas. Los supermercados generalmente tienen más clientes que empleados, ya que, a cada cliente, le toma tiempo seleccionar los productos, y por tal razón no necesitamos un cajero por cada cliente.

En el mundo del software, las **peticiones** a la aplicación representan a esos **clientes**, y ésta utiliza recursos (hardware y software) para manejar esas peticiones.

Carga es el término que representa a todos los **usuarios** de la aplicación en un determinado momento sin importar su actividad. Nos importa el número total de usuarios, no sólo los que están haciendo peticiones, ya que todos los usuarios consumen recursos. En un supermercado, los clientes ocupan un espacio físico, mientras que en una aplicación ocupan memoria u otro tipo de mecanismo para almacenar el estado de un cliente.

Después de que el supermercado abre, el tráfico de clientes no permanece constante durante el día, sino que tiene horas pico. Una carga de picos se refiere al número máximo de clientes dentro de un período de tiempo. Este número no es un promedio. **Al planear una prueba de desempeño, encontrar la carga pico y probar la aplicación contra esta carga es lo más importante.**

El **rendimiento** de procesamiento, o **throughput**, mide la **cantidad de peticiones servidas** por unidad de tiempo. En el caso del supermercado, suponiendo que cobrarle a cada cliente toma un minuto, y que hay cinco cajeros, el rendimiento será de cinco clientes por minuto. Esta métrica se caracteriza por tener un

límite superior, es decir, no importa cuántos clientes visiten el supermercado, el número máximo de clientes atendidos en un intervalo de tiempo permanecerá constante. Así mismo, si una aplicación recibe 50 peticiones por segundo, pero su rendimiento máximo es de 30, algunas peticiones tendrán que encolarse. Este límite típicamente representa un **cuello de botella** en la aplicación.

El **tiempo de respuesta** de una aplicación se refiere al tiempo desde que el usuario inicia una petición hasta que el resultado es visualizado. Hay veces en que en el supermercado hay que esperar en la cola hasta que una caja esté disponible. Este tiempo de espera se suma al tiempo requerido para realizar la compra. Por lo tanto, si el cliente espero cuatro minutos, más el minuto que requirió la transacción, el tiempo de respuesta sería de cinco minutos. Entender el **tiempo de respuesta** de una aplicación requiere entender la **relación entre carga, rendimiento de procesamiento, y el mismo tiempo de respuesta**. La aplicación alcanza su **rendimiento máximo** con cierta **cantidad de usuarios**. Más allá de ésta (**punto de saturación**), el rendimiento permanece constante.

Sin embargo, el tiempo de respuesta empieza a aumentar. La carga adicional espera a que los recursos saturados se liberen antes de que los pueda utilizar. *Entender cuántos usuarios simultáneos soporta la aplicación y cuanto aumentan los tiempos de respuesta en los momentos de mayor tráfico son las principales motivaciones para realizar una prueba de desempeño.*

Así que, **¿qué podemos hacer para mejorar el desempeño de la aplicación?** Básicamente, se pueden hacer dos cosas: **optimizar** o **escalar** la aplicación. La **optimización** se enfoca en **modificar** la aplicación para que consuma menos recursos, o para minimizar los cuellos de botella. Las técnicas de **escalamiento**, por otra parte, se enfocan en **añadir recursos**, por ejemplo, aumentando el número de servidores que hospedan la aplicación. En el escalamiento vertical se añaden más recursos al servidor, como CPU o memoria (el equivalente a añadir más cajeros en el supermercado), mientras que en el escalamiento horizontal, se añaden servidores para formar un cluster (el equivalente a añadir sucursales del supermercado).

Perfiles de aplicaciones

Una pregunta importante antes de planear una prueba de desempeño es: **¿Qué es lo que la aplicación necesita hacer bien?** Obviamente, los requerimientos de varios tipos de aplicaciones son diferentes, y en esa misma forma deben ser sus pruebas de desempeño.

Por ejemplo, una **aplicación financiera** por lo general maneja grandes volúmenes de usuarios e información confidencial y actualizada en tiempo real. Así que en este tipo de aplicaciones no se puede sacar gran provecho de caches de datos. Otra consideración importante es la seguridad. Los mecanismos de cifrado de datos, y otras medidas de seguridad como firewalls tienen un gran impacto en el desempeño. También es necesario registrar la actividad de los usuarios en bitácoras para auditoría o por regulaciones gubernamentales, lo cual impacta el procesamiento en la aplicación.

Por otro lado, las **aplicaciones tipo e-Commerce** son usadas por los clientes para ver y comprar productos, y usar servicios como estado del envío o cancelaciones. Aunque puede haber una gran cantidad de usuarios concurrentes, solo un pequeño porcentaje realiza una compra. Sin embargo, se debe manejar muy bien la memoria para evitar un consumo excesivo. En este caso, hay que asegurar las partes de la aplicación donde se realicen transacciones, aunque éstas normalmente representan una pequeña fracción del tráfico total.

Por último, en el caso de **aplicaciones de administración de información** interna en una empresa, un día típico para el usuario puede incluir revisar el estatus del inventario, registrar nuevos productos, o consultar reportes generados en el momento. Por lo general, se conoce la cantidad de usuarios. Las páginas no contienen muchos gráficos pesados, ya que son ayudas de navegación, más que contenido. Los usuarios de estas aplicaciones están dispuestos a aceptar un tiempo de respuesta más lento que los usuarios de otro tipo de aplicaciones, sin embargo, no por eso el desempeño pierde importancia.

Scripts de prueba

Los scripts dirigen la prueba de rendimiento ya que simulan una serie de peticiones que el usuario realiza cuando utiliza la aplicación. Para desarrollarlos se necesita entender el comportamiento de los usuarios. Representar de manera precisa a los usuarios en los scripts de la prueba tiene gran influencia en el rendimiento y carga de la misma prueba y sus resultados.

Durante la prueba, el mismo script es ejecutado varias veces simulando múltiples usuarios. Para simular las diferentes opciones presentadas por la aplicación, se utiliza un conjunto de scripts. Éstos forman un escenario de prueba.

Típicamente, se utiliza una herramienta para grabar las actividades de un usuario y generar el script automáticamente.

Las siguientes recomendaciones se deben tomar en cuenta para desarrollar scripts:

- Desarrollar scripts pequeños.
- Escribir scripts atómicos (funciones bien concretas que empiecen y terminen).
- Permitir datos dinámicos (que se pueda parametrizar el script para que represente varios usuarios).

Lo importante es generar una cantidad razonable de datos o selecciones dinámicas y no caer en la trampa de que los usuarios interactuarán con la aplicación justo como los diseñadores lo planearon. Un ejemplo claro, es la función de logout. Los usuarios no saben cómo salir (logout) de la aplicación, no importa que tan grande sea el botón para hacerlo. Por esto, no se puede depender de que los usuarios salgan correctamente del sistema, y se deben desarrollar los scripts tomándolo en cuenta.

Planeación de la prueba

Antes de empezar las pruebas de desempeño, es necesario realizar las **pruebas de funcionalidad e integración** para asegurase de que la aplicación funciona correctamente. El comportamiento errático puede exhibir mejor o peor desempeño que la versión correcta.

Comenzamos definiendo **objetivos claros** de desempeño para la aplicación. Ésta puede contener una infinidad de cuellos de botella, y si no hay objetivos claros, nunca sabremos cuando hemos llegado a un nivel aceptable de desempeño.

Decir que un caso de uso debe completarse en “alrededor” de cinco segundos no es muy útil. Se debe enunciar un valor exacto que permita verificar el desempeño de la aplicación. Los **objetivos deben ser flexibles** también, en el sentido en que puede haber variación entre ejecuciones. Por ejemplo, ¿es aceptable que el caso de uso se ejecute en tiempo el 90% de las veces? ¿Es aceptable que el tiempo de respuesta sea de 5 segundos (cuando se espera sea de 3) una de cada 10 veces?

Proceso y análisis

El éxito de la prueba depende de la recolección y análisis de datos y los cambios para mejorar el desempeño de la aplicación. El primer paso es ejecutar una prueba de desempeño simulando algunos usuarios y tal vez en un solo servidor. Durante la prueba hay que recolectar datos acerca de los clientes, los servidores y otros componentes usados.

Después, hay que analizar esos datos antes de seguir con la siguiente prueba. Sin embargo, se recomienda ejecutar la misma prueba dos o tres veces sin hacer cambios y comparar los resultados. ¿Las pruebas produjeron más o menos los mismos resultados? Si la respuesta es afirmativa, se puede proceder a las optimizaciones. En caso contrario (si la variación es más del 10%), hay que corregir esto antes de continuar.

Algunas posibles causas pueden ser:

- Probar en un ambiente compartido.
- Ejecuciones cortas. Hay que darle suficiente tiempo a la prueba, es decir, "calentar motores".
- Crecimiento en la base de datos. La aplicación probablemente creo y/o actualizo muchos registros durante la prueba y esto hace que se degrade el desempeño. En este caso, hay que considerar scripts que limpien la base entre ejecuciones.

Después de analizar los datos y posiblemente identificar cuellos de botella, es tiempo de optimizar. Idealmente, se debe hacer solo un cambio, ejecutar la prueba de nuevo y analizar los datos antes de hacer otro ajuste o seguir con otra prueba. Esto ayuda a medir de forma aislada, el impacto del cambio en la aplicación.

Datos

La prueba de desempeño es tan buena como los datos que produce. Los siguientes datos son aquellos que siempre se deben recolectar en la prueba de desempeño:

- Utilización del CPU.
- Datos de la Red (tráfico generado).
- Monitoreo de Servidores (herramientas que permiten monitorear sus recursos internos, pueden impactar en el desempeño, por lo que hay que utilizarlas con moderación.

TIPOS DE PRUEBAS DE DESEMPEÑO

- Pruebas de **estrés** (esfuerzo) → Esta prueba implica someter al sistema a grandes cargas (un pico de volúmenes de datos por un corto período).
- Pruebas de **volumen** → Implica someter al sistema a grandes volúmenes de datos.
- Pruebas de **facilidad de uso** → Tratar de encontrar problemas en la facilidad de uso. Este tipo de pruebas son muy importantes, sino se hacen pueden terminar en un sistema inutilizable.

- Por ejemplo: ¿Han sido ajustadas las interfaces a la inteligencia y nivel educacional del usuario final? ¿Son las salidas significativas para cada usuario? ¿Son directos los mensajes de error y fácilmente interpretados?

- Pruebas de **seguridad** → La idea es encontrar casos de prueba que burlen los controles de seguridad del sistema. Una forma puede ser, buscar problemas conocidos en sistemas similares y probarlos en el sistema bajo test.
- Pruebas de **rendimiento** → Generar casos de prueba para demostrar que el sistema no cumple con sus especificaciones de rendimiento.
  - Por ejemplo: Tiempos de respuestas en sistemas interactivos. Tiempo máximo de ejecución de alguna tarea, tareas simultáneas, etc.
- Pruebas de **configuración** → Se prueba el sistema con distintas configuraciones de hardware y software.
  - Por ejemplo: Se instala y prueba en equipos con distintos sistemas operativos y distintas capacidades de procesamiento.
- Pruebas de **compatibilidad** → Son necesarias cuando el sistema tiene interfaces con otros sistemas.
- Pruebas de **facilidad de instalación** → Se prueban las distintas formas de instalar el sistema.
- Pruebas de **recuperación** → Se simulan o crean fallas para testear la recuperación del sistema.
- Pruebas de **confiabilidad** → El propósito de toda prueba es aumentar la confiabilidad del sistema. Es necesario diseñar casos de pruebas especiales cuando se desea probar requerimientos especiales de confiabilidad.
  - Por ejemplo: El sistema tiene un valor medio de tiempo entre fallas de 20 horas (en condiciones normales).
- Pruebas de **documentación** → La documentación del usuario debe ser motivo de inspección, controlando su exactitud y claridad.
  - Por ejemplo: Probar cada uno de los casos de uso y obtener los resultados esperados, o bien encontrar ejemplos en la documentación que no funcionan en el sistema.
- Pruebas de **regresión** → Después que se realizan modificaciones, se verifica que lo que ya estaba probado sigue funcionando.
- Pruebas **piloto** → Se pone a funcionar el sistema en producción de forma localizada. Produce un menor impacto de las fallas.

CONCLUSIONES

Las pruebas de desempeño son una actividad importante del desarrollo de sistemas. Desafortunadamente, su uso no es muy extendido. Por medio de una analogía, vimos que básicamente los mismos problemas encontrados en el mundo real, también evitan que las aplicaciones de software alcance su máximo potencial en cuanto a desempeño. Finalmente, cabe recalcar que una prueba de desempeño es un proceso iterativo. **Se prueba, recolectan datos, se verifican, analizan y se optimiza. Este proceso ocurre repetidamente hasta que se alcancen los objetivos de todos los escenarios de prueba.**